

Windows Azure and SQL Database Tutorials

Jonathan Gao

Step-by-Step



Microsoft

Windows Azure and SQL Database Tutorials

Jonathan Gao

Summary: These Windows Azure and SQL Database (formerly SQL Azure) tutorials are designed for beginners who have some .NET development experience. Using a common scenario, each tutorial introduces one or two Windows Azure features or components. Even though each tutorial builds upon the previous ones, the tutorials are self-contained and can be used without completing the previous tutorials.

Category: Step-by-Step

Applies to: Windows Azure SQL Database

Source: TechNet Wiki ([link to source content](#))

E-book publication date: November 2012

Copyright © 2012 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft and the trademarks listed at <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Contents

Before You Begin the Tutorials	5
Set Up the Development Environment.....	5
Installing the Tutorial Files	5
Installing the Code Snippets.....	5
Using the Code Snippets	6
Provisioning Windows Azure.....	6
Tutorial 1: Using Windows Azure Web Role and Windows Azure Table Service	7
Overview	7
Lesson 1: Create a Windows Azure Visual Studio Project.....	11
Lesson 2: Create a Data Source	13
Lesson 3: Create a Web Role to Display the Message Board and Process User Input	19
Lesson 4: Test the Application and Deploy the Application.....	24
Tutorial 2: Using SQL Database	29
Overview	29
Lesson 1: Prepare SQL Database Account.....	33
Lesson 2: Modify the Application to use SQL Database	37
Lesson 3: Test and Deploy the Application	42
Tutorial 2.1: Create an OData Service	46
Overview	46
Lesson 1: Modify the Application to Define the Entity Framework Data Model	47
Lesson 2: Modify the Application to Add the Data Service.....	50
Lesson 3: Test and Deploy the Application	52
Tutorial 3: Using Windows Azure Blob Service	57
Overview	57
Lesson 1: Modify the Application to Use the Blob Service	61
Lesson 2: Test and Deploy the Application	67
Tutorial 4: Using Windows Azure Worker Role and Windows Azure Queue	72
Overview	72
Lesson 1: Modify the Message Data Model.....	76
Lesson 2: Modify the Web Role.....	81
Lesson 3: Add a Worker Role for Background Processing	83
Lesson 4: Test and Deploy the Application	86

These Windows Azure and SQL Database (formerly SQL Azure) tutorials are designed for beginners who have some .NET development experience. Using a common scenario, each tutorial introduces one or two Windows Azure features or components. Even though each tutorial builds upon the previous ones, the tutorials are self-contained and can be used without completing the previous tutorials.

Windows Azure is an open cloud platform that enables you to quickly build, deploy and manage applications across a global network of Microsoft-managed datacenters. You can build applications using any language, tool or framework.

Windows Azure provides three options for executing applications. These options can be used separately or combined.

- **Virtual Machine:** Virtual Machines (VM) provides the ability to create a virtual machine on demand. You can use the virtual machine as a development computer, or as a server to run your applications. Once you have a virtual machine created, you can deploy your existing applications to it without engaging developers to make code changes. The downside of using virtual machines is that you are responsible to administering the VMs.
- **Web Site:** The Web Site option offers a managed web environment using Internet Information Services (IIS). You can move an existing IIS website into Windows Azure Web Sites unchanged, or you can create a new one directly in the cloud. Once a website is running, you can add or remove instances dynamically, relying on Web Sites to load balance requests across them.
- **Cloud Service:** Cloud Service is designed expressly to support scalable, reliable, and low-admin applications, and it's an example of what's commonly called Platform as a Service (PaaS). To use it, you create an application using the technology you choose, such as C#, Java, PHP, Python, Node.js, or something else. Your code then executes in virtual machines (referred to as instances) running a version of Windows Server.

This set of tutorials demonstrates how to create cloud services:

- [**Tutorial 1: Using Windows Azure Web Role and Windows Azure Table Service**](#) This first tutorial goes beyond a "Hello World" sample to show you how to use a Web role and table service.
- [**Tutorial 2: Using SQL Database**](#) Based on the tutorial 1 application, this tutorial replaces the table service with SQL Database.
- [**Tutorial 2.1: Create an OData Service**](#) This tutorial extends tutorial 2 to add an OData service to access and change data in the SQL Database created in tutorial 2.
- [**Tutorial 3: Using Windows Azure Blob Service**](#) In this tutorial, blob storage is added to the tutorial 1 application for storing binary images.
- [**Tutorial 4: Using Windows Azure Worker Role and Windows Azure Queue Service**](#) This tutorial adds a worker role for background processing, and a queue is used to facilitate communication between the two roles.

Before You Begin the Tutorials

Set Up the Development Environment

Before you can begin developing your Windows Azure application, you need to get the tools and set-up your development environment.

For configuration instructions, see *How to Prepare the Windows Azure Compute Emulator* at <http://msdn.microsoft.com/en-us/library/gg433136.aspx>.

Installing the Tutorial Files

Tutorial files can be found in the AzureTutorials\TutorialFiles folder. Extract the files to the C root directory. The tutorial files contain the following folders:

- **CodeSnippets** contains the code snippets used for developing the application.
- **CompletedSolution** contains the completed solution. If you run into a problem, you can compare your code with the code in this folder.
- **GolferMessageBoard** contains the application you created in the previous tutorials. This is the application that the current application is based on.
- **TutorialFiles** contains the files that you will need to import into your project.

Note: This release of the tutorials only provide the C# code samples.

Installing the Code Snippets

For convenience, much of the code that you need to type while executing the tutorials is available as Visual Studio code snippets.

After installing the tutorial files, you can find the code snippets in the **C:\AzureTutorials\Tutorial[#]\CodeSnippets** folder.

You must install the code snippets manually by copying the contents from one folder to another for each of the 2 folders listed in the table below.

Copy from	Paste to
C:\AzureTutorials\Tutorial[#]\CodeSnippets\Visual Web Developer	..\My Documents\Visual Studio 2010\Code Snippets\Visual Web Developer\My Code Snippets

C:\AzureTutorials\Tutorial[#]\CodeSnippets\Visual C#	..\My Documents\Visual Studio 2010\Code Snippets\Visual C#\My Code Snippets
--	---

Using the Code Snippets

With code snippets, you have all the code you need at your fingertips. You can use the following procedure to add a code snippet:

1. Right-click where you want to insert the code snippet, and then click **Insert Snippet**.
2. Click **My Code Snippets** or **My HTML Snippets**, click **Windows Azure Tutorials**, and then click the name of the code snippet you want to insert.

Provisioning Windows Azure

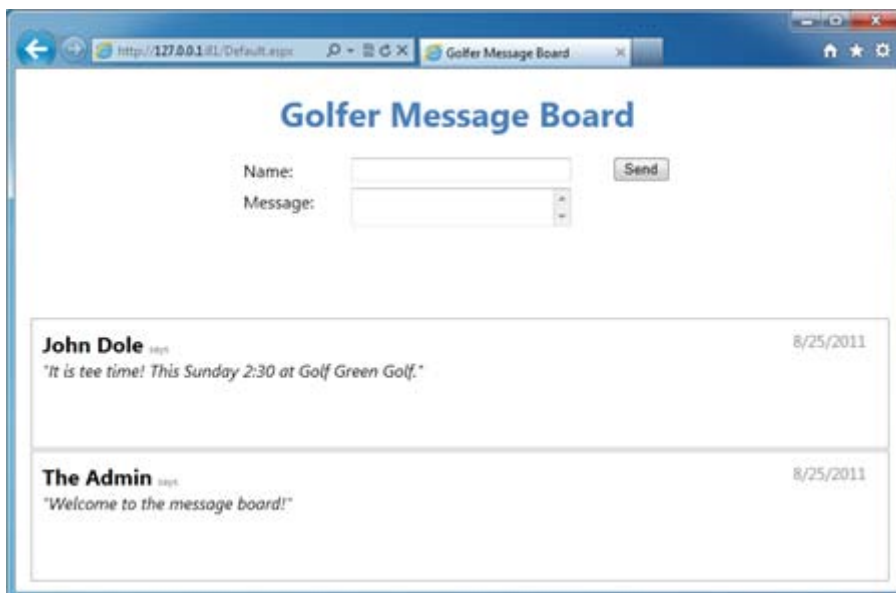
In order to complete the deployment lesson in each tutorial, you can either get a [trial](#) account or purchase an account. A Windows Azure account is not required for developing a Windows Azure application. You can always test your application using the local compute emulator and storage emulator.

Tutorial 1: Using Windows Azure Web Role and Windows Azure Table Service

This tutorial demonstrates the process of creating a Windows Azure application and the process of deploying the application to a Windows Azure Cloud Service.

Overview

The application is a simple golfer message board web application for users to enter or view messages, it contains one [web role](#) (Web front-end), that allows golfers to view the message board and add new message entries. The application uses the [Windows Azure Table service](#) to store messages. Here is a screenshot of the application:



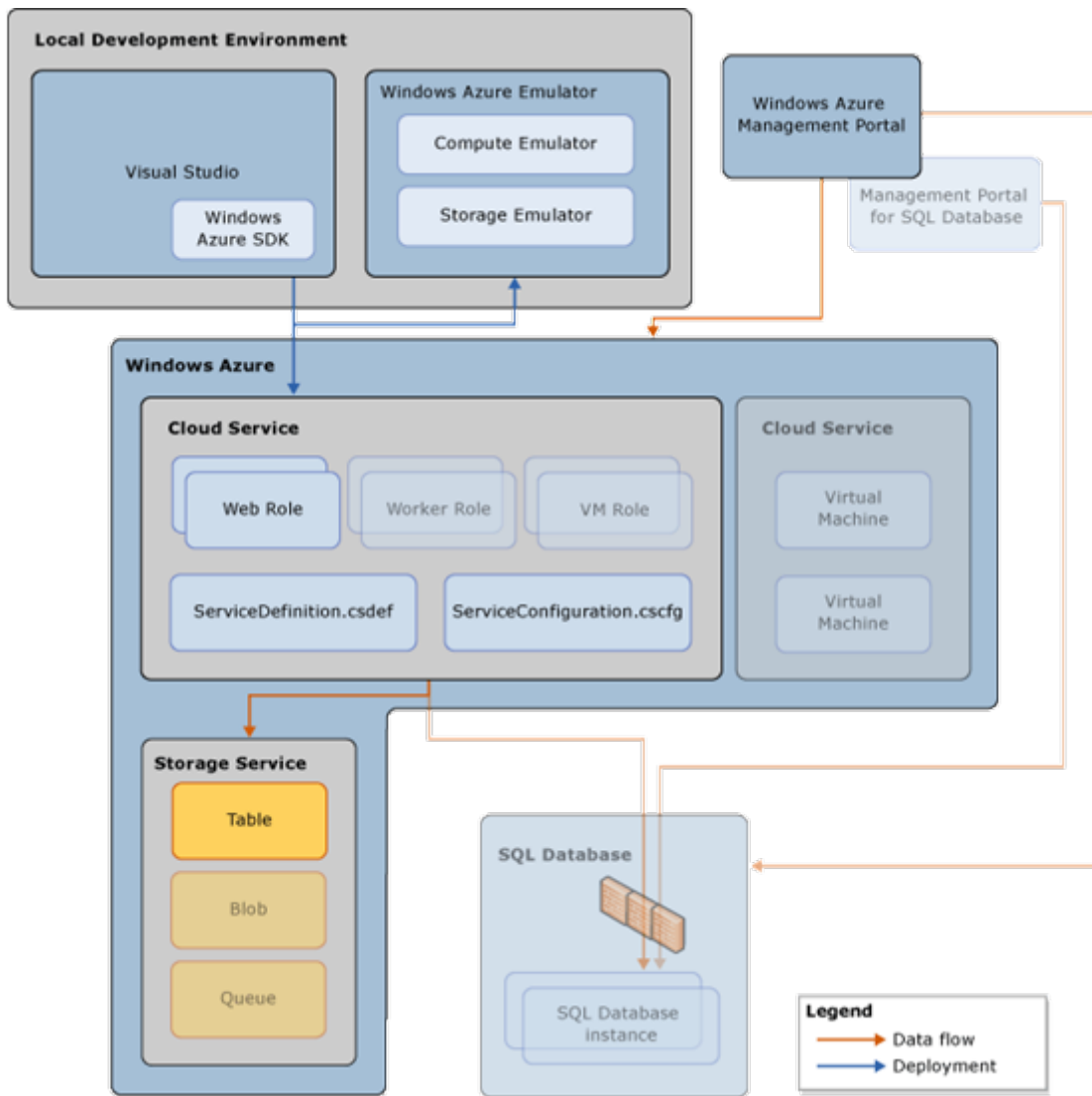
Objectives

In this tutorial, you will learn how to:

- Understand the process of developing a Windows Azure application
- Understand the process of deploying an application to Windows Azure
- Create a web role
- Use the Table service

Understanding the Architecture

The following diagram illustrates the development components and the runtime components involved in this tutorial:



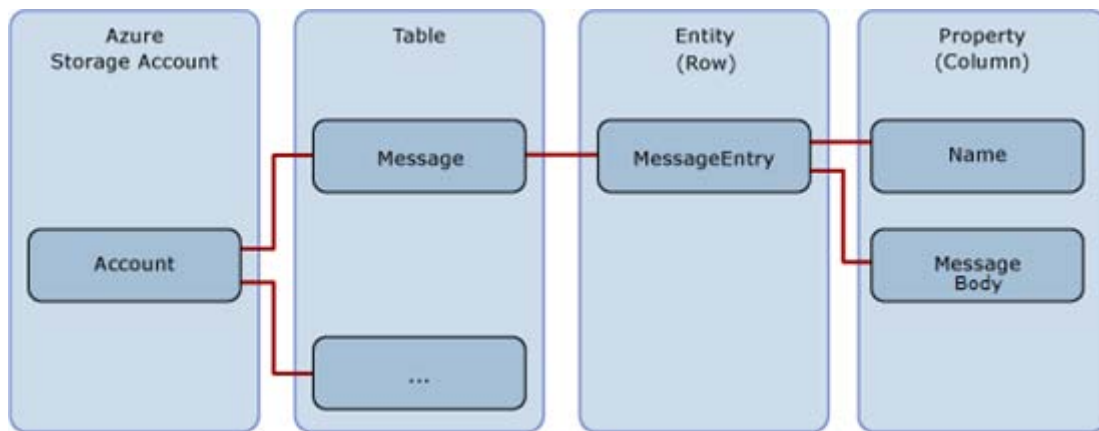
Note: The gray out components are the components that are not covered in this tutorial but in other tutorials.

- There are two environments: the Windows Azure runtime environment and the local development environment. The runtime environment is hosted in one of the Microsoft data centers.
- There are two main services within the runtime environment: Compute and Storage. Compute is the computational part of Windows Azure. Storage consists of three main parts: Table service, Blob service and Queue service. Other than the Windows Azure Storage Service, you can also use SQL Database to store data. In this tutorial, you will use Windows Azure Table service. [Tutorial 2](#) covers SQL Database; [tutorial 3](#) covers Blob service; and [tutorial 4](#) covers Queue service. For more information, see [Data Management](#).

- The Compute service is also referred as Cloud Service. It is the application you deploy on Windows Azure. Every Cloud Service can have web role and worker role. 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. A worker role is a role that is useful for generalized development, and may perform background processing for a web role. In this tutorial, you will create a web role project. For more information, see [What is a Cloud Service](#). [Tutorial 4](#) will introduce Worker role.
- You can use the [Windows Azure Management portal](#) to administrate Windows Azure platform resources.
- You develop the Windows Azure application using Visual Studio and Windows Azure SDK.
- You deploy the application to local Windows Azure Emulator for testing, and to Windows Azure.
- A Windows Azure project includes two configuration files: ServiceDefinition.csdef and ServiceConfiguration.cscfg. These files are packaged with your Windows Azure application and deployed to Windows Azure.

Understanding the Table Service Object Model

The Windows Azure Table service is structured storage in the cloud. Here is a diagram of the Table service data model:



An application must use a valid account to access Windows Azure Storage service. In lesson 4, you will create a new [Windows Azure storage account](#) using Windows Azure Management Portal. An application may create many tables within a storage account. A table contains a set of entities (rows). Each entity contains a set of properties. An entity can have at most 255 properties including the mandatory system properties - PartitionKey, RowKey, and Timestamp. "PartitionKey" and "RowKey" form the unique key for the entity.

In this Tutorial

- [Lesson 1: Create a Windows Azure Visual Studio Project](#)
- [Lesson 2: Create a Data Source for Accessing Windows Azure Table Service](#)
- [Lesson 3: Create a Web Role to Display the Message Board and Process User Inputs](#)
- [Lesson 4: Test the Application and Deploy the Application](#)

This page intentionally left blank

Lesson 1: Create a Windows Azure Visual Studio Project

In this lesson, you create a Windows Azure Project with Visual Studio. A Windows Azure application can contain one or more roles. The application you will develop in this tutorial only contains one web role talking to Windows Azure Table service. In [tutorial 4](#), you will add a Worker role to the application for background processing.

Procedures

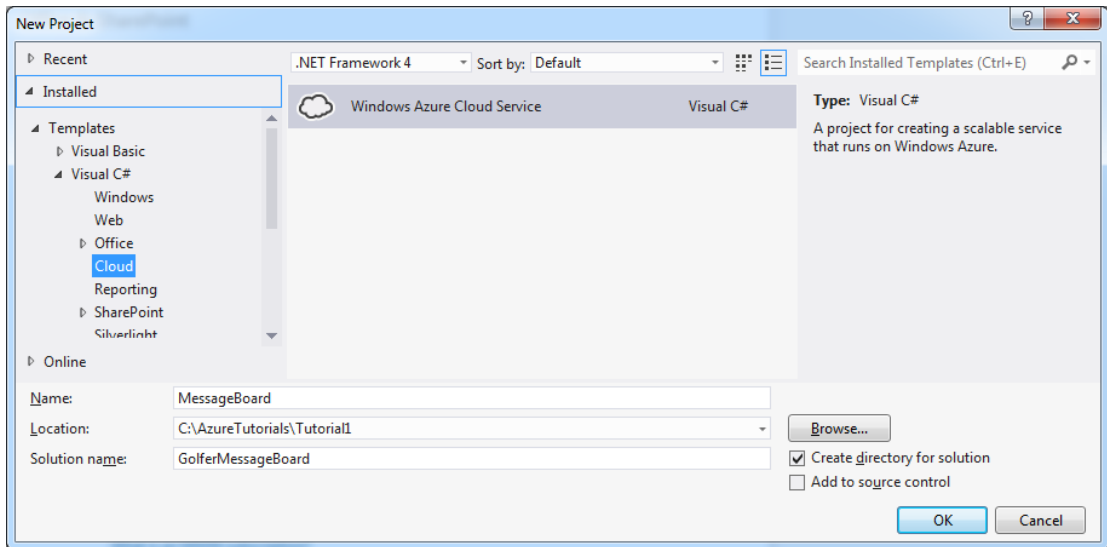
In this lesson, you will go through the following procedure:

- [Create a Visual Studio project](#)

To create a Visual Studio project

1. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2012**, right-click **Visual Studio 2012**, and then click **Run as administrator**.
2. If the User Account Control dialog appears, click **Yes**.
3. Click the **File** menu, point to **New**, and then click **Project**.
4. In the **New Project** dialog, type or select the following values:

Name	Value
Installed Templates	(In the left pane, expand Installed , expand Templates , expand Visual C# , and then click Cloud .)
Template	Windows Azure Cloud Service
Name	MessageBoard
Location	C:\AzureTutorials\Tutorial1
Solution name	GolferMessageBoard
Create directory for solution	(selected)



5. Click **OK**.
6. In the **New Windows Azure Project** dialog, inside the left pane, double-click **ASP.NET Web Role** to add it to the **Windows Azure Cloud Service Solution** list. The default name is **WebRole1**.
7. Right-click **WebRole1**, rename the role to **MessageBoard_WebRole**, and then click **OK**. The generated solution contains two projects. *MessageBoard* is a cloud service project, *MessageBoard_WebRole* is an ASP.NET web role project.
8. From Solution Explorer, expand **MessageBoard**, and then expand **Roles**. There is one role listed there, called **MessageBoard_WebRole**. This application only contains one web role.

Under the *MessageBoard* cloud service project, there are also one definition file and two configuration files. The *ServiceDefinition.csdef* file defines the runtime settings for the application including what roles are required, endpoints, and so on. The *ServiceConfiguration.Local.cscfg* contains the storage account connection string to use the local Windows Azure Storage emulator; the *ServiceConfiguration.Cloud.cscfg* file contains the settings to use Windows Azure storage service. For more information, see *Configuring the Windows Azure Application with Visual Studio* at <http://msdn.microsoft.com/en-us/library/ee405486.aspx>.

What did I just do?

In this step, you created a Windows Azure project with Visual Studio.

Next Steps

You add a new class library project to the solution for accessing Windows Azure Table service.

Lesson 2: Create a Data Source

The application stores message entries using Windows Azure Table service. The Table service is NOT relational database tables. It is helpful to think of it as object storage.

In this lesson, you first define the entity for golfer message entries. An entity contains a set of properties, for example, golfer name, and message. Then you create the data source so that the ASP.NET web role that you will create in the next lesson can use the data source to access the Table service. The data source has two methods, one for adding messages to the storage, and the other for listing the messages in the storage.

To access the Table service, you can use LINQ.

Procedures

In this lesson, you will go through the following procedures:

1. [Add a new project to the solution](#)
2. [Define the entity](#)
3. [Create a data source](#)
4. [Compile the project](#)

To add a new project

1. From Solution Explorer, right-click **Solution 'GolferMessageBoard'**, point to **Add**, and then click **New Project**.
2. In the **Add New Project** dialog, type or select the following values

Name	Value
Installed Templates	Visual C# , Windows
Template	Class Library
Name	MessageBoard_Data
Location	C:\AzureTutorials\Tutorial1\GolferMessageBoard

3. Click **OK**.
4. In Solution Explorer, delete **Class1.cs** inside the **MessageBoard_Data** project. You will add a new class file, instead of using the default one.
5. In Solution Explorer, right-click the **MessageBoard_Data** project, and then click **Add Reference**.
6. In the **Reference Manager** dialog, expand **Assemblies**, click **Framework**, and then select **System.Data.Services.Client**; Expand **Assemblies**, click **Extensions**, and then select **Microsoft.WindowsAzure.Configuration**, **Microsoft.WindowsAzure.StorageClient**

and **Microsoft.WindowsAzure.ServiceRuntime**. And then click **OK**. `System.Data.Services.Client` and `Microsoft.Windows.StorageClient` are used for utilizing the Windows Azure storage services. `Microsoft.WindowsAzure.ServiceRuntime` is used for retrieving data connection string in the configuration file.

Note: If multiple extension assemblies are presented, select the ones with version 1.7.0.0.

Note: If you cannot find a component, use the search box on the upper right corner of the dialog.

Next, define the entity for the message entries. 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, this message board application uses a fixed schema to store its data.

To define the entity

1. In Solution Explorer, right-click **MessageBoard_Data**, point to **Add**, and then click **New Item**.
2. In the **Add New Item – MessageBoard_Data** dialog, type and select the following values:

Name	Value
Installed Templates	Visual C# , Code
Template	Class
Name	MessageBoardEntry.cs

3. Click **Add**.
4. At the top of the `MessageBoardEntry.cs` file, insert the following namespace declaration.

```
using Microsoft.WindowsAzure.StorageClient;
```

5. Update the declaration of the `MessageBoardEntry` class to make it public and derive from the `TableServiceEntity` class.

```
public class MessageBoardEntry : TableServiceEntity  
{  
}
```

6. Add a default constructor and properties to the class by inserting the code snippet **Tutorial01-Lesson02-Task02_MessageBoardEntry**.

```
public MessageBoardEntry()  
{  
    PartitionKey = DateTime.UtcNow.ToString("MMddyyyy");  
    // Row key allows sorting, so we make sure the rows come back in time  
    order.  
    RowKey = string.Format("{0:10}_{1}",
```

```

        DateTime.MaxValue.Ticks - DateTime.Now.Ticks,
        Guid.NewGuid());
    }
    public string GolferName { get; set; }
    public string GolferMessage { get; set; }

```

Note: You can either copy/paste the code or use code snippet. The code snippets have more comments which can help you to review the code in the future. For more information on using the code snippets, see [Before You Begin the Tutorials](#).

In addition to the properties required by the data model, every entity 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.

The MessageBoardEntry.cs looks like the following (collapse to definitions) when completed:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure.StorageClient;

namespace MessageBoard_Data
{
    public class MessageBoardEntry:TableServiceEntity
    {
        //In addition to the properties required by the data model, every entity in table ...
        public MessageBoardEntry()...

        public string GolferName { get; set; }
        public string GolferMessage { get; set; }
    }
}

```

7. Press **Ctrl+S** to save the MessageBoardEntry.cs file.

Finally, you create a data source that can be bound to data controls in the ASP.NET web role that you will create in the next lesson.

To create a data source

1. In Solution Explorer, right-click **MessageBoard_Data**, point to **Add**, and then click **New Item**.
2. In the **Add New Item – MessageBoard_Data** dialog, type and select the following values, and then click **Add**.

Name	Value
Installed Templates	Visual C# , Code
Template	Class

Name	MessageBoardDataSource.cs
------	---------------------------

- At the top of the MessageBoardDataSource.cs file, insert the following namespace declarations.

```
using Microsoft.WindowsAzure;  
using Microsoft.WindowsAzure.StorageClient;  
using Microsoft.WindowsAzure.ServiceRuntime;
```

- Make the MessageBoardDataSource class public.

```
public class MessageBoardDataSource  
{  
}
```

- Define member fields by inserting the code snippet **Tutorial01-Lesson02-Task04_MessageBoardDataSourceFields**.

```
private const string messageTableName = "MessageTable";  
private const string connectionStringName = "DataConnectionString";  
private static CloudStorageAccount storageAccount;  
private CloudTableClient tableClient;
```

- Add a default constructor to the class by inserting the code snippet **Tutorial01-Lesson02-Task04_MessageBoardDataSourceConstructors**.

```
public MessageBoardDataSource()  
{  
    //add reference Microsoft.WindowsAzure.Configuration  
    storageAccount = CloudStorageAccount.Parse(  
        ConfigurationManager.GetSetting(connectionStringName));  
  
    // Create the table client  
    tableClient = storageAccount.CreateCloudTableClient();  
  
    tableClient.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(1));  
    tableClient.CreateTableIfNotExist(messageTableName);  
}
```

The constructor initializes the storage account by reading its settings from the configuration files and then uses the `CreateTablesIfNotExist` method in the `CloudTableClient` class to create the table used by the application. You will configure `DataConnectionString` in [Lesson 3](#).

- Inside the MessageBoardDataSource class, add some methods by inserting the code snippet **Tutorial01-Lesson02-Task04_MessageBoardDataSourceMethods**.

```
public IEnumerable<MessageBoardEntry> GetEntries()  
{  
    TableServiceContext tableServiceContext =  
    tableClient.GetDataServiceContext();  
    var results = from g in  
    tableServiceContext.CreateQuery<MessageBoardEntry>(messageTableName)  
    where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
```

```

        select g;
        return results;
    }

    public void AddEntry(MessageBoardEntry newItem)
    {
        TableServiceContext tableServiceContext =
        tableClient.GetDataServiceContext();
        tableServiceContext.AddObject(messageTableName, newItem);
        tableServiceContext.SaveChanges();
    }
}

```

The GetMessageBoardEntries method retrieves today's message board 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 message board.

The AddMessageBoardEntry method inserts new entries into the table.

The MessageBoardDataSource.cs looks like the following (collapse to definitions) when completed:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure.ServiceRuntime;

namespace MessageBoard_Data
{
    public class MessageBoardDataSource
    {
        internal const string messageTableName = "MessageTable";
        internal const string connectionStringName = "DataConnectionString";
        private static CloudStorageAccount storageAccount;
        private CloudTableClient tableClient;

        //The default constructor initializes the storage account by reading its settings from ...
        public MessageBoardDataSource()...

        //The GetMessageBoardEntries method retrieves today's message board entries by constructing ...
        public IEnumerable<MessageBoardEntry> GetEntries()...

        //The AddMessageBoardEntry method inserts new entries into the table.
        public void AddEntry(MessageBoardEntry newItem)...)
    }
}

```

8. Press **Ctrl+S** to save the MessageBoardDataSource.cs file.

To compile the project

- In Solution Explorer, right-click **MessageBoard_Data**, and then click **Build**.

What did I just do?

In this step, you created a data source that will be consumed by the ASP.Net web role that you will create in the next Lesson.

Next Steps:

You will create an ASP.NET web role for displaying the message board and process user input.

Lesson 3: Create a Web Role to Display the Message Board and Process User Input

In this lesson, you modify the web role project that you generated in Lesson 1 when you created the Windows Azure Cloud Service solution. This involves updating the UI to render the list of message board entries.

Procedures

In this lesson, you will go through the following procedures:

1. [Configure the web role project](#)
2. [Modify Default.aspx](#)
3. [Modify Default.aspx.cs](#)
4. [Add the storage account settings](#)

You must reference the `MessageBoard_Data` object from the web role project. To save time, you will begin with a skeleton `default.aspx` file.

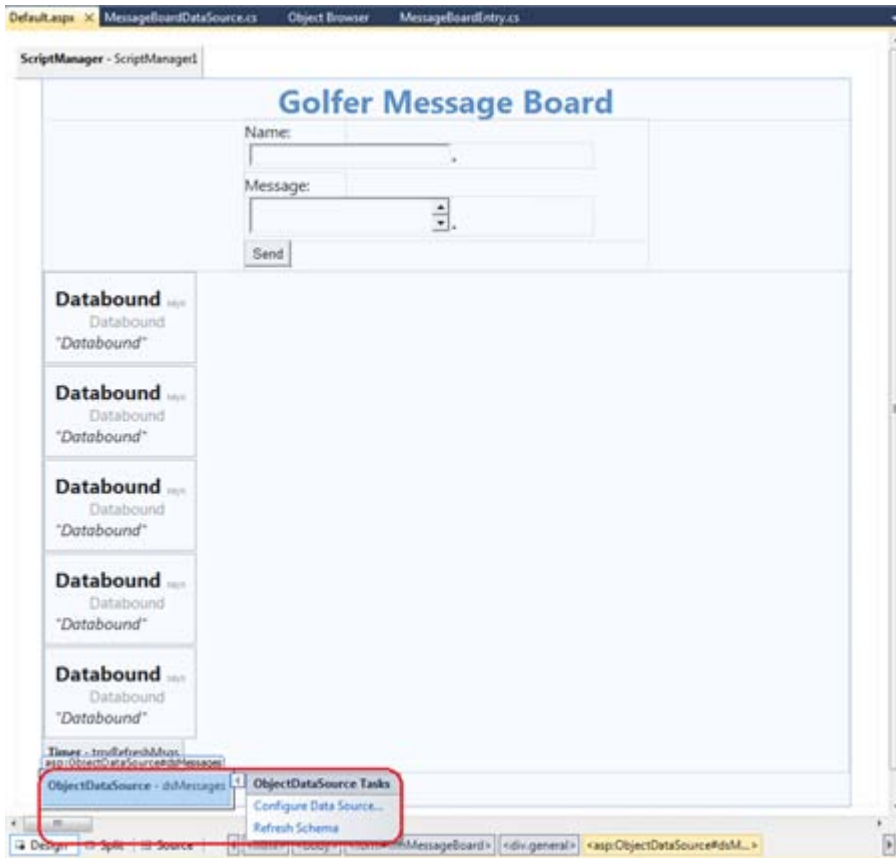
To configure the web role project

1. In Solution Explorer, right-click the **MessageBoard_WebRole** project, and then click **Add Reference**.
2. In the **Reference Manager** dialog, expand **Solution**, and then click **Projects**.
3. Select **MessageBoard_Data**, and then click **OK**.
4. In Solution Explorer, right-click **Default.aspx** of the **MessageBoard_WebRole** project, and then click **Delete**.
5. In Solution Explorer, right-click **MessageBoard_WebRole**, point to **Add**, and then select **Existing Item**.
6. In **Add Existing Item – MessageBoard_WebRole**, browse to the **C:\AzureTutorials\Tutorial1\TutorialFiles** folder, hold the **CTRL** key down while selecting all four files in the folder, and then click **Add**.
7. In Solution Explorer, right-click **Solution 'GolferMessageBoard'**, and then click **Rebuild Solution**.

The `default.aspx` file uses a `DataList` control and an `ObjectDataSource` control to display the messages. In the following procedure, you configure the two controls.

To modify Default.aspx

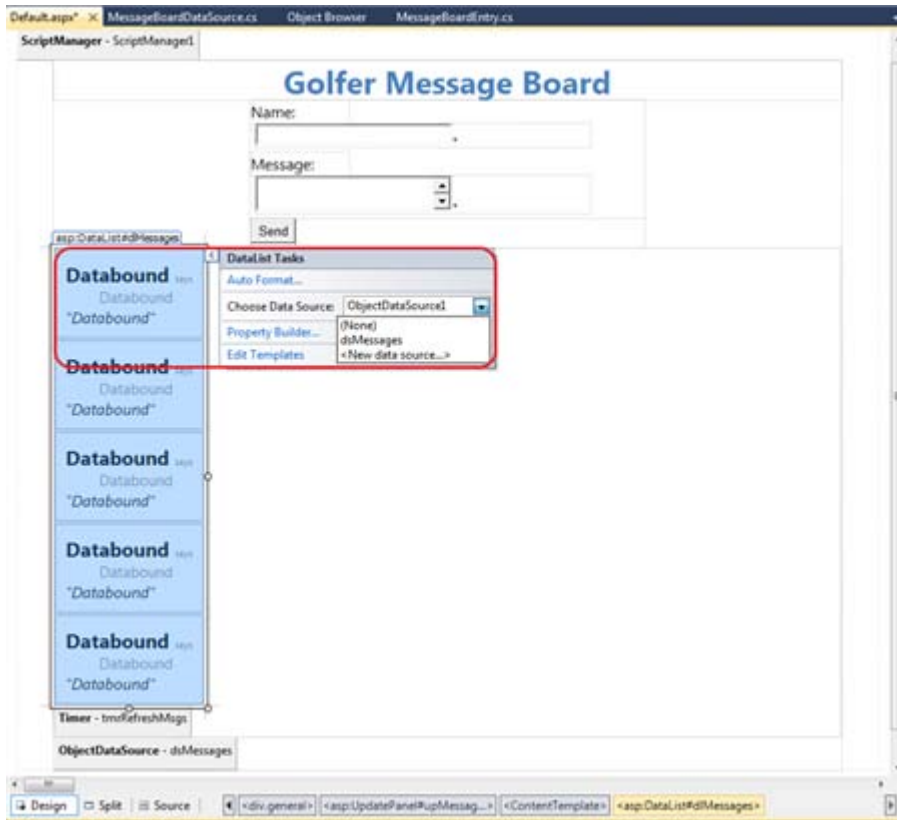
1. In Solution Explorer, right-click **Default.aspx** of the **MessageBoard_WebRole** project, and then click **View Designer**.
2. Hover over the **dsMessages** `ObjectDataSource` control on the bottom of the page, click the right arrow button, and then click **Configure Data Source**.



3. In **Choose your business object**, select **MessageBoard_Data.MessageBoardDataSource**, and then click **Next**.

Note: If you get a message saying "The Type 'MessageBoard_Data.MessageBoardDataSource' could not be found", rebuild the solution and try again.

4. Click the **SELECT** tab.
5. In **Choose a method**, select **GetEntries()**, returns **IEnumerable<MessageBoardEntry>**, and then click **Finish**. The GetEntries() is what you defined in the object in the previous lesson.
6. Hover over the @Messages DataList control, and then click the right arrow button. In **Choose Data Source**, select **dsMessages**.



7. Click **No** to reject reset template.
8. Press **Ctrl+S** to save the Default.aspx file.

Next, you implement the code necessary to store submitted entries to the Table service.

To modify Default.aspx.cs

1. In Solution Explorer, right-click **Default.aspx** of the **MessageBoard_WebRole** project, and then click **View Code**.
2. At the top of Default.aspx.cs, add the following namespace declaration.

```
using MessageBoard_Data;
```

3. Locate the **btnSend_Click** method, and insert the code snippet **Tutorial01-Lesson03-Task02_WebRoleBtnSend_Click** inside the method.

```
// create a new entry in the table
MessageBoardEntry entry = new MessageBoardEntry() { GolferName = txtName.Text,
GolferMessage = txtMessage.Text };
MessageBoardDataSource ds = new MessageBoardDataSource();
ds.AddEntry(entry);

txtName.Text = "";
txtMessage.Text = "";

dlMessages.DataBind();
```

This method creates a new `MessageBoardEntry` entity, initializes it with the information submitted by the user, and then uses the `MessageBoardDataSource` class to save the entry to the table. Then, it refreshes its contents using the `DataBind()` method.

4. Locate the `tmrRefreshMsgs_Tick` method, and insert the following line of code.

```
dlMessages.DataBind();
```

This method refreshes the page every 15 seconds. The refreshing interval is pre-configured in the `default.aspx` file.

5. Locate the `Page_Load` method, and insert the code snippet **Tutorial01-Lesson03-Task02_WebRolePage_Load**.

```
if (!Page.IsPostBack)
{
    tmrRefreshMsgs.Enabled = true;
}
```

The code enables the page refresh timer.

6. Press **Ctrl+S** to save the `Default.aspx.cs` file.

Earlier in the tutorial, you defined a constant called `connectionStringName`, the value is `DataConnectionString`. `DataConnectionString` is a setting in the configuration file. You must define it. When you add a role to the Windows Azure project, Visual Studio generated two configuration files, `ServiceConfiguration.Cloud.cscfg` and `ServiceConfiguration.Local.cscfg`. You can configure the local configuration file to use the Storage Emulator for testing the application locally, and the cloud configuration file to use a Windows Azure storage account. Creating a storage account will be covered in lesson 4. For the time being, you configure both configuration files to use Storage Emulator.

Note: Storage Emulator offers local storage services that simulate the Blob, the Queue, and the Table services available in Windows Azure. The Storage Emulator UI provides a means to view the status of local storage service and to start, stop, and reset them. For more information, see *Overview of the Windows Azure Storage Emulator* at <http://msdn.microsoft.com/en-us/library/gg432983.aspx>.

To add the storage account settings

1. In **Solution Explorer**, expand **MessageBoard**, expand **Roles**, and then double-click **MessageBoard_WebRole** to open the properties for this role.
2. In the left, click **Settings**.
3. In the **Service Configuration** drop down in the heading, select **All Configurations**.
4. Click **Add Setting**.
5. Type or select the following values.

Name	Value
------	-------

Name	DataConnectionString
Type	Connection String
Value	(click the ellipses button (...), and then select Use the Windows Azure storage emulator). "UseDevelopmentStorage=true" will appear in the Value cell.

6. The **DataConnectionString** is used in the MessageBoardDataSource class you defined in [Lesson 2](#).
7. Press **Ctrl+S** to save changes to the role configuration.

What did I just do?

In this step, you created a web role to display the message board and process user input.

Next Steps:

You will test the application in the compute emulator. Then you will generate a service package that you can use to deploy the application to Windows Azure.

Lesson 4: Test the Application and Deploy the Application

In this lesson, you test the message board application in the local compute emulator, and then deploy the application to Windows Azure as a cloud service.

Note: The Windows Azure compute emulator simulates the Windows Azure fabric on your local computer so that you can run and test your service locally before deploying it. For more information, see *Overview of the Windows Azure SDK Tools* at <http://msdn.microsoft.com/en-us/library/gg432968.aspx>.

Procedures

In this lesson, you will go through the following procedures:

1. [Test the application](#)
2. [Generate the service package](#)
3. [Sign in to Windows Azure](#)
4. [Create a storage account](#). You need a storage account to utilize Windows Azure storage services, for example the table service.
5. [Create a cloud service](#). The cloud service is used to host the message board application.
6. [To configure the ServiceConfiguration.Cloud.cscfg file](#)
7. [Deploy the application to the staging environment](#)
8. [Test the application](#)
9. [Promote the application to the production environment](#)

To test the application

1. In Solution Explorer, right-click **MessageBoard**, and then click **Set as Startup Project**.
2. From the **Debug** menu, click **Start Debugging**. You will see a compute emulator icon added to the notification area of taskbar, and a new Internet Explorer window showing the Golfer Message Board application.
3. (Optional) Right-click the compute emulator icon (a blue flag icon) on the Taskbar to show the compute emulator UI and the storage emulator UI.
4. Switch to Internet Explorer to view the message board application.
5. Add a few entries to the message board by entering your name and a message before clicking **Send**.
6. Close the Internet Explorer window.

After the application is tested successfully in the compute emulator environment, the next step is to create the service package and then deploy the application to Windows Azure.

To generate the service package

1. In Solution Explorer, right-click the **MessageBoard** cloud project, and then select **Package**.
2. In the **Package Windows Azure Application** dialog, select the following values:

Name	Value
Service configuration	Cloud
Build configuration	Release

3. Click **Package**. 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. The default directory is `C:\AzureTutorials\Tutorial1\GolferMessageBoard\MessageBoard\bin\Release\app.publish`. Make sure to write down the path. You will need it in the deployment lesson.

You will get a few warning messages about "DataConnectionString" set up to use the local storage emulator. You can ignore these warning for now.

For deploying the golfer message board application, you must have a *storage account* for accessing the Windows Azure storage services, and a *cloud service*, which is a container for service deployments in Windows Azure. For better performance, you might want to create an *affinity group* to group the service and the storage accounts within a subscription according to geo-location.

To sign in to Windows Azure

1. Open a Web browser and browse to <http://windows.azure.com/>. This is the Windows Azure Management Portal.
2. Sign in using the Windows Live ID associated with your Windows Azure account.

Note: If you haven't had a Windows Azure Platform subscription, see the [Provisioning Windows Azure](#) section of this tutorial.

To create a storage account for the golfer message board application to store its data

1. From the portal, in the left pane, click **STORAGE**.
2. On the bottom left corner, click **NEW**, click **STORAGE**, and then click **QUICK CREATE**.
3. Type or select the following values.

Name	Value
URL	<yourname>gmb (i.e. johndolegmb)
REGION/AFFINITY GROUP	(Choose the region where you wish your service to run, most likely, the one that is closest to your current location. Select an affinity group instead of a region if you want your storage services to be in the same data center with the cloud service that you will create in the next step. To create an affinity group, open the NETWORKS area of the Management Portal, click AFFINITY

	GROUPS, and then click CREATE.)
--	---------------------------------

4. Click the check mark on the bottom right corner. Wait until the status of the storage account changes to **Online**. The process can take several minutes.
5. Once the storage account is created, click the storage account from the storage account list to select it, and then click **MANAGE KEYS** on the bottom of the page.
6. Record the **STORAGE ACCOUNT NAME** and **PRIMARY ACCESS KEY**. Later in the tutorial, you will need to configure your cloud service to use storage account by specifying the account name and the access key.
7. Close the dialog.

To create a cloud service

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. On the bottom left corner of the portal page, click **NEW**, click **COMPUTE**, click **CLOUD SERVICE**, and then click **CUSTOM CREATE**.
3. Type or select the following values.

Name	Value
URL	<yourname>gmb Note: The URL prefix must be unique.
REGION/AFFINITY GROUP	(For better performance, select the same region as the one you chose for the storage service, or use an affinity group.)
Deployment a cloud service package now	(not selected)

4. Click the check sign on the bottom right corner. Wait until the status changes to **Created**. This process could take several minutes.

When you create and test the application locally, the application is configured to use the development storage. Now you have created a storage account, you can configure the application to use the storage account before deploying the application to Windows Azure. The configuration information is in the ServiceConfiguration.Cloud.cscfg file. This file was created when you generated the service package.

To configure the ServiceConfiguration.Cloud.cscfg file

1. Use Notepad to open **C:\AzureTutorials\Tutorial1\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg**. This path can be different if you installed the tutorial files into

a different folder than the c root directory. Notice the value of `DataConnectionString` is **"UseDevelopmentStorage=true"**.

2. Change the value of `DataConnectionString` to **DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your storage account primary access key>**. Replace <your storage account name> and <your storage account access key> accordingly.
3. Change the value of `Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString` to **DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your storage account primary access key>**. You must replace <your storage account name> and <your storage account primary access key> with the actual account name and account key for the storage account that you created earlier in the tutorial.
4. Optionally, you can increase the Instances count to scale out the application.
5. Save the file and close Notepad.

To deploy the application to the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you just created. The default name is <yourname>gmb.
3. From the top of the page, click **STAGING**.
4. Click **UPLOAD A NEW STAGING DEPLOYMENT**.
5. From **Upload a package**, type or select the following value.

Name	Value
Deployment name	v1.0.0.0
Package location	C:\AzureTutorials\Tutorial1\GolferMessageBoard\MessageBoard\bin\Release\app.publish\MessageBoard.cspkg
Configuration file	C:\AzureTutorials\Tutorial1\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg
Deploy even if one or more roles contain a single instance	(Selected). You can always increase the number of instances from the portal.
Start deployment	(Selected)

6. Click the check sign on the bottom right corner of the page.
7. Wait until the upload process is completed. The process can take several minutes to complete.

To test the application in the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the right side of the page, click the site URL.
5. Test the application by entering one or more entries.

After the application is working correctly in the staging environment, you are ready to promote it to the production environment.

To promote the application to production

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the bottom of the page, click **SWAP**.
5. On the right, click **YES**.
6. On the top of the page, click **PRODUCTION**. It takes several minutes to complete the operation.
7. On the right, click the SITE URL.
8. In the **Properties** pane, click the URL in the **DNS name** box. The application is opened in a new browser tab or a new browser window depending on your browser configuration.

Note: Some DNS services take longer to replicate the records. If you get a page not found error, you might need to try browsing to the URL again in a few minutes.

9. Test the application in the production environment by entering one or more entries.

What did I just do?

In this step, you deployed the golfer message board to Windows Azure.

Next Steps:

Congratulations! You have completed tutorial 1. [Tutorial 2](#), [tutorial 3](#) and [tutorial 4](#) show you how to use SQL Database and other Windows Azure storage services.

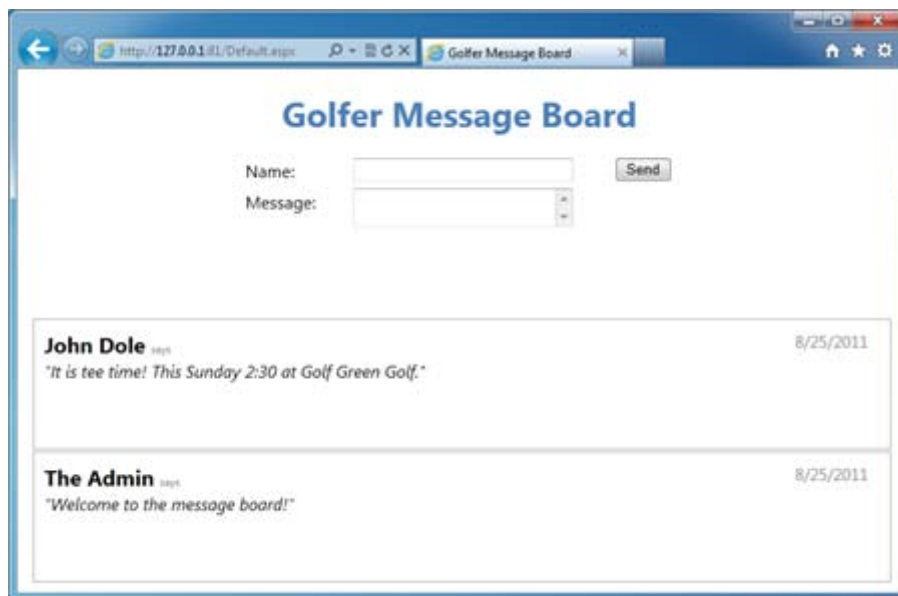
Tutorial 2: Using SQL Database

This tutorial demonstrates the process of creating a Windows Azure cloud service consuming Windows Azure SQL Database and the process of deploying the application to Windows Azure.

Overview

In [tutorial 1](#), you created a simple golfer message board application. In the application, a ASP.NET Web role provides the front-end that allows golfers to view the contents of the message board and post new entries. Each entry contains a name and a message. When golfers post a new message, the Web role creates an entry that contains the information entered by the golfers using Windows Azure Table service. In this tutorial, you will modify the application so that it uses SQL Database instead of the Table service.

Here is a screenshot of the application:



Note: This tutorial is not intended to show you how to choose between SQL Database and the Table service, but to show you how to use SQL Database. For information on SQL Database and Windows Azure Table service comparison, see [Windows Azure SQL Database and Windows Azure Table Service](#) at <http://msdn.microsoft.com/en-us/magazine/gg309178.aspx>, and [Data Storage Offerings on the Windows Azure Platform](#).

Note: Completing [Tutorial 1](#) is not a pre-requisite for this tutorial; however, it helps with understanding the scenario.

Note: For the tutorial in PHP, see [Using the Windows Azure Web Role and SQL Database with PHP](#).

Objectives

In this tutorial, you will learn how to:

- Understanding the process of developing a SQL Database cloud service application
- Use SQL Database as a cloud-hosted database platform for your applications

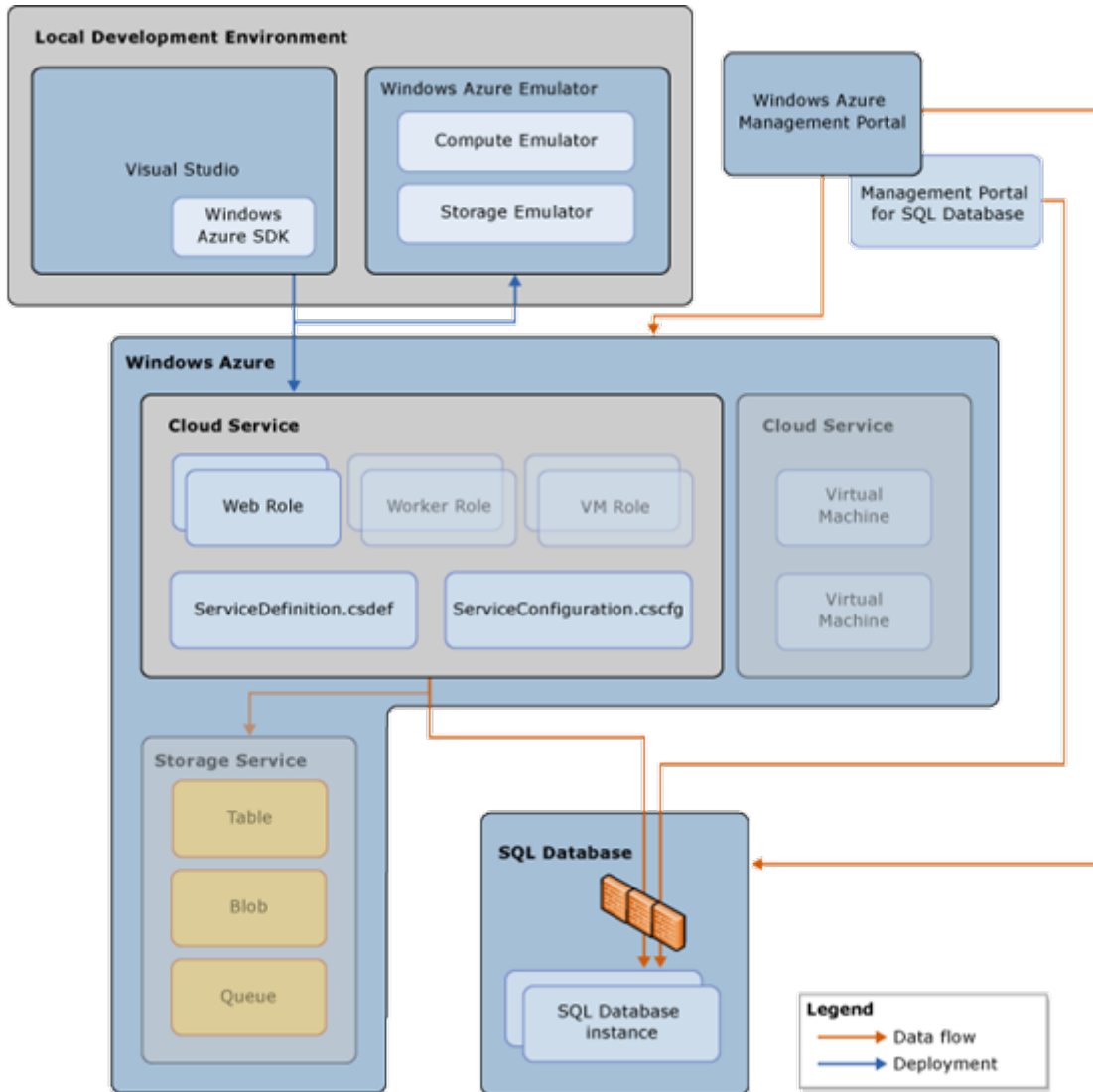
Prerequisites

Note the following requirements before you begin this lesson:

- .NET development experience
- Before you begin this step you must complete the steps in [Before You Begin the Tutorials](#).

Understanding the Architecture

The following diagram illustrates the development components and the runtime components involved in this tutorial:



- There are two environments: the Windows Azure runtime environment and the local development environment. The runtime environment is hosted in one of the Microsoft data centers.
- There are two main services within the runtime environment: Compute and Storage. Compute is the computational part of Windows Azure. Storage consists of three main parts: Table service, Blob service and Queue service. Other than the Windows Azure Storage Service, you can also use SQL Database to store data. SQL Database is a cloud-based relational database service built on Microsoft SQL Server technologies. In this tutorial, you will use SQL Database. Tutorial 1 covers Table service; tutorial 3 covers Blob service; and tutorial 4 covers Queue service. For more information, see [Data Storage Offerings on the Windows Azure Platform](#).
- The Compute service is also referred as Cloud Service. It is the application you deploy on Windows Azure. Every Cloud Service can have Web role and Worker role. 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. A worker role is a role that is useful for generalized development, and may perform background processing for a Web role. In tutorial 1, you created a Web role

project. Tutorial 2 and tutorial 3 uses the same Web role. Tutorial 4 will introduce Worker role. For more information, see [Overview of a Windows Azure Application](#).

- You can use the portal to administrate Windows Azure platform resources, and use Database Manager to perform basic database management tasks.
- You develop the SQL Database application using Visual Studio and Windows Azure SDK.
- You deploy the application to Windows Azure Emulator for testing and to Windows Azure.
- A Windows Azure project includes two configuration files: ServiceDefinition.csdef and ServiceConfiguration.cscfg. These files are packaged with your Windows Azure application and deployed to Windows Azure.

In this Article

- [Lesson 1: Prepare SQL Database account](#)
- [Lesson 2: Modify the application to use SQL Database](#)
- [Lesson 3: Test and deploy the application](#)

Lesson 1: Prepare SQL Database Account

In this lesson, you connect to your SQL Database account and create a server and a database for the Golfer Message Board application using Windows Azure Management Portal. You can also use SQL Database Management API to create SQL Database servers, databases, and tables programmatically. For more information, see [About the Windows Azure SQL Database Management API](#).

Procedures

In this lesson, you will go through the following procedures:

- [Create a SQL Database server](#)
- [Create a database](#)
- [Create a table](#)

You must have a Windows Azure Platform subscription.

To create a SQL Database server and a database

1. Open a Web browser, and browse to **http://windows.azure.com**.
2. Log in to your Windows Live account.
3. On the bottom of the page, click + **NEW**, click **DATA SERVICES**, click **SQL DATABASE**, and then click **CUSTOM CREATE**.
4. Type or select the following values:

Name	Value
NAME	GolferMessageBoardDB
SERVER	New SQL Database Server

5. Click the right arrow on the bottom right corner of the dialog.
6. Type the following values:

Name	Value
LOGIN NAME	MyAdmin
LOGIN PASSWORD	pass@word1
LOGIN PASSWORD CONFIRMATION	pass@word1
REGION	(Select a region that you want your service to reside. For

	better performance, choose the one that is close to you.)
ALLOW WINDOWS AZURE SERVICES TO ACCESS THE SERVER	(selected)

Note: An administrator account is a master account used to manage the new server. You should avoid using this account in connection strings where the username and password may be exposed. To simplify the tutorial instructions, this tutorial uses the administrator account. The default administrator username is *MyAdmin*, and the default password is *ypass@word1*. If you change the username and the password in this step, you must change them accordingly in the rest of the tutorial.

Note: The password policy requires that this password contain at least one number, one character, one letter, and one symbol. In addition, the password cannot be less than six characters nor contain three consecutive characters from the username.

7. Click the check mark on the bottom right corner of the page.
8. On the top of the page, click **DATABASES** to list the SQL databases. From the list, you can find out the SQL Database server name for the GolferMessageBoardDB database. Wait until the **STATUS** column shows **Online**.

You can use either SQL Server Management Studio or Windows Azure Management Portal to manage your SQL Database. To connect to SQL Database from SQL Server Management Studio, you must provide the fully qualified domain name of the server. In this tutorial, you will use Windows Azure Management Portal.

Note: The SQL Server Management Studio from SQL Server 2008 R2 and SQL Server 2008 R2 Express can be used to access, configure, manage and administer SQL Database. Previous versions of SQL Server Management Studio are not supported.

SQL Database has two types of access control: firewall and SQL authentication. You must configure the SQL Database firewall settings to allow connections from your computer(s). You must also allow connections from Windows Azure, because the golfer message board application is hosted in Windows Azure (This is done in the previous procedure). In addition to configuring the SQL Database server-side firewall, you must also configure your client-side environment to allow outbound TCP connections over TCP port 1433. For more information on SQL Database security, see [Security Guidelines for Windows Azure SQL Database](#).

To create a SQL Database server firewall rule

1. From Windows Azure Management Portal, in the left pane, click **SQL DATABASES**. You shall see a list of SQL databases.
2. Find the GolferMessageBoardDB database, and then click the server (on the same row) which hosts the database.
3. On the top of the page, click **CONFIGURE**.

4. Click **ADD TO ALLOWED IP ADDRESSES**. A new firewall rule which allows the development computer to access is added to the list. This step is required before using Management Portal for SQL Database.
5. At the bottom of the page, click **SAVE**.

The Golfer Message Board application has one table for storing the messages.

To create a table

1. From Windows Azure Management Portal, click **SQL DATABASES**.
2. From the SQL database list, click **GolferMessageBoardDB** to select the row.
3. On the bottom of the page, click **MANAGE**. The Management Portal for SQL Database is opened in a different browser tab or new browser.
4. Type the following values:

Name	Value
SERVER	(use default value)
DATABASE	GolferMessageBoardDB
USERNAME	MyAdmin
PASSWORD	pass@word1

5. Click **Log on**.
6. Click **New Query**.
7. In the **Query** window, type the following query, and then click **Run**.

```
CREATE TABLE [Messages] (  
    [MessageID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,  
    [GolferName] [nvarchar](100) NOT NULL,  
    [GolferMessage] [nvarchar](1000) NOT NULL,  
    [Timestamp] [datetime] NOT NULL  
)
```

8. Click the **Message** button on the bottom of the main pane, and make sure the query is completed successfully.

What did I just do?

In this step, you created a SQL Database server, a database, and a table needed by the golfer message board application.

Next Steps:

You will modify the golfer message board application you created in tutorial 1 so that it uses SQL Database instead of the Table service for storing the messages.

Lesson 2: Modify the Application to use SQL Database

Currently, the application only needs to access a single table. It is easy and straightforward to access the data using LINQ.

There are two places you need to modify the application to use SQL Database. One is listing the messages, and the other is inserting new messages to the database.

Procedures

In this lesson, you will go through the following procedures:

1. [Open the golfer message board application](#)
2. [Create messages entity class](#)
3. [Modify the default.aspx page for listing the messages](#)
4. [Modify the default.aspx.cs page for inserting new messages](#)
5. [Compile the MessageBoard_WebRole project](#)

To open the Golfer Message Board application

1. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2012**, right-click **Visual Studio 2012**, and then click **Run as administrator**.
2. If the **User Account Control** dialog appears, click **Yes**.
3. Click the **FILE** menu, point to **Open**, and then click **Project/Solution**.
4. In **File name**, type **C:\AzureTutorials\Tutorial2\GolferMessageBoard\GolferMessageBoard.sln**, and then click **Open**.

To create message entity class

1. In Solution Explorer, right-click **MessageBoard_WebRole**, point to **Add**, and then click **New Item**.
2. In the Add New Item – MessageBoard_WebRole dialog, type and select the following values:

Name	Value
Installed Templates	Visual C# , Data
Template	LINQ to SQL Classes
Name	MessageBoard.dbml

3. Click **Add**.
4. From the **View** menu, click **Server Explorer**. You can also press **Ctrl+Alt+S** to open Server Explorer.
5. In Server Explorer, right-click **Data Connections**, and then click **Add Connection**.
6. Type and select the following values.

Name	Value
Data source	Microsoft SQL Server
Data provider	.NET Framework Data Provider for SQL Server

- Click **Continue**.
- Type and select the following values.

Name	Value
Data source	Microsoft SQL Server (SqlClient)
Data provider	.NET Framework Data Provider for SQL Server
Server name	<ServerName>.database.windows.net
User SQL Server Authentication	(Selected)
User name	MyAdmin
Password	pass@word1
Select or enter a database name	GolferMessageBoardDB

Note: SQL Database doesn't support unencrypted connections. If you try to request a connection via SQL Server Management Studio that is unencrypted, SQL Database signals SQL Server Management Studio to establish an encrypted connection. To change the Encrypt setting, click **Advanced** from the **Add Connection** dialog. For more information, see [Overview of Security in Windows Azure SQL Database](#).

Note: The <ServerName> is the SQL Database server name you wrote down when you created the server.

Note: SQL Database supports only SQL Server Authentication. For more information, see *Security Guidelines and Limitations (Windows Azure SQL Database)* at <http://msdn.microsoft.com/en-us/library/ff394108.aspx#authentication>.

- Click **OK**.
- In Server Explorer, expand the database connection you just added, expand **Tables**.
- Drag-and-drop the **Messages** table onto the O/R Designer. The Object Relational Designer (O/R Designer) provides a visual design surface for creating and editing LINQ to SQL classes (entity classes) that are based on objects in a database.
- In Solution Explorer, right-click **MessageBoard_WebRole**, and then click **Rebuild**.

In [tutorial 1](#), you used a DataList control and an ObjectDataSource control to list the messages. You will replace the ObjectDataSource control with a LinqDataSource control in this tutorial.

To modify the Default.aspx page for listing the messages

1. In Solution Explorer, expand **MessageBoard_WebRole**, right-click **Default.aspx**, and then click **View Designer**.
2. From Toolbox, drag a **LinqDataSource** data control (under the Data category), and then drop it to the right or the bottom of **dsMessages**.



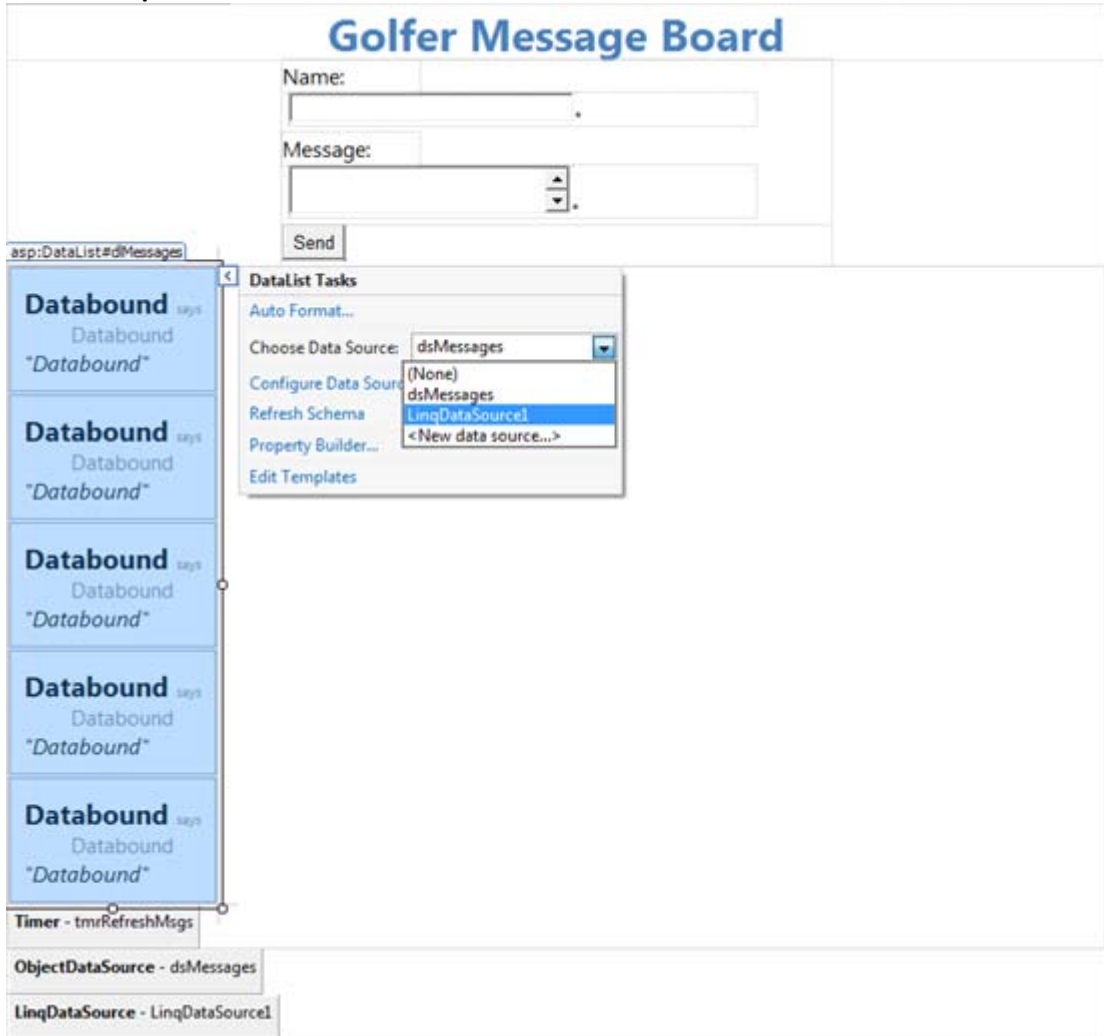
Note: Click **Ctrl+Alt+X** to display the Toolbox. The **LinqDataSource** control is listed under the **Data** category. The **dsMessages** control appears on the bottom of the page.

3. Hover over the **LinqDataSource1** control, click the right button, and then click **Configure Data Source**.
4. In **Choose your context object**, select **MessageBoard_WebRole.MessageBoardDataContext**, and then click **Next**.

Note: The context object will not be listed until you compile the **MessageBoard_WebRole** project.

5. In **Select**, select **GolferName**, **GolferMessage**, and **Timestamp**, and then click **Finish**.

6. Hover over the **dlMessages** DataList control, click the right button, then in **Choose Data Source**, choose **LinqDataSource1**.



7. Click **No** to reject resetting the template.

To modify the Default.aspx.cs page for inserting new messages

1. In Solution Explorer, right-click **Default.aspx**, and then click **View Code**.
2. Delete the code inside the **btnSend_Click** method, and then insert the code snippet **Tutorial02-Lesson02-Task04_WebRoleBtnSend_Click**:

```
protected void btnSend_Click(object sender, EventArgs e)
{
    MessageBoardDataContext context = new MessageBoardDataContext();
    var newMessage = new Message
    {
        GolferName = txtName.Text,
        GolferMessage = txtMessage.Text,
        Timestamp = DateTime.UtcNow
    };

    context.Messages.InsertOnSubmit(newMessage);
    context.SubmitChanges();
}
```

```
txtName.Text = "";  
txtMessage.Text = "";  
dlMessages.DataBind();
```

If you don't see the code snippet, follow the instructions found in [Before You Begin The Tutorials](#) to copy the snippet.

Because you don't need the Table service, you can delete the MessageBoard_Data project from the solution, all of the references, and the associated using statements. However it will not impact your testing if you choose to keep the project.

To compile the MessageBoard_WebRole project

- In Solution Explorer, right-click **MessageBoard_WebRole**, and then click **Rebuild**. Make sure the project is compiled successfully.

What did I just do?

In this step, you modified the golfer message board application to connect to SQL Database via LINQ.

Next Steps:

You will test and deploy the application.

Lesson 3: Test and Deploy the Application

In this lesson, you test the application in Windows Azure computer emulator, package the application, and then deploy the application to Windows Azure.

Note: If you don't have a Windows Azure Platform subscription, see the *Provisioning Windows Azure* section of this tutorial.

Procedures

In this lesson, you will go through the following procedures:

1. [Test the application in compute emulator](#)
2. [Generate the service package](#)
3. [Sign in to Windows Azure](#)
4. [Create a cloud service](#)
5. [Deploy the application to the staging environment](#)
6. [Test the application in the staging environment](#)
7. [Promote the application to the production environment](#)

To test the application in compute emulator

1. If MessageBoard is not the startup project: In Solution Explorer, right-click **MessageBoard**, and then click **Set as Startup Project**.
2. From the **Debug** menu, click **Start Debugging**. The Golfer Message Board page is opened in a new browser window.
3. Switch to the browser window to view the message board application.
4. Add a few entries to the message board by entering your name and a message before clicking **Send**.
5. Close the browser window to stop the debugger and shut down the development in the compute emulator.

After the application is tested successfully in the compute emulator environment, the next step is to create the service package and then deploy the application to Windows Azure.

To generate the service package

1. In Solution Explorer, right-click the **MessageBoard** cloud project, and then click **Package**.
2. In Package Windows Azure Application, select the following values:

Name	Value
Service configuration	Cloud
Build configuration	Release

3. Click **Package**. 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. The default directory is `C:\AzureTutorials\Tutorial2\GolferMessageBoard\MessageBoard\bin\Release\app.publish`. Make sure to write down the path. You will need it later in this lesson.

To sign in to Windows Azure

1. Open a Web browser and browse to <http://windows.azure.com/>. This is the Windows Azure Management Portal.
2. Sign in using the Windows Live ID associated with your Windows Azure account.

Note: If you haven't had a Windows Azure Platform subscription, see the [Provisioning Windows Azure](#) section of this tutorial.

You need to create a cloud service for the Web role. If you have created a cloud service account in [tutorial 1](#), you can skip this step.

To create a cloud service

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. On the bottom left corner of the portal page, click **NEW**, click **COMPUTE**, click **CLOUD SERVICE**, and then click **CUSTOM CREATE**.
3. Type or select the following values.

Name	Value
URL	<yourname>gmb Note: The URL prefix must be unique.
REGION/AFFINITY GROUP	(For better performance, select the same region as the one you chose for the storage service, or use an affinity group.)
Deployment a cloud service package now	(not selected)

4. Click the check sign on the bottom right corner. Wait until the status changes to **Created**. This process could take several minutes.

To deploy the application to the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you just created. The default name is <yourname>gmb..

3. From the top of the page, click **STAGING**.
4. Click **UPLOAD A NEW STAGING DEPLOYMENT**.
5. From **Upload a package**, type or select the following value.

Name	Value
Deployment name	v1.1.0.0
Package location	C:\AzureTutorials\Tutorial2\GolferMessageBoard\MessageBoard\bin\Release\app.publish\MessageBoard.cspkg
Configuration file	C:\AzureTutorials\Tutorial2\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg
Deploy even if one or more roles contain a single instance	(Selected). You can always increase the number of instances from the portal.
Start deployment	(Selected)

6. Click the check sign on the bottom right corner of the page.
7. Wait until the upload process is completed. The process can take several minutes to complete.

To test the application in the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the right side of the page, click the site URL.
5. Test the application by entering one or more entries.

After the application is working correctly in the staging environment, you are ready to promote it to the production environment.

To promote the application to production

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the bottom of the page, click **SWAP**.
5. On the right, click **YES**.
6. On the top of the page, click **PRODUCTION**. It takes several minutes to complete the operation.
7. On the right, click the SITE URL.

8. In the **Properties** pane, click the URL in the **DNS name** box. The application is opened in a new browser tab or a new browser window depending on your browser configuration.

Note: Some DNS services take longer to replicate the records. If you get a page not found error, you might need to try browsing to the URL again in a few minutes.

9. Test the application in the production environment by entering one or more entries.

What did I just do?

In this step, you tested and deployed the golfer message board to Windows Azure.

Next Steps:

Congratulations! You have completed the tutorial 2. [tutorial 3](#) shows you another Windows Azure storage called Blob. [tutorial 2.1](#), which is optional, shows you how to add a data service that implements the Open Data Protocol (OData) to the project.

Tutorial 2.1: Create an OData Service

This tutorial demonstrates the process of creating a data service application that implements the Open Data Protocol (OData) to expose data from Windows Azure SQL Database and the process of deploying the application to Windows Azure.

Overview

In [tutorial 1](#) you created a simple golfer message board application. In [tutorial 2](#) you updated the application to store message board data, which includes a name and a message, in a SQL Database.

This tutorial is based on these two previous tutorials. In this tutorial you extend the Web application to add a data service that implements the Open Data Protocol (OData). This OData service uses the SQL Database created in tutorial 2.

Objectives

In this tutorial, you will learn how to:

- Create an Entity Framework data model to access SQL Database-hosted database
- Use WCF Data Services with the Entity Framework provider to create an OData service
- Deploy a WCF Data Services Web application to Windows Azure

Prerequisites

Note the following requirements before you begin this lesson:

- Complete the steps in [Tutorial 2 Using SQL Database](#)

Understanding the Architecture

The data service runs in the Windows Azure Web role and uses Entity Framework to access data from the SQL Database service. For more general information on the Windows Azure and SQL Database architecture used in this application, see the section Understanding the Architecture in [tutorial 2](#).

In this Article

- [Lesson 1: Define the Entity Framework data model](#)
- [Lesson 2: Modify the application to add the data service](#)
- [Lesson 3: Test and deploy the application](#)

Lesson 1: Modify the Application to Define the Entity Framework Data Model

In [tutorial 2](#), the application accessed the GolferMessageBoardDB database by using a data source based on LINQ-to-SQL classes.

You need to use the Entity Data Model wizard to add an Entity Framework data model to the project. This data model will be used by the data service to access the database. When you create a new Entity Data Model in your project, the wizard also adds the appropriate assembly references to the project.

Procedures

In this lesson, you will go through the following procedures:

1. [Open the golfer message board application](#)
2. [Create the Entity Data Model to access the GolferMessageBoardDB on SQL Database](#)
3. [Compile the MessageBoard_WebRole project](#)

To open the Golfer Message Board application

1. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2012**, right-click **Visual Studio 2012**, and then click **Run as administrator**.
2. If the **User Account Control** dialog appears, click **Yes**.
3. Click the **File** menu, point to **Open**, and then click **Project/Solution**.
4. In **File name**, type **C:\AzureTutorials\Tutorial2.1\GolferMessageBoard\GolferMessageBoard.sln**, and then click **Open**.

To create the Entity Data Model

1. In Solution Explorer, right-click **MessageBoard_WebRole**, point to **Add**, and then click **New Item**.
2. In the **Add New Item – MessageBoard_WebRole** dialog, type and select the following values:

Name	Value
Installed Templates	Visual C# , Data
Template	ADO.NET Entity Data Model
Name	MessageBoard.edmx

3. Click **Add**
4. In the **Entity Data Model Wizard**, select **Generate from Database**, and then click **Next**.
5. In the **Choose Your Data Connection** screen of the **Entity Data Model Wizard** dialog, type and select the following values:

Name	Value
Which data connection should your application use to connect to the database?	GolferMessageBoardDBConnectionString
Select the option:	Yes, include the sensitive data in the connection string
Save entity connection settings in Web.config as:	GolferMessageBoardDBEntities

- Click **Next**.

Security Note: To properly secure the SQL Database connection string, you should encrypt the Web.config file. Encrypting the Web.config file is outside the scope of this tutorial. For an example of how to do this, see the blog series on Windows Azure [SQL Database](#)

- In the **Choose Your Database Objects and Settings** screen of the **Entity Data Model Wizard** dialog, select the following values:

Name	Value
Enable these options:	Tables Pluralize or singularize generated object names Include foreign key columns in the model
Model Namespace	GolferMessageBoardDBModel

- Click **Finish**. This adds the MessageBoard.edmx file and references to Entity Framework assemblies to the project. The MessageBoard.edmx file represents the data model and mapping to the SQL Database.
- In Solution Explorer, expand the **MessageBoard_WebRole** project and double-click the MessageBoard.edmx file. This opens the data model in the ADO.NET Entity Data Model Designer (Entity Designer).
- In the Entity Designer surface, right-click the **Message** entity, and click **Properties**.
- In the **Properties** window, update the following property values:

Name	Value
Name	Message2

Entity Set Name	Messages
-----------------	----------

Note: By default, both the Object Relational Designer (LINQ-to-SQL) and the Entity Designer (LINQ-to-Entities) generate the *Message* type in the default project namespace. To prevent errors when building the project, you need to rename one of the types to *Message2*. The entity set name is used by WCF Data Services as the name of the feed, which is changed to *Messages*.

To compile the **MessageBoard_WebRole** project

- In Solution Explorer, right-click **MessageBoard_WebRole**, and then click **Rebuild**. Make sure the project is compiled successfully.

What did I just do?

In this step, you modified the golfer message board application to add an Entity Framework data model. This data model and generated classes are used to connect to SQL Database by using the Entity Framework.

Next Steps:

You will use the WCF Data Services item template to add the code to the project that defines the OData service. This data service uses the new Entity Framework data model.

Lesson 2: Modify the Application to Add the Data Service

In this lesson, you add the code for the OData service, which uses the data model created in the previous lesson.

This template adds both a markup page (.svc) and a code-behind page to the project. It also adds the appropriate assembly references to the project.

Note: When you create a data service such that the generic type of the `DataService<T>` class is the strongly-typed `ObjectContext` of an Entity Framework data model, WCF Data Services uses the Entity Framework provider to access and change data in the database. For more information, see [Entity Framework Provider \(WCF Data Services\)](#).

Procedures

In this lesson, you will go through the following procedures:

- [Add the data service code files to the project](#)
- [Modify the data service to use the entity data model from the previous lesson](#)
- [Compile the MessageBoard_WebRole project](#)

To add WCF Data Services code files to the project

1. In Solution Explorer, right-click **MessageBoard_WebRole**, point to **Add**, and then click **New Item**.
2. In the **Add New Item – MessageBoard_WebRole** dialog, type and select the following values:

Name	Value
Installed Templates	Visual C# , Web
Template	WCF Data Service
Name	MessageBoard.svc

3. Click **Add**. This adds both the data service code files and WCF Data Services assembly references to the project.

To modify the WCF Data Services code files to use the data model

1. In **MessageBoard_WebRole**, right-click **MessageBoard.svc**, and click **View Code**.
2. In the code for the data service, locate the following comment in code:

```
/* TODO: put your data source class name here */
```

3. Replace this comment with the name of the strongly-typed `ObjectContext` of the Entity Framework data model, which is `GolferMessageBoardDBEntities`. The class definition should look like the following:

```
public class MessageBoard : DataService<GolferMessageBoardDBEntities>
```

4. Locate the **InitializeService** method and replace it with the following code:

```
// This method is called only once to initialize service-wide policies.  
public static void InitializeService(DataServiceConfiguration config)  
{  
    // Grant only the rights needed to support client applications,  
    // in this case read all and insert new messages.  
    config.SetEntitySetAccessRule("Messages", EntitySetRights.AllRead |  
        EntitySetRights.WriteAppend);  
}
```

To compile the MessageBoard_WebRole project

1. In Solution Explorer, right-click **MessageBoard_WebRole**, and then click **Rebuild**. Make sure the project is compiled successfully.

What did I just do?

In this step, you used WCF Data Services to add an OData feed to the golfer message board application, which accesses the SQL Database by using and the Entity Framework provider. This enables you to read messages as OData feeds and insert new messages by sending POST requests to the OData service.

Next Steps:

You will test and deploy the application.

Lesson 3: Test and Deploy the Application

In this lesson, you test the application in Windows Azure computer emulator, package the application, and then deploy the application to Windows Azure.

Note: You must disable feed reading view in the browser to view the raw Atom XML feed returned from the OData service.

Procedures

In this lesson, you will go through the following procedures:

1. [Disable feed reading view in the browser](#)
2. [Test the application in compute emulator](#)
3. [Generate the service package](#)
4. [Sign in to Windows Azure](#)
5. [Create a cloud service](#)
6. [Deploy the application to the staging environment](#)
7. [Test the application in the staging environment](#)
8. [Promote the application to the production environment](#)

To disable feed reading view in Internet Explorer

1. Open Internet Explorer.
2. Click **Tools** menu, and then select **Internet Options**.
3. Click the **Content** tab, click **Settings** in the **Feeds and Web Slices** section, and clear **Turn on feed view**.

Note: If you are using a Web browser other than Internet Explorer and if your browser cannot display the feed as raw XML data, you should still be able to view the feed as the source code for the page.

To test the application in the computer emulator

1. If MessageBoard is not the startup project: In Solution Explorer, right-click **MessageBoard**, and then click **Set as Startup Project**.
2. Right-click Message.svc and click **Set As Start Page**.
3. From the **Debug** menu, click **Start Debugging**.
4. Switch to the browser window to view the service document for the OData service.
5. In the address bar, append the entity set name **Messages** to the root URI of the data service, and then press return. This displays the Messages OData feed, which returns all messages in the database.

Note: If the feed returned appears empty, then you will need to first insert some data into the SQL Database. The easiest way to add new messages to the database is by entering them in the default.aspx page of the Web application.

6. Close the browser window to stop the debugger and shut down the development in the compute emulator.

After the application is tested successfully in the compute emulator environment, the next step is to create the service package and then deploy the application to Windows Azure.

To generate the service package

1. In Solution Explorer, right-click the **MessageBoard** cloud project, and then click **Package**.
2. In Package Windows Azure Application, select the following values:

Name	Value
Service configuration	Cloud
Build configuration	Release

3. Click **Package**. 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. The default directory is `C:\AzureTutorials\Tutorial2.1\GolferMessageBoard\MessageBoard\bin\Release\app.publish`. Make sure to write down the path. You will need it later in this lesson.

To sign in to Windows Azure

1. Open a Web browser and browse to <http://windows.azure.com/>. This is the Windows Azure Management Portal.
2. Sign in using the Windows Live ID associated with your Windows Azure account.

Note: If you haven't had a Windows Azure Platform subscription, see the [Provisioning Windows Azure](#) section of this tutorial.

You need to create a cloud service for the Web role. If you have created a cloud service account in [tutorial 1](#) or [tutorial 2](#), you can skip this step.

To create a cloud service

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. On the bottom left corner of the portal page, click **NEW**, click **COMPUTE**, click **CLOUD SERVICE**, and then click **CUSTOM CREATE**.
3. Type or select the following values.

Name	Value
URL	<yourname>gmb Note: The URL prefix must be unique.
REGION/AFFINITY GROUP	(For better performance, select the same region as the one you chose for the storage service, or use an affinity group.)
Deployment a cloud service package now	(not selected)

- Click the check sign on the bottom right corner. Wait until the status changes to **Created**. This process could take several minutes.

To deploy the application to the staging environment

- From the portal, in the left pane, click **CLOUD SERVICES**.
- In the middle pane, click the cloud service you just created. The default name is <yourname>gmb..
- From the top of the page, click **STAGING**.
- Click **UPLOAD A NEW STAGING DEPLOYMENT** or UPDATE if you have deployed a Golfer Message Board application from other tutorials before.
- Type or select the following value.

Name	Value
Deployment name	v1.1.1.0
Package location	C:\AzureTutorials\Tutorial2.1\GolferMessageBoard\MessageBoard\bin\Release\app.publish\MessageBoard.cspkg
Configuration file	C:\AzureTutorials\Tutorial2.1\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg
Deploy even if one or more roles contain a single instance	(Selected). You can always increase the number of instances from the portal.
Start deployment (only available when deploying)	(Selected)

a new application)	
--------------------	--

6. Click the check sign on the bottom right corner of the page.
7. Wait until the upload process is completed. The process can take several minutes to complete.

To test the application in the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the right side of the page, click the site URL.
5. Test the application by entering one or more entries.

After the application is working correctly in the staging environment, you are ready to promote it to the production environment.

To promote the application to production

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the bottom of the page, click **SWAP**.
5. On the right, click **YES**.
6. On the top of the page, click **PRODUCTION**. It takes several minutes to complete the operation.
7. On the right, click the SITE URL.
8. In the **Properties** pane, click the URL in the **DNS name** box. The application is opened in a new browser tab or a new browser window depending on your browser configuration.

Note: Some DNS services take longer to replicate the records. If you get a page not found error, you might need to try browsing to the URL again in a few minutes.

9. Test the OData feed in the production environment by accessing the Messages feed as before.

What did I just do?

In this step, you tested and deployed the golfer message board OData service to Windows Azure.

Next Steps:

Congratulations! You have completed the tutorial 2.1. Now that you have deployed the data service, you can create client applications that consume the Messages OData feed. For more information on how to create client applications that access OData services, see one of the following topics in the MSDN library:

- [WCF Data Services Client Library](#) – used to consume an OData feed in a .NET Framework client application.
- [WCF Data Services \(Silverlight\)](#) – used to consume an OData feed in a Silverlight client application.
- [Open Data Protocol \(OData\) Client for Windows Phone](#) – used to consume an OData feed in a Windows Phone client application

For a complete list of client libraries that support OData, see the [OData SDK](#).

Tutorial 3: Using Windows Azure Blob Service

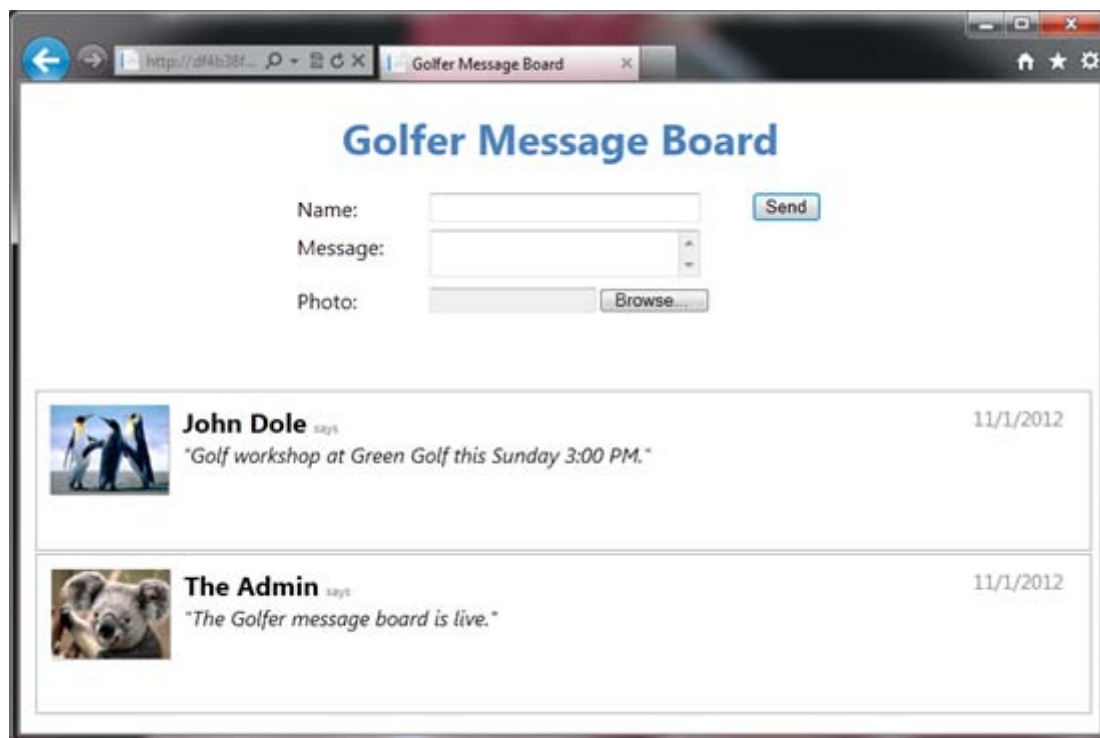
This tutorial demonstrates how to use the Windows Azure Blob Service.

Overview

In [tutorial 1](#), you created a simple golfer message board application. In the application, a Web role provides the front-end that allows golfers to view the contents of the message board and add new entries. Each entry contains a name and a message. When golfers post a new message, the Web role creates an entry using the Table service that contains the information entered by the golfers. The Web role also renders this information to the browser so golfers can view the content of the message board.

Tutorial 3 is based on tutorial 1. In this tutorial, you will modify the application you created in tutorial 1. In addition to golfer name and a short message for each message ported to the message board, you will also add an image. You will keep using the Table service for the text portion of the messages, but use the Blob service for images.

Here is a screenshot of the application.



Note: Completing tutorial 1 is not a pre-requisite for this tutorial. However, it helps with understanding the scenario.

Objectives

In this tutorial, you will learn how to:

- Use the Blob service

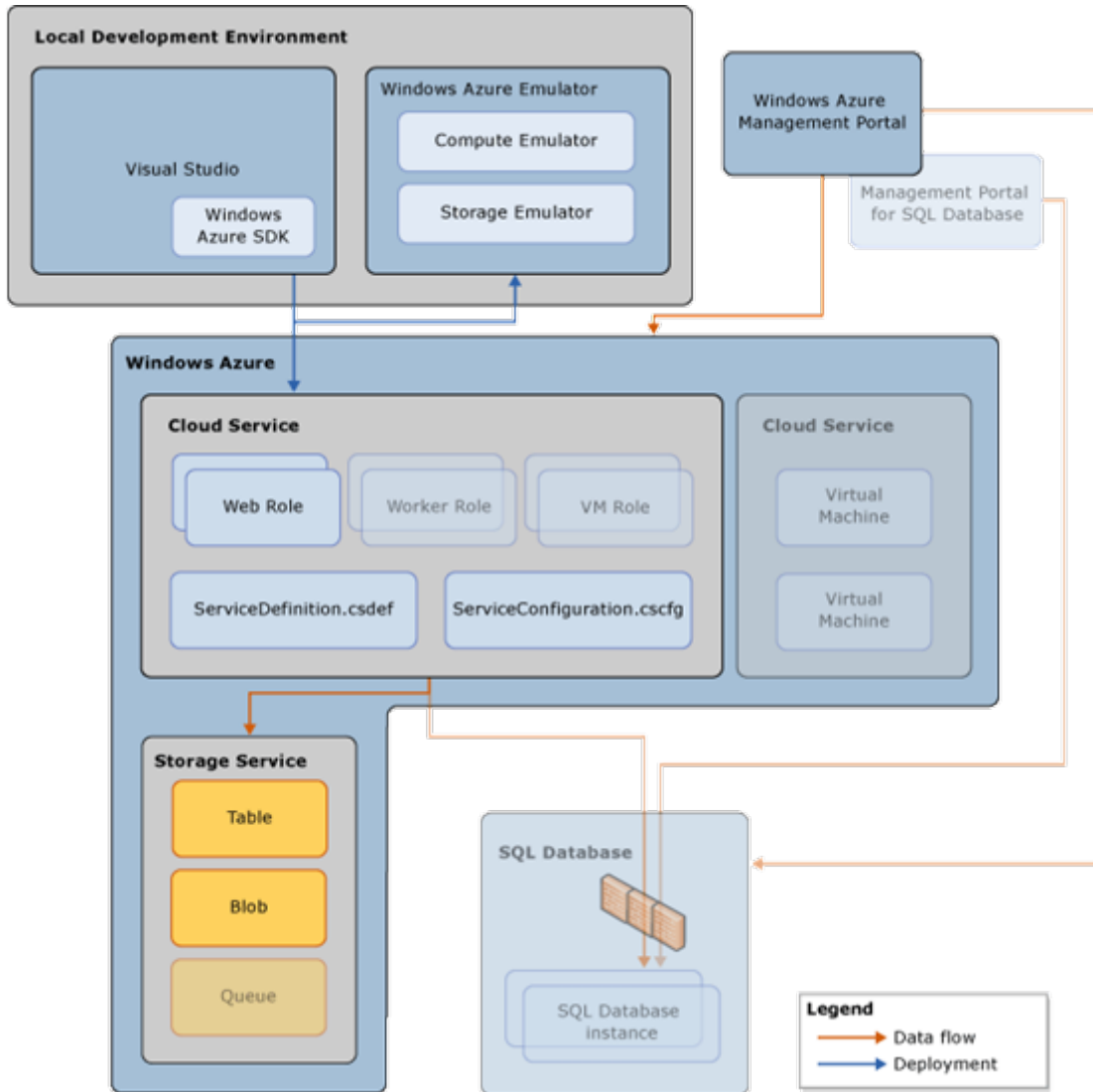
Prerequisites

Note the following requirements before you begin this lesson:

- Before you begin this step you must complete the steps in [Before You Begin The Tutorials](#).

Understanding the Architecture

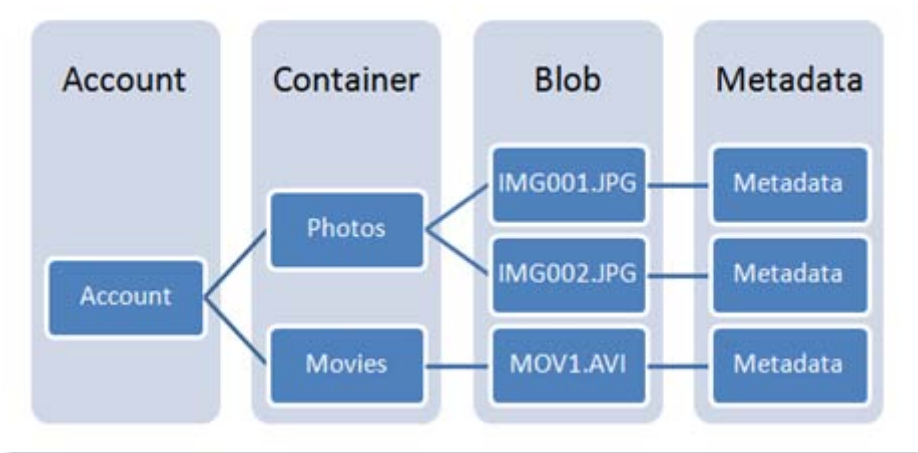
The following diagram illustrates the development components and the runtime components involved in this tutorial:



- You develop the Windows Azure application using Visual Studio and Windows Azure SDK.
- You deploy the application to Windows Azure Emulator for testing, and to Windows Azure.
- A Windows Azure project includes two configuration files: ServiceDefinition.csdef and ServiceConfiguration.cscfg. These files are packaged with your Windows Azure application and deployed to Windows Azure.
- 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. A worker role is a role that is useful for generalized development, and may perform background processing for a Web role. The Tutorial 3 application contains a Web role project. For more information, see *Overview of a Windows Azure Application* at <http://msdn.microsoft.com/en-us/library/gg432976.aspx>.
- The Windows Azure storage services provide persistent, redundant storage in the cloud. The storage services include these fundamental services: Blob service, Queue service, and Table service. The Table service and the Blob service are used in this tutorial. For more information, see [Data Storage Offerings on the Windows Azure Platform](#).

Understanding the Blob Service

The Windows Azure Blob service enables applications to store large objects, up to 1TB each. It supports a massively scalable blob system via the Windows Azure CDN, where hot blobs will be served from many servers to scale out and meet the traffic needs of your application. Furthermore, the system is highly available and durable. You can always access your data from anywhere at any time, and the data is replicated at least 3 times for durability. In addition, strong consistency is provided to ensure that the object is immediately accessible once it is added or updated; a subsequent read will immediately see the changes made from a previously committed write.



An application must use a valid account to access Windows Azure Storage. An account can have many Blob containers. Each container provides a grouping of a set of blobs.

In this Article

- [Lesson 1: Modify the application to use the Blob service](#)
- [Lesson 2: Test and deploy the application](#)

Lesson 1: Modify the Application to Use the Blob Service

In this lesson, you modify the golfer message board application to use the Blob service for storing images.

Procedures

In this lesson, you will go through the following procedures:

1. [Open the golfer message board application](#)
2. [Modify the table schema](#)
3. [Modify the data source](#)
4. [Modify default.aspx](#)
5. [Modify default.aspx.cs](#)

To open the Golfer Message Board application

1. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2012**, right-click **Visual Studio 2012**, and then click **Run as administrator**.
2. If the **User Account Control** dialog appears, click **Yes**.
3. Click the **FILE** menu, point to **Open**, and then click **Project/Solution**.
4. In File name, type **C:\AzureTutorials\Tutorial3\GolferMessageBoard\GolferMessageBoard.sln**, and then click **Open**.

In [tutorial 1](#), you defined a fixed schema for the Table service to store messages. You must modify the schema to include a new data member for storing the URL to the image in the Blob service.

To modify the table schema

1. In Solution Explorer, expand **MessageBoard_Data**, and then double-click **MessageBoardEntry.cs** to open the file.
2. At the end of the class, define a new data member by adding the following line:

```
public string ImageURL {get; set;}
```

The file looks like the following after modification:

```

using ...

namespace MessageBoard_Data
{
    public class MessageBoardEntry
        : Microsoft.WindowsAzure.StorageClient.TableServiceEntity
    {
        //In addition to the properties required by the data model, every entity in table ...
        public MessageBoardEntry(...)

        public string GolferName { get; set; }
        public string GolferMessage { get; set; }
        public string ImageURL {get; set;}
    }
}

```

3. Press **Ctrl+S** to save the Default.aspx file.

Modify the data source

1. In Solution Explorer, expand **MessageBoard_Data**, and then double-click **MessageBoardDataSource.cs** to open the file.
2. At the beginning of the file, add the following namespace declaration:

```
using System.IO
```

3. At the beginning of the MessageBoardDataSource class, add new data members by inserting the code snippet **Tutorial03-Lesson01-Task03_MessageBoardDataSourceDataMembers**:

```
private const string messageImageBlobName = "golfermessageboardpics";
private CloudBlobClient blobClient;
private CloudBlobContainer blobContainer;
```

Note: All letters in a container name must be lowercase. For more restrictions on container names, see [Naming Containers, Blobs, and Metadata](#).

4. At the end of the default constructor, add code to create a blob container by inserting the code snippet **Tutorial03-Lesson01-Task03_MessageBoardDataSourceDefaultConstructor**:

```
blobClient = storageAccount.CreateCloudBlobClient();
blobContainer = blobClient.GetContainerReference(messageImageBlobName);
blobContainer.CreateIfNotExist();

var permissions = blobContainer.GetPermissions();
permissions.PublicAccess = BlobContainerPublicAccessType.Container;
blobContainer.SetPermissions(permissions);
```

5. Inside the class, add a new method for adding a blob by inserting the code snippet **Tutorial03-Lesson01-Task03_MessageBoardDataSourceAddBlob**:

```
public string AddBlob(string fileExtension, string fileContentType, Stream
fileContent)
{
    // upload the image to blob storage
    string uniqueBlobName = string.Format(messageImageBlobName +
"/image_{0}{1}", Guid.NewGuid().ToString(), fileExtension);
```

```

CloudBlockBlob blob = blobClient.GetBlockBlobReference(uniqueBlobName);
blob.Properties.ContentType = fileContentType;
blob.UploadFromStream(fileContent);
return blob.Uri.ToString();
}

```

The file looks like the following after modification:

```

using ...

namespace MessageBoard_Data
{
    public class MessageBoardDataSource
    {
        private const string messageTableName = "MessageTable";
        private const string messageImageBlobName = "golfermessageboardpics";
        private const string connectionStringName = "DataConnectionString";
        private static CloudStorageAccount storageAccount;
        private CloudTableClient tableClient;
        private CloudBlobClient blobClient;
        private CloudBlobContainer blobContainer;

        //The default constructor initializes the storage account by reading its settings from ...
        public MessageBoardDataSource()
        {
            string connectionString = RoleEnvironment.GetConfigurationSettingValue(connectionStringName);

            storageAccount = CloudStorageAccount.Parse(connectionString);
            tableClient = new CloudTableClient(storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);
            tableClient.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(1));
            tableClient.CreateTableIfNotExist(messageTableName);

            //create blob container
            blobClient = storageAccount.CreateCloudBlobClient();
            blobContainer = blobClient.GetContainerReference(messageImageBlobName);
            blobContainer.CreateIfNotExist();

            //set blob container permissions
            var permissions = blobContainer.GetPermissions();
            permissions.PublicAccess = BlobContainerPublicAccessType.Container;
            blobContainer.SetPermissions(permissions);
        }

        //The GetMessageBoardEntries method retrieves today's message board entries by constructing ...
        public IEnumerable<MessageBoardEntry> GetEntries()...

        //The AddMessageBoardEntry method inserts new entries into the table.
        public void AddEntry(MessageBoardEntry newItem)...

        public string AddBlob(string fileExtension, string fileContentType, Stream fileContent)
        {
            // upload the image to blob storage
            string uniqueBlobName = string.Format(messageImageBlobName + "/image_{0}{1}", Guid.NewGuid().ToString(), fileExtension);
            CloudBlockBlob blob = blobClient.GetBlockBlobReference(uniqueBlobName);
            blob.Properties.ContentType = fileContentType;
            blob.UploadFromStream(fileContent);
            return blob.Uri.ToString();
        }
    }
}

```

6. Press **Ctrl+S** to save the file.
7. In Solution Explorer, right-click **MessageBoard_Data**, and then click **Rebuild**.

Default.aspx presents the message board Web interface. You must modify the page so that golfers can upload an image, and the page can display images along with messages. See the screenshot at the beginning of the tutorial.

To modify the default.aspx file

1. In Solution Explorer, expand **MessageBoard_WebRole**, right-click **Default.aspx**, and then click **View Markup**.

- Find the `<div>` tag with `class="inputSection"`. At the end of the `<dl>` tag, add the code for uploading images by inserting the code snippet **Tutorial03-Lesson01-Task03_WebRoleFileUpload**.

```

<dt>
<label for="FileUpload1">Photo:</label>
</dt>
<dd>
<asp:FileUpload ID="FileUpload1" runat="server" size="16" />
<asp:RequiredFieldValidator ID="PhotoRequiredValidator" runat="server"
    ControlToValidate="FileUpload1" Text="*" />
<asp:RegularExpressionValidator ID="PhotoRegularExpressionValidator"
    runat="server"
    ControlToValidate="FileUpload1" ErrorMessage="Only .jpg or .png files are
    allowed"
    ValidationExpression="([a-zA-Z\\]|\.|\.jpg|\.JPG|\.png|\.PNG)$" />
</dd>

```

If you don't see the code snippet, follow the instructions in [Before You Begin the Tutorial](#) to copy the code snippets.

The `<div>` tag looks like the following after the modification:

```

<div class="inputSection">
  <dl>
    <dt>
      <label for="NameLabel">
        Name:</label></dt>
      <dd>...</dd>
    <dt>
      <label for="MessageLabel">
        Message:</label>
      </dt>
      <dd>...</dd>
    <dt>
      <label for="FileUpload1">
        Photo:</label></dt>
      <dd>
        <asp:FileUpload ID="FileUpload1" runat="server" size="16" />
        <asp:RequiredFieldValidator ID="PhotoRequiredValidator" runat="server"
          ControlToValidate="FileUpload1" Text="*" />
        <asp:RegularExpressionValidator ID="PhotoRegularExpressionValidator" runat="server"
          ControlToValidate="FileUpload1" ErrorMessage="Only .jpg or .png files are allowed"
          ValidationExpression="([a-zA-Z\\]|\.|\.jpg|\.JPG|\.png|\.PNG)$" />
      </dd>
    </dl>
    <div class="submitSection">...</div>
  </div>

```

- Find the `<div>` tag with `class="signature"`. At the beginning of the `<div>` tag, add the code for displaying images by inserting the code snippet **Tutorial03-Lesson01-Task03_WebRoleDisplayImage**:

```

<div class="signatureImage">
" alt="<%=# Eval("GolferName") %>" />
</div>

```

The `<div>` tag looks like the following after the modification:

```
0:00 <div class="signature">
0:00 <div class="signatureImage">
0:00 " alt="<%= Eval("GolferName") %>" />
0:00 </div>
0:00 <div class="signatureDescription">
0:00 <div class="signatureName">
0:00 <%= Eval("GolferName") %>
0:00 </div>
0:00 <div class="signatureSays">
0:00 says
0:00 </div>
0:00 <div class="signatureDate">
0:00 <%= ((DateTime)Eval("Timestamp")).ToShortDateString() %>
0:00 </div>
0:00 <div class="signatureMessage">
0:00 <%= Eval("GolferMessage") %>
0:00 </div>
0:00 </div>
0:00 </div>
```

4. Press **Ctrl+S** to save the Default.aspx file.

Next, you modify the default.aspx.cs for creating and using the Blob service. In the code, you define a lock to prevent the Blob service being created multiple times.

To modify the default.aspx.cs file

1. In Solution Explorer, expand **MessageBoard_WebRole**, right-click **Default.aspx**, and then click **View Code**.
2. At the top of the file, insert the following namespace declaration:

```
using System.IO;
```

3. Inside the **btnSend_Click()** method, add or modify the lines highlighted in the screenshot:

```

protected void btnSend_Click(object sender, EventArgs e)
{
    string blobURI;

    MessageBoardDataSource ds = new MessageBoardDataSource();

    //create a blob
    blobURI = ds.AddBlob(
        Path.GetExtension(FileUpload1.FileName),
        FileUpload1.PostedFile.ContentType,
        FileUpload1.FileContent);

    // create a new entry in table storage
    MessageBoardEntry entry = new MessageBoardEntry() {
        GolferName = txtName.Text,
        GolferMessage = txtMessage.Text,
        ImageURL= blobURI };
    ds.AddEntry(entry);

    txtName.Text = "";
    txtMessage.Text = "";

    dlMessages.DataBind();
}

```

4. Press **Ctrl+S** to save the file.
5. In Solution Explorer, right-click **MessageBoard_WebRole**, and then click **Rebuild**.

What did I just do?

In this step, you introduced the Blob service to the golfer message board application.

Next Steps:

You will test the application and prepare the service package for the application.

Lesson 2: Test and Deploy the Application

In this lesson, you test the application in the compute emulator environment, package the application, and then deploy the application to Windows Azure.

Procedures

In this lesson, you will go through the following procedures:

1. [Test the application](#)
2. [Generate the service package](#)
3. [Sign in to Windows Azure](#)
4. [Create a storage account](#). You need a storage account to utilize Windows Azure storage services, for example the table service.
5. [Create a cloud service](#). The cloud service is used to host the message board application.
6. [To configure the ServiceConfiguration.Cloud.cscfg file](#)
7. [Deploy the application to the staging environment](#)
8. [Test the application](#)
9. [Promote the application to the production environment](#)

To test the application

1. In Solution Explorer, right-click **MessageBoard**, and then click **Set as Startup Project**.
2. From the **Debug** menu, click **Start Debugging**. You will see a compute emulator icon added to the notification area of taskbar, and a new Internet Explorer window showing the Golfer Message Board application.
3. Switch to Internet Explorer to view the message board application.
4. Add a few entries to the message board by entering your name and a message before clicking **Send**.
5. Close the Internet Explorer window.

After the application is tested successfully in the compute emulator environment, the next step is to create the service package and then deploy the application to Windows Azure.

To generate the service package

1. In Solution Explorer, right-click the **MessageBoard** cloud project, and then select **Package**.
2. In the **Package Windows Azure Application** dialog, select the following values:

Name	Value
Service configuration	Cloud
Build configuration	Release

3. Click **Package**. 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. The default directory is `C:\AzureTutorials\Tutorial3\GolferMessageBoard\MessageBoard\bin\Release\app.publish`. Make sure to write down the path. You will need it in the deployment lesson.

You will get a few warning messages about "DataConnectionString" set up to use the local storage emulator. You can ignore these warning for now.

For deploying the golfer message board application, you must have a *storage account* for accessing the Windows Azure storage services, and a *cloud service*, which is a container for service deployments in Windows Azure. For better performance, you might want to create an *affinity group* to group the service and the storage accounts within a subscription according to geo-location.

To sign in to Windows Azure

1. Open a Web browser and browse to <http://windows.azure.com/>. This is the Windows Azure Management Portal.
2. Sign in using the Windows Live ID associated with your Windows Azure account.

Note: If you haven't had a Windows Azure Platform subscription, see the [Provisioning Windows Azure](#) section of this tutorial.

The next step is to create a storage account for using Windows Azure table service and blob service. If you have created a storage account in [tutorial 1](#), you can skip this step.

To create a storage account for the golfer message board application to store its data

1. From the portal, in the left pane, click **STORAGE**.
2. On the bottom left corner, click **NEW**, click **STORAGE**, and then click **QUICK CREATE**.
3. Type or select the following values.

Name	Value
URL	<yourname>gmb (i.e. johndolegmb)
REGION/AFFINITY GROUP	(Choose the region where you wish your service to run, most likely, the one that is closest to your current location. Select an affinity group instead of a region if you want your storage services to be in the same data center with the cloud service that you will create in the next step. To create an affinity group, open the NETWORKS area of the Management Portal, click AFFINITY GROUPS, and then click CREATE.)

4. Click the check mark on the bottom right corner. Wait until the status of the storage account changes to **Online**. The process can take several minutes.

- Once the storage account is created, click the storage account from the storage account list to select it, and then click **MANAGE KEYS** on the bottom of the page.
- Record the **STORAGE ACCOUNT NAME** and **PRIMARY ACCESS KEY**. Later in the tutorial, you will need to configure your cloud service to use storage account by specifying the account name and the access key.
- Close the dialog.

If you have published other tutorial projects before, you can either update the existing cloud service, or create a new cloud service. If you choose to create a new cloud service, you must give a different cloud service URL.

To create a cloud service

- From the portal, in the left pane, click **CLOUD SERVICES**.
- On the bottom left corner of the portal page, click **NEW**, click **COMPUTE**, click **CLOUD SERVICE**, and then click **CUSTOM CREATE**.
- Type or select the following values.

Name	Value
URL	<yourname>gmb Note: The URL prefix must be unique.
REGION/AFFINITY GROUP	(For better performance, select the same region as the one you chose for the storage service, or use an affinity group.)
Deployment a cloud service package now	(not selected)

- Click the check sign on the bottom right corner. Wait until the status changes to **Created**. This process could take several minutes.

When you create and test the application locally, the application is configured to use the development storage. Now you have created a storage account, you can configure the application to use the storage account before deploying the application to Windows Azure. The configuration information is in the `ServiceConfiguration.Cloud.cscfg` file. This file was created when you generated the service package.

To configure the `ServiceConfiguration.Cloud.cscfg` file

- Use Notepad to open `C:\AzureTutorials\Tutorial3\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg`. This path can be different if you installed the tutorial files into a different folder than the c root directory. Notice the value of `DataConnectionString` is `"UseDevelopmentStorage=true"`.

2. Change the value of **DataConnectionString** to **DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your storage account primary access key>**. Replace <your storage account name> and <your storage account access key> accordingly.
3. Change the value of **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString** to **DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your storage account primary access key>**. You must replace <your storage account name> and <your storage account primary access key> with the actual account name and account key for the storage account that you created earlier in the tutorial.
4. Optionally, you can increase the Instances count to scale out the application.
5. Save the file and close Notepad.

To deploy the application to the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you just created. The default name is <yourname>gmb..
3. From the top of the page, click **STAGING**.
4. Click **UPLOAD A NEW STAGING DEPLOYMENT**, or **UPDATE** on the bottom of the page if you have deployed a tutorial application before.
5. From **Upload a package**, type or select the following value.

Name	Value
Deployment name	v1.2.0.0
Package location	C:\AzureTutorials\Tutorial3\GolferMessageBoard\MessageBoard\bin\Release\app.publish\MessageBoard.cspkg
Configuration file	C:\AzureTutorials\Tutorial3\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg
Deploy(or Update) even if one or more roles contain a single instance	(Selected). You can always increase the number of instances from the portal.
Start deployment (this option is not available for update a service)	(Selected)

6. Click the check sign on the bottom right corner of the page.

7. Wait until the upload process is completed. The process can take several minutes to complete. Notice the DEPLOYMENT NAME is changed to v1.2.0.0.

To test the application in the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the right side of the page, click the site URL.
5. Test the application by entering one or more entries.

After the application is working correctly in the staging environment, you are ready to promote it to the production environment.

To promote the application to production

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the bottom of the page, click **SWAP**.
5. On the right, click **YES**.
6. On the top of the page, click **PRODUCTION**. It takes several minutes to complete the operation.
7. On the right, click the SITE URL.
8. In the **Properties** pane, click the URL in the **DNS name** box. The application is opened in a new browser tab or a new browser window depending on your browser configuration.

Note: Some DNS services take longer to replicate the records. If you get a page not found error, you might need to try browsing to the URL again in a few minutes.

9. Test the application in the production environment by entering one or more entries.

What did I just do?

In this step, you tested and deployed the golfer message board to Windows Azure.

Next Steps:

Congratulations! You have completed tutorial 3. [Tutorial 4](#) shows you how to use another Windows Azure storage service called Windows Azure Queue service.

Tutorial 4: Using Windows Azure Worker Role and Windows Azure Queue

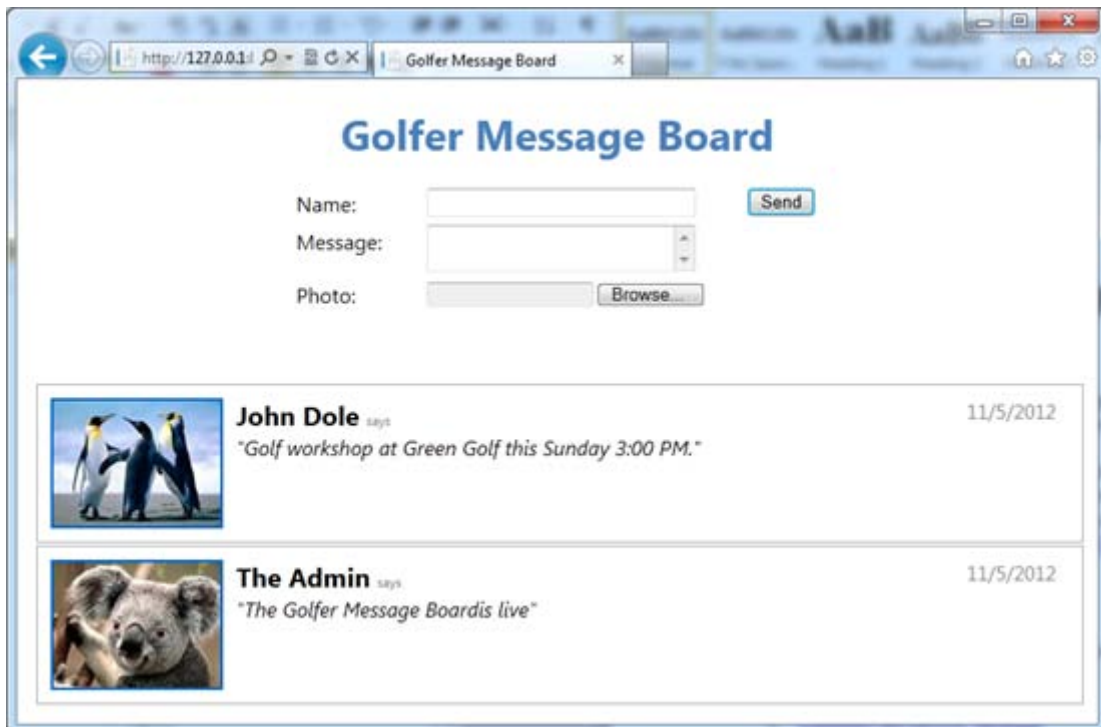
This tutorial demonstrates using a worker role and Windows Azure queue service.

Overview

In [tutorial 1](#) and [tutorial 3](#), you created a simple golfer message board application. In the application, a Web role provides the front-end that allows golfers to view the contents of the message board and add new entries. Each entry contains a name, a message and an image. When golfers post new messages, a Web role stores the text portion of the message using the Table service, and the image using the Blob service. The Web role also renders this information to the browser so golfers can view the content of the message board.

You have probably noticed that the images are displayed in the original size, which takes a lot of space. In tutorial 4, you will expand the application with a new worker role in the background generating thumbnails based on the images. When the messages are listed, only the thumbnails are used.

Here is a screenshot of the application.



Note: Completing tutorial 1 and tutorial 3 are not pre-requisites for this tutorial. However, it helps with understanding the scenario.

Objectives

In this tutorial, you will learn how to:

- Use the Queue service
- Understand worker role

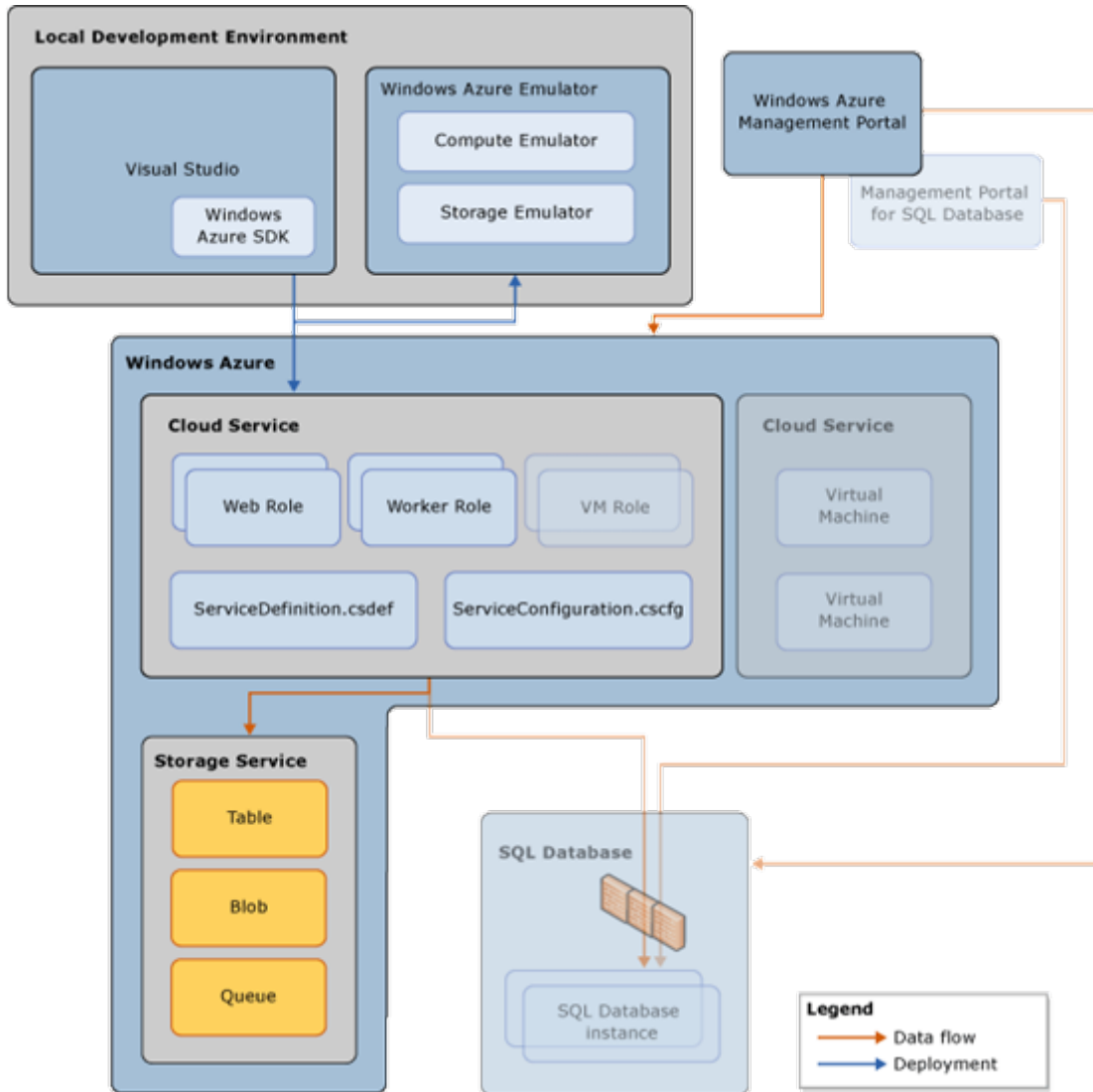
Prerequisites

Note the following requirements before you begin this lesson:

- Before you begin this step you must complete the steps in [Before You Begin the Tutorials](#).

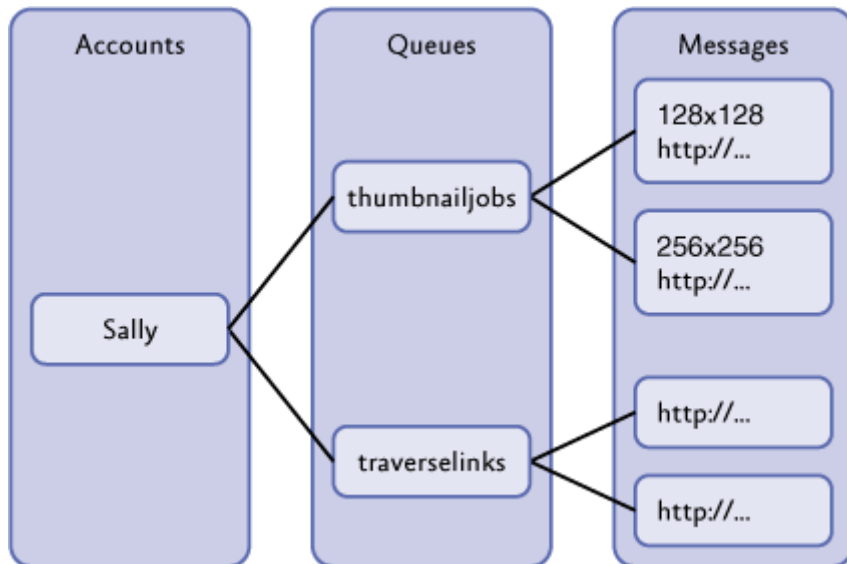
Understanding the Architecture

The following diagram illustrates the development components and the runtime components involved in this tutorial:



- You develop the Windows Azure application using Visual Studio and Windows Azure SDK.
- You deploy the application to Windows Azure Emulator for testing, and to Windows Azure.
- A Windows Azure project includes two configuration files: ServiceDefinition.csdef and ServiceConfiguration.cscfg. These files are packaged with your Windows Azure application and deployed to Windows Azure.
- 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. A worker role is a role that is useful for generalized development, and may perform background processing for a Web role. This tutorial contains a Web role and a worker role. For more information, see *Overview of a Windows Azure Application* at <http://msdn.microsoft.com/en-us/library/gg432976.aspx>.
- The Windows Azure storage services provide persistent, redundant storage in the cloud. The storage services include these fundamental services: Blob service, Queue service, and Table service. The Table service and the Blob service are used in this tutorial. For more information, see [Data Storage Offerings on the Windows Azure Platform](#).
- You can use the portal to administrate Windows Azure platform resources.

Understanding the Queue Service



The Queue service architecture consists of a three-level hierarchy: accounts, queues, and messages. A Windows Azure storage account encompasses the Blob, Queue, and Table services. A queue is a logical destination for sending messages. There can be any number of queues in an account in the Queue service. A queue stores messages and makes them available to applications. Messages are stored in queues. There is no limit to the number of messages that can be stored in a queue, but the size of each individual message cannot exceed 8KB. To accommodate large object messages, you can put the large object in a blob and then send the URI of that object as a message to a queue. For more information, see [Data Storage Offerings on the Windows Azure Platform](#), and *Windows Azure Queue – Programming Queue Storage* at <http://go.microsoft.com/fwlink/?LinkId=153402>.

In this Article

1. [Lesson 1: Modify the Message Data Model](#)
2. [Lesson 2: Modify the Web Role](#)
3. [Lesson 3: Add a Worker Role for Background Processing](#)
4. [Lesson 4: Test and Deploy the Application](#)

Lesson 1: Modify the Message Data Model

In tutorial 1, you created the data model in the MessageBoard_Data project. In this lesson, you will modify the schema so that it includes an additional data member for the thumbnail URL. You will also add a member function for updating the thumbnail URLs. This function will be called from the worker role after the worker role generates the thumbnails.

Procedures

In this lesson, you will go through the following procedures:

1. [Open the golfer message board application](#)
2. [Modify the table schema](#)
3. [Modify the implementation of the object](#)
4. [Rebuild the project](#)

To open the Golfer Message Board application

1. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2012**, right-click **Visual Studio 2012**, and then click **Run as administrator**.
2. If the **User Account Control** dialog appears, click **Yes**.
3. Click the **FILE** menu, point to **Open**, and then click **Project/Solution**.
4. In File name, type **C:\AzureTutorials\Tutorial4\GolferMessageBoard\GolferMessageBoard.sln**, and then click **Open**.

In tutorial 1, you defined a fixed schema for the table to store messages. You must modify the schema to include a new data member for storing the URL to the image in the Blob service.

To modify the table schema

1. In Solution Explorer, expand **MessageBoard_Data**, and then double-click **MessageBoardEntry.cs** to open the file.
2. At the end of the class, define a new data member by adding the following line:

```
public string ThumbnailURL {get; set;}
```

The file looks like the following after modification:

```

using ...

namespace MessageBoard_Data
{
    public class MessageBoardEntry
        : Microsoft.WindowsAzure.StorageClient.TableServiceEntity
    {
        //In addition to the properties required by the data model, every entity in table ...
        public MessageBoardEntry()...

        public string GolferName { get; set; }
        public string GolferMessage { get; set; }
        public string ImageURL { get; set; }
        public string ThumbnailURL { get; set; }
    }
}

```

3. Press **Ctrl+S** to save the MessageBoardEntry.cs file.

To modify the implementation of the object

1. In Solution Explorer, right-click the **MessageBoard_Data** project, and then click **Add Reference**.
2. In the **Reference Manager** dialog, expand **Assemblies**, and then click **Framework**.
3. Select **System.Drawing**, and then click **OK**.
4. In Solution Explorer, expand **MessageBoard_Data**, and then double-click **MessageBoardDataSource.cs** to open the file.
5. In the beginning of the file, add the following namespace declarations by inserting the code snippet **Tutorial04-Lesson01-Task03_MessageBoardDataSourceNameSpaceDeclaration**:

```

using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

```

6. In the beginning of the class, add the following data members by inserting the code snippet **Tutorial04-Lesson01-Task03_MessageBoardDataSourceDataMembers**:

```

private const string messageQueueName = "golfermessageboardqueue"; //queue name
must be in lower case
private CloudQueueClient queueClient;
private CloudQueue queue;

```

7. At the end of the constructor, add the following lines of code to create the queue by inserting the code snippet **Tutorial04-Lesson01-Task03_MessageBoardDataSourceConstructor**:

```

//create queue
queueClient = storageAccount.CreateCloudQueueClient();
queue = queueClient.GetQueueReference(messageQueueName);
queue.CreateIfNotExist();

```

8. At the end of the class, add a new member functions for updating thumbnail URL by inserting the code snippet **Tutorial04-Lesson01-Task03_MessageBoardDataSourceMethods**:

```

//add message to the queue
public void EnQueue(string uri, MessageBoardEntry entry)

```

```

{
    // queue a message to process the image
    CloudQueueMessage message = new
CloudQueueMessage(String.Format("{0},{1},{2}", uri, entry.PartitionKey,
entry.RowKey));
    queue.AddMessage(message);
}

public void ProcessQueueMessage()
{
    // retrieve a new message from the queue
    CloudQueueMessage msg = queue.GetMessage();

    if (msg != null)
    {
        // parse message retrieved from queue
        var messageParts = msg.AsString.Split(new char[] { ',' });
        var imageBlobUri = messageParts[0];
        var partitionKey = messageParts[1];
        var rowkey = messageParts[2];

        string thumbnailBlobUri =
System.Text.RegularExpressions.Regex.Replace(imageBlobUri,
"([^\.\.]+)(\\.([^\.\.]+)?$", "$1-thumb$2");
        CloudBlob inputBlob = blobContainer.GetBlobReference(imageBlobUri);
        CloudBlob outputBlob = blobContainer.GetBlobReference(thumbnailBlobUri);
        using (BlobStream input = inputBlob.OpenRead())
        using (BlobStream output = outputBlob.OpenWrite())
        {
            CreateThumbnail(input, output);

            // commit the blob and set its properties
            output.Commit();
            outputBlob.Properties.ContentType = "image/jpeg";
            outputBlob.SetProperties();

            // update the entry in the table to point to the thumbnail
            UpdateThumbnailURL(partitionKey, rowkey, thumbnailBlobUri);

            queue.DeleteMessage(msg);
        }
    }
}

//create thumbnail for an image
private void CreateThumbnail(Stream input, Stream output)
{
    int width;
    int height;
    var originalImage = new Bitmap(input);
    if (originalImage.Width > originalImage.Height)
    {
        width = 128;
        height = 128 * originalImage.Height / originalImage.Width;
    }
    else
    {
        height = 128;
        width = 128 * originalImage.Width / originalImage.Height;
    }
    var thumbnailImage = new Bitmap(width, height);

```

```

using (Graphics graphics = Graphics.FromImage(thumbnailImage))
{
    graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;
    graphics.SmoothingMode = SmoothingMode.AntiAlias;
    graphics.PixelOffsetMode = PixelOffsetMode.HighQuality;
    graphics.DrawImage(originalImage, 0, 0, width, height);
}

thumbnailImage.Save(output, ImageFormat.Jpeg);
}

//update the thumbnail URL property for an entry.

public void UpdateThumbnailURL(string partitionKey, string rowKey, string
thumbUrl)
{
    TableServiceContext tableServiceContext =
tableClient.GetDataServiceContext();

    var results = from g in
tableServiceContext.CreateQuery<MessageBoardEntry>(messageTableName)
    where g.PartitionKey == partitionKey && g.RowKey == rowKey
    select g;

    var entry = results.FirstOrDefault<MessageBoardEntry>();
    entry.ThumbnailURL = thumbUrl;
    tableServiceContext.UpdateObject(entry);
    tableServiceContext.SaveChanges();
}

```

The file looks like the following after modification:


```

using ...

namespace MessageBoard_Data
{
    public class MessageBoardDataSource
    {
        private const string messageTableName = "MessageTable";
        private const string messageImageBlobName = "golfermessageboardpics"; //container name must be in lower case
        private const string connectionStringName = "DataConnectionString";
        private const string messageQueueName = "golfermessageboardqueue"; //queue name must be in lower case

        private static CloudStorageAccount storageAccount;

        private CloudTableClient tableClient;
        private CloudBlobClient blobClient;
        CloudBlobContainer blobContainer;
        private CloudQueueClient queueClient;

        private CloudQueue queue;

        //The default constructor initializes the storage account by reading its settings from ...
        public MessageBoardDataSource(...

        //The GetMessageBoardEntries method retrieves today's message board entries by constructing ...
        public IEnumerable<MessageBoardEntry> GetEntries(...

        //The AddMessageBoardEntry method inserts new entries into the table.
        public void AddEntry(MessageBoardEntry newItem)...

        public string AddBlob(string fileExtension, string fileContentType, Stream fileContent)...

        //add message to the queue
        public void EnQueue(string uri, MessageBoardEntry entry)...

        public void ProcessQueueMessage(...

        //create thumbnail for an image
        private void CreateThumbnail(Stream input, Stream output)...

        //update the thumbnail URL property for an entry.
        public void UpdateThumbnailURL(string partitionKey, string rowKey, string thumbUrl)...
    }
}

```

To rebuild the project

- In Solution Explorer, right-click **MessageBoard_Data**, and then click **Rebuild**. Make sure the rebuild is successful.

What did I just do?

In this step, you modified the data model.

Next Steps:

You will modify the Web role.

Lesson 2: Modify the Web Role

In this lesson, you modify the markup so that the Web page displays the thumbnails instead of the original images. Each thumbnail is a link to a page displaying the original images. You also modify the code behind file for creating a queue and adding messages to the queue.

Procedures

In this lesson, you will go through the following procedure:

- [Modify the markup](#)
- [Modify the code-behind](#)

To modify the markup

1. In Solution Explorer, expand **MessageBoard_WebRole**, right-click **Default.aspx**, and then click **View Markup**.
2. Find the `<div>` tag with `class="signatureImage"`. Replace the `<div>` tag with the one in the code snippet **Tutorial04-Lesson02-Task01_WebRoleDisplayImage**:

```
<div class="signatureImage">  
<a href="<%=# Eval("ImageUrl") %>" target="_blank">  
"  
      alt="<%=# Eval("GolferName") %>" /></a>  
</div>
```

The `<div>` tag looks like the following after the modification:



```
<div class="signature">  
  <div class="signatureImage">  
    <a href="<%=# Eval("ImageUrl") %>" target="_blank">  
      "  
        alt="<%=# Eval("GolferName") %>" />  
    </a>  
  </div>  
  <div class="signatureDescription">  
    <div class="signatureName">  
      <%=# Eval("GolferName") %>  
    </div>  
    <div class="signatureSays">  
      says  
    </div>  
    <div class="signatureDate">  
      <%=# ((DateTime)Eval("Timestamp")).ToShortDateString() %>  
    </div>  
    <div class="signatureMessage">  
      "<%=# Eval("GolferMessage") %>"  
    </div>  
  </div>  
</div>
```

3. Press **Ctrl+S** to save the **Default.aspx** file.

The following procedure demonstrates creating a queue and adding a message. A message may be up to 8KB in size and must be in a format that can be included in an XML request with UTF-8 encoding.

To modify the code-behind

1. In Solution Explorer, expand **MessageBoard_WebRole**, right-click **Default.aspx**, and then click **View Code**.
2. Inside the **btnSend_Click()** method, find the line where the variable *entry* is declared and update the line with the following:

```
MessageBoardEntry entry = new MessageBoardEntry() {  
    GolferName = txtName.Text,  
    GolferMessage = txtMessage.Text,  
    ImageURL=blobURI,  
    ThumbnailURL = blobURI};
```

3. Inside the **btnSend_Click()** method, after the “*ds.AddEntry(entry)*” line, add code to queue a message that will process the image:

```
ds.Enqueue(blobURI, entry);
```

4. Press **Ctrl+S** to save the file.
5. In Solution Explorer, right-click **MessageBoard_WebRole**, and then click **Rebuild**.

What did I just do?

In this step, you modified the Web role to communicate with the worker role for generating thumbnails in the background.

Next Steps:

You will add a new worker role to the solution for background processing.

Lesson 3: Add a Worker Role for Background Processing

A worker role runs in the background to provide services or execute time-related tasks like a service process. In this lesson, 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 message board entry from the message and then retrieves the corresponding entity from the Table service. It then fetches the associated image from the Blob service and creates the thumbnail, which is also stored as a blob. Finally, to complete the processing, it updates the URL of the generated thumbnail blob in the message board entry.

Procedures

In this lesson, you will go through the following procedures:

- [Add a new worker role project](#)
- [Modify the WorkerRole class](#)
- [Build the project](#)
- [Add the storage account settings](#)

To add a new worker role project

1. In Solution Explorer, expand **MessageBoard**, right-click **Roles**, point to **Add**, and then click **New Worker Role Project**.
2. In the **Add New Role Project** dialog, type or select the following values, and then click **Add**.

Name	Value
Category	Installed/Windows Azure Cloud Service/.NET Framework 4.X/Worker Role/Visual C#
Template	Worker Role
Name	MessageBoard_WorkerRole

3. In Solution Explorer, right-click the **MessageBoard_WorkerRole** project, and then click **Add Reference**.
4. In the **Reference Manager** dialog, expand **Projects**, click **Projects**, select **MessageBoard_Data**, and then click **OK**.

To modify the WorkerRole class

1. In Solution Explorer, expand **MessageBoard_WorkerRole**, and then double-click **WorkerRole.cs** to open the file.
2. At the top of the file, add the following namespace declarations:

```
using MessageBoard_Data;
```

3. In the beginning of the class, add the following declaration:

```
private MessageBoardDataSource ds;
```

4. Inside the OnStart method and immediately before the return method, add the following code to create the MessageBoardDataSource object. The MessageBoardDataSource constructor creates the queue:

```
ds = new MessageBoardDataSource();
```

5. Replace the body of the Run() method, the code snippet for the new Run() method is **Tutorial04-Lesson03-Task02_WorkerRoleRun**:

```
public override void Run()  
{  
    while (true)  
    {  
        ds.ProcessQueueMessage();  
    }  
}
```

6. Press **Ctrl+S** to save the file.

To build the project

- In Solution Explorer, right-click **MessageBoard_WorkerRole**, and then click **Build**. Make sure the rebuild is successful.

To add the storage account settings

1. In **Solution Explorer**, expand **MessageBoard**, expand **Roles**, and then double-click **MessageBoard_WorkerRole** to open the properties for this role.
2. Switch to the **Settings** tab.
3. Click **Add Setting**.
4. Type or select the following values.

Name	Value
Name	DataConnectionString
Type	Connection String
Value	(click the ellipses button (...), and then select Use the Windows Azure storage emulator)

Note: The Windows Azure storage emulator provides local instances of the Blob, Queue, and Table services that are available in the Windows Azure. For more information, see *Overview of*

Running a Windows Azure Application with the Storage Emulator at <http://msdn.microsoft.com/en-us/library/gg432983.aspx>.

5. Press **Ctrl+S** to save changes to the role configuration.

What did I just do?

In this step, you added a new worker role to the solution for generating thumbnail images from given images in the background.

Next Steps:

You will test and deploy the application.

Lesson 4: Test and Deploy the Application

In this lesson, you test the application in the development fabric environment, package the application, and then deploy the application to Windows Azure.

Note: If you don't have a Windows Azure Platform subscription, see the *Provisioning Windows Azure* section of [Windows Azure and SQL Azure Tutorials](#).

Procedures

In this lesson, you will go through the following procedures:

1. [Test the application](#)
2. [Generate the service package](#)
3. [Sign in to Windows Azure](#)
4. [Create a storage account](#). You need a storage account to utilize Windows Azure storage services, for example the table service.
5. [Create a cloud service](#). The cloud service is used to host the message board application.
6. [To configure the ServiceConfiguration.Cloud.cscfg file](#)
7. [Deploy the application to the staging environment](#)
8. [Test the application](#)
9. [Promote the application to the production environment](#)

To test the application

1. In Solution Explorer, right-click **MessageBoard**, and then click **Set as Startup Project**.
2. From the **Debug** menu, click **Start Debugging**. You will see a compute emulator icon added to the notification area of taskbar, and a new Internet Explorer window showing the Golfer Message Board application.
3. Switch to Internet Explorer to view the message board application.
4. Add a few entries to the message board by entering your name and a message before clicking **Send**.
5. Close the Internet Explorer window.

After the application is tested successfully in the compute emulator environment, the next step is to create the service package and then deploy the application to Windows Azure.

To generate the service package

1. In Solution Explorer, right-click the **MessageBoard** cloud project, and then select **Package**.
2. In the **Package Windows Azure Application** dialog, select the following values:

Name	Value
Service configuration	Cloud

Build configuration	Release
---------------------	---------

- Click **Package**. 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. The default directory is **C:\AzureTutorials\Tutorial4\GolferMessageBoard\MessageBoard\bin\Release\app.publish**. Make sure to write down the path. You will need it in the deployment lesson.

You will get a few warning messages about 'DataConnectionString' set up to use the local storage emulator. You can ignore these warning for now.

For deploying the golfer message board application, you must have a *storage account* for accessing the Windows Azure storage services, and a *cloud service*, which is a container for service deployments in Windows Azure. For better performance, you might want to create an *affinity group* to group the service and the storage accounts within a subscription according to geo-location.

To sign in to Windows Azure

- Open a Web browser and browse to <http://windows.azure.com/>. This is the Windows Azure Management Portal.
- Sign in using the Windows Live ID associated with your Windows Azure account.

Note: If you haven't had a Windows Azure Platform subscription, see the [Provisioning Windows Azure](#) section of this tutorial.

The next step is to create a storage account for using Windows Azure table service and blob service. If you have created a storage account in [tutorial 1](#) or [tutorial 2](#), you can skip this step.

To create a storage account for the golfer message board application to store its data

- From the portal, in the left pane, click **STORAGE**.
- On the bottom left corner, click **NEW**, click **STORAGE**, and then click **QUICK CREATE**.
- Type or select the following values.

Name	Value
URL	<yourname>gmb (i.e. johndolegmb)
REGION/AFFINITY GROUP	(Choose the region where you wish your service to run, most likely, the one that is closest to your current location. Select an affinity group instead of a region if you want your storage services to be in the same data center with the cloud service that you will create in the next step. To create an affinity group, open the NETWORKS area of the Management Portal, click AFFINITY GROUPS, and then click CREATE.)

4. Click the check mark on the bottom right corner. Wait until the status of the storage account changes to **Online**. The process can take several minutes.
5. Once the storage account is created, click the storage account from the storage account list to select it, and then click **MANAGE KEYS** on the bottom of the page.
6. Record the **STORAGE ACCOUNT NAME** and **PRIMARY ACCESS KEY**. Later in the tutorial, you will need to configure your cloud service to use storage account by specifying the account name and the access key.
7. Close the dialog.

If you have published other tutorial projects before, you can either update the existing cloud service, or create a new cloud service. If you choose to create a new cloud service, you must give a different cloud service URL.

To create a cloud service

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. On the bottom left corner of the portal page, click **NEW**, click **COMPUTE**, click **CLOUD SERVICE**, and then click **CUSTOM CREATE**.
3. Type or select the following values.

Name	Value
URL	<yourname>gmb Note: The URL prefix must be unique.
REGION/AFFINITY GROUP	(For better performance, select the same region as the one you chose for the storage service, or use an affinity group.)
Deployment a cloud service package now	(not selected)

4. Click the check sign on the bottom right corner. Wait until the status changes to **Created**. This process could take several minutes.

When you create and test the application locally, the application is configured to use the development storage. Now you have created a storage account, you can configure the application to use the storage account before deploying the application to Windows Azure. The configuration information is in the ServiceConfiguration.Cloud.cscfg file. This file was created when you generated the service package.

To configure the ServiceConfiguration.Cloud.cscfg file

1. Use Notepad to open **C:\AzureTutorials\Tutorial4\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg**. This path can be different if you installed the tutorial files into

a different folder than the c root directory. Notice the value of DataConnectionString is "UseDevelopmentStorage=true".

2. Change the value of **DataConnectionString** to **DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your storage account primary access key>**. Replace <your storage account name> and <your storage account access key> accordingly.
3. Change the value of **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString** to **DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your storage account primary access key>**. You must replace <your storage account name> and <your storage account primary access key> with the actual account name and account key for the storage account that you created earlier in the tutorial.
4. Optionally, you can increase the Instances count to scale out the application.
5. Save the file and close Notepad.

To deploy the application to the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you just created. The default name is <yourname>gmb..
3. From the top of the page, click **STAGING**.
4. Click **UPLOAD A NEW STAGING DEPLOYMENT**, or **UPDATE** on the bottom of the page if you have deployed a tutorial application before.
5. From **Upload a package**, type or select the following value.

Name	Value
Deployment name	v1.3.0.0
Package location	C:\AzureTutorials\Tutorial3\GolferMessageBoard\MessageBoard\bin\Release\app.publish\MessageBoard.cspkg
Configuration file	C:\AzureTutorials\Tutorial3\GolferMessageBoard\MessageBoard\bin\Release\app.publish\ServiceConfiguration.Cloud.cscfg
Deploy(or Update) even if one or more roles contain a single instance	(Selected). You can always increase the number of instances from the portal.
Start deployment (this option is not available for updating a	(Selected)

service)	
Allow update if role sizes or number of roles change. (this option is only available for updating a service)	(Selected)
ROLE(this option is only available for updating a service)	All

6. Click the check sign on the bottom right corner of the page.
7. Wait until the upload process is completed. The process can take several minutes to complete. Notice the DEPLOYMENT NAME is changed to v1.3.0.0.

To test the application in the staging environment

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the right side of the page, click the site URL.
5. Test the application by entering one or more entries.

After the application is working correctly in the staging environment, you are ready to promote it to the production environment.

To promote the application to production

1. From the portal, in the left pane, click **CLOUD SERVICES**.
2. In the middle pane, click the cloud service you created.
3. From the top of the page, click **STAGING**.
4. On the bottom of the page, click **SWAP**.
5. On the right, click **YES**.
6. On the top of the page, click **PRODUCTION**. It takes several minutes to complete the operation.
7. On the right, click the SITE URL.
8. In the **Properties** pane, click the URL in the **DNS name** box. The application is opened in a new browser tab or a new browser window depending on your browser configuration.

Note: Some DNS services take longer to replicate the records. If you get a page not found error, you might need to try browsing to the URL again in a few minutes.

9. Test the application in the production environment by entering one or more entries.

What did I just do?

In this step, you tested and deployed the golfer message board to Windows Azure.

Next Steps:

Congratulations! You have completed the first four Windows Azure and SQL Database tutorials.