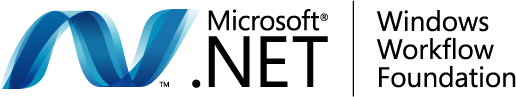


Hands-On Lab

Using Workflow on Windows Azure

Lab version: 1.0.0

Last updated: 12/20/2011



Contents

[Overview 3](#_Toc310421647)

[Starting Materials 4](#_Toc310421648)

[Exercise 1: Building Your First Workflow Service on Windows Azure 4](#_Toc310421649)

[Task 1 – Creating the Visual Studio Project 4](#_Toc310421650)

[Task 2 – Add a Workflow Service 8](#_Toc310421651)

[Task 3 – Add the Model, View and Controller 9](#_Toc310421652)

[Task 4 – Add a Service Reference for MyService.xamlx 12](#_Toc310421653)

[Task 5 – Add code to invoke the service 14](#_Toc310421654)

[Task 6 – (Optional) Enable Workflow Tracking Output in Visual Studio 14](#_Toc310421655)

[Exercise 1 Verification 16](#_Toc310421656)

[Summary 18](#_Toc310421657)

[Task ? (Optional) – Add configuration settings for your Azure account 19](#_Toc310421658)

[Task 1 - [Task Name] 20](#_Toc310421659)

[Appendix 20](#_Toc310421660)

[[Appendix Item 1] 20](#_Toc310421661)

Overview

* 1. In this lab, you will learn how you can use Windows Workflow Foundation on Windows Azure to create a Workflow Service.

# Objectives

* + Learn about options for using Workflow in Azure Web and Worker Roles
  + Learn how to setup Persistence with SQLAzure
  + Discover options for monitoring workflows in Azure

# Prerequisites

* 1. The following is required to complete this hands-on lab:
  + [Microsoft Visual Studio 2010](http://msdn.microsoft.com/vstudio/products/)
  + Microsoft .NET Framework 4.0
  + [Windows Azure Tools for Microsoft Visual Studio 1.5](http://www.microsoft.com/windowsazure/sdk/)
  + [SQL Server 2005 Express Edition (or later)](http://www.microsoft.com/express/sql/download/)

# Exercises

* 1. This Hands-On Lab comprises the following exercises:
  2. Building Your First Workflow Service on Windows Azure
  3. Exercise 2: Publishing your first Workflow Service to Windows Azure
  4. Estimated time to complete this lab: **[120] minutes**.

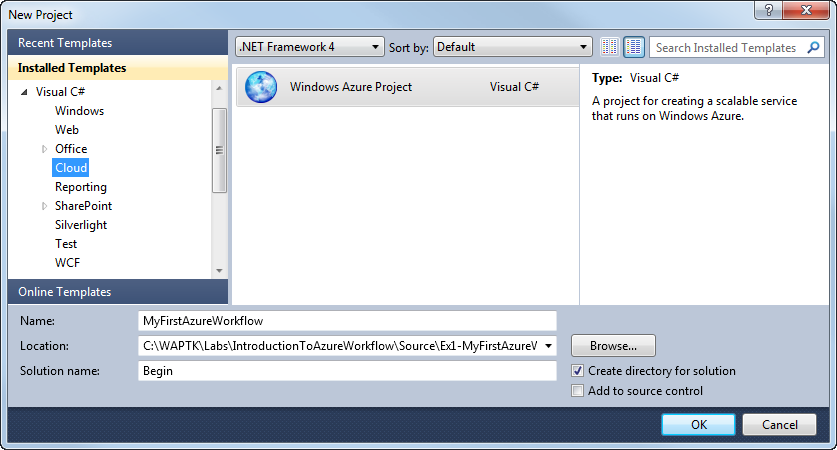
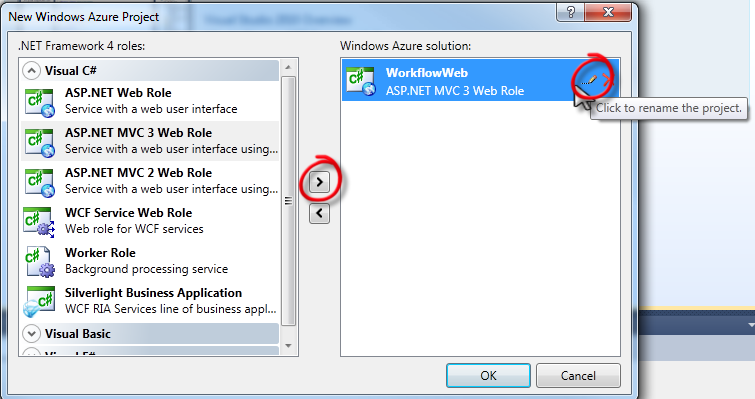
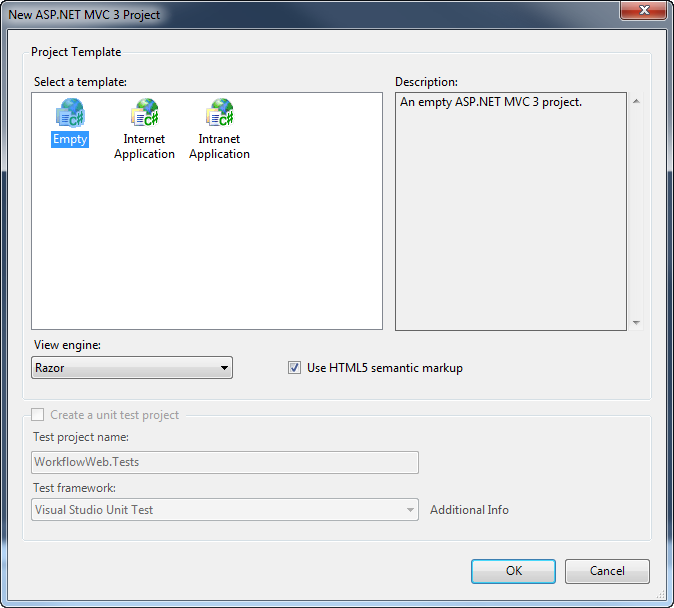
### Starting Materials

* 1. This Hands-On Lab includes the following starting materials.
  + **Visual Studio solutions.** The lab provides the following Visual Studio solutions that you can use as starting point for the exercises.
    - **[Exercise 1 folder]\begin.sln**: [write description for the solution here].
    - **[Exercise 2 folder]\begin.sln**: [write description for the solution here].
    1. **Note:** Inside each exercise folder, you will find an **end** folder containing a solution with the completed lab exercise.

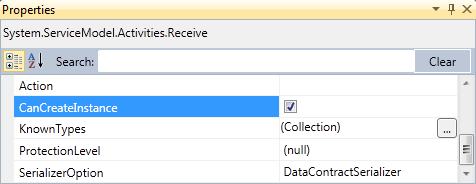
Exercise 1: Building Your First Workflow Service on Windows Azure

Workflow Services are workflows that are implemented as Web Services using Windows Communication Foundation. In this exercise you will learn how to create and use a Workflow Service in an ASP.NET MVC3 application in both the development environment and Windows Azure.

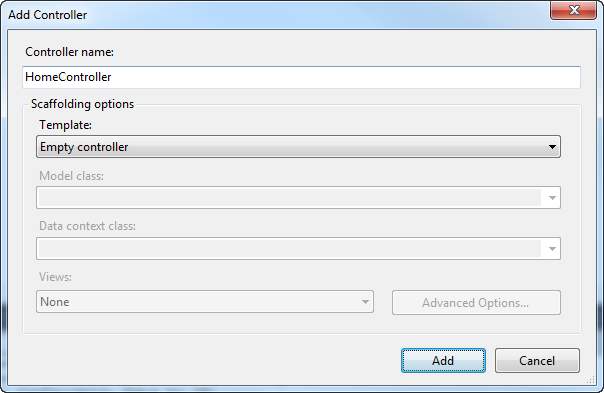
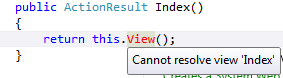
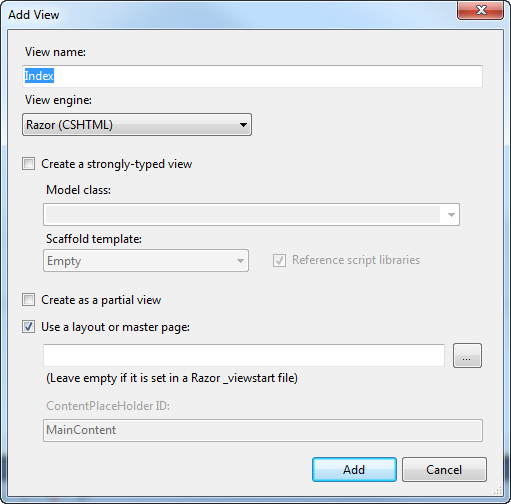
Task 1 – Creating the Visual Studio Project

* 1. In this task, you create a new Cloud Service project in Visual Studio.
  2. Open Visual Studioas administrator from **Start | All Programs | Microsoft Visual Studio 2010** by right clicking the **Microsoft Visual Studio 2010** shortcut and choosing **Run as administrator**.
  3. If the **User Account Control** dialog appears, click **Yes**.
  4. From the **File** menu, choose **New** and then **Project**.
  5. In the **New Project** dialog, expand the language of your preference (Visual C# or Visual Basic) in the **Installed Templates** list and select **Cloud**. Choose the **Windows Azure Project** template, set the **Name** of the project to **MyFirstAzureWorkflow**, set the location to **\Source\Ex1- MyFirstAzureWorkflow \[CS|VB]**, change the solution name to **Begin**, and ensure that **Create directory for solution** is checked. Click **OK** to create the project.
     1. 
     2. Figure 1
     3. Creating a new Windows Azure Cloud Service project (C#)
     4. **Note:** Windows Azure supports the .NET Framework 4.0. If you use Visual Studio 2010 to create the project, you can select this version for the target framework and take advantage of its new features.
  6. In the **New Windows Azure Project** dialog, inside the **Roles** panel, expand the tab for the language of your choice (Visual C# or Visual Basic), select **ASP.NET MVC3 Web Role** from the list of available roles and click the arrow (>) to add an instance of this role to the solution. Before closing the dialog, select the new role in the right panel, click the pencil icon and rename the role as **WorkflowWeb**. Click **OK** to create the cloud service solution.
     1. 
     2. Figure 3
     3. Assigning roles to the cloud service project (C#)
  7. In the New ASP.NET MVC 3 Project dialog select the **Empty** template and click OK.
     1. 
     2. Figure 4
     3. Adding a new Empty ASP.NET MVC3 project

Task 2 – Add a Workflow Service

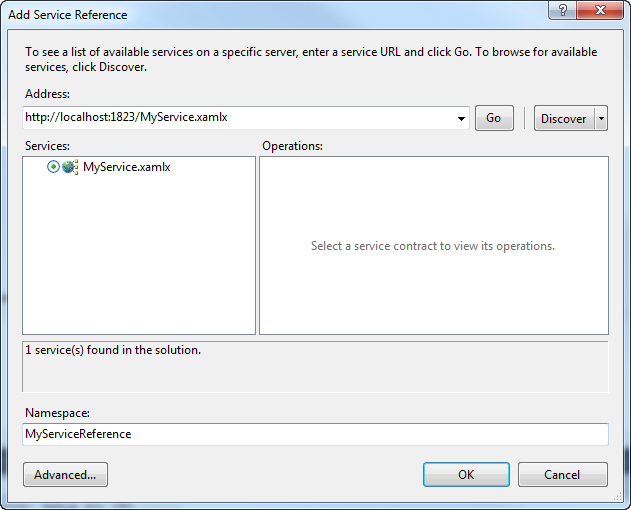
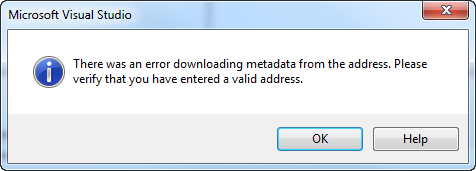
* 1. Right click on the **WorkflowWeb** project and select **Add / New Item…**
  2. From the **Workflow** templates choose **WCF Workflow Service** and name the service **MyService.xamlx**. The Workflow Service will open in the Workflow designer.
  3. Click on the **ReceiveRequest** activity and in the properties windows check the **CanCreateInstance** checkbox.
     1. 
     2. Figure 5
     3. Checking the CanCreateInstance checkbox in the properties window
  4. Open the **WorkflowWeb\web.config** and add a new **system.ServiceModel** section to enable service metadata.
     1. XML
     2. <system.serviceModel>
     3. <behaviors>
     4. <serviceBehaviors>
     5. <behavior>
     6. <serviceMetadata httpGetEnabled="true"/>
     7. </behavior>
     8. </serviceBehaviors>
     9. </behaviors>
     10. </system.serviceModel>

Task 3 – Add the Model, View and Controller

* 1. Right click on the **Models** folder and select **Add / Class**
  2. Set the name to **GetDataModel**
  3. Add the following code
     1. C#
     2. namespace WorkflowWeb.Models
     3. {
     4. using System.ComponentModel.DataAnnotations;
     6. public class GetDataModel
     7. {
     8. [Required]
     9. public int Data { get; set; }
     10. }
     11. }
  4. Right click on the **Controllers** folder and select **Add / Controller…**
  5. Name the controller **HomeController**
     1. 
     2. Figure 6
     3. Adding an Empty Controller
  6. Add the following namespace directive to the controller
     1. C#
     2. using WorkflowWeb.Models;
  7. Add a method to handle the form post
     1. C#
     2. [HttpPost]
     3. public ActionResult Index(GetDataModel model)
     4. {
     5. return this.View();
     6. }
  8. In the controller code, right click on the View() and select Add View…
     1. 
     2. Figure 7
     3. Visual Studio will display the view in red until you add a view
  9. Name the view **Index** and click **Add**
     1. 
     2. Figure 8
     3. Adding the view
  10. Add the following code to the view
      1. C#

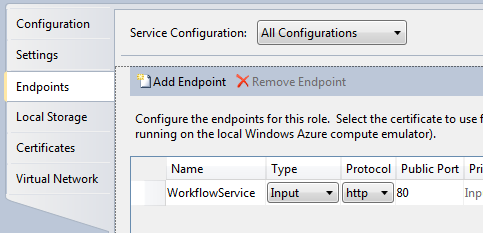
1. @model WorkflowWeb.Models.GetDataModel
3. @{
4. ViewBag.Title = "My First Azure Workflow Service";
5. }
7. <h2>My First Azure Workflow Service</h2>
8. @Html.ValidationSummary("Request data was invalid. Please correct the errors and try again.")
10. @using (Html.BeginForm()) {
11. <div>
12. <fieldset>
13. <legend>GetData Request</legend>
15. <div class="editor-label">
16. @Html.LabelFor(m => m.Data)
17. </div>
18. <div class="editor-field">
19. @Html.TextBoxFor(m => m.Data)
20. @Html.ValidationMessageFor(m => m.Data)
21. </div>
22. <p>
23. <input type="submit" value="Submit" />
24. </p>
25. </fieldset>
26. @if (ViewBag.Response != null)
27. {
28. <p>Response is <b>&quot;@ViewBag.Response&quot;</b></p>
29. }
30. </div>
31. }

Task 4 – Add a Service Reference for MyService.xamlx

* 1. Right click on **WorkflowWeb** and select **Add Service Reference…**
  2. Click the **Discover** button
  3. Set the namespace to **MyServiceReference** and click OK
     1. 
     2. Figure 9
     3. Adding a service reference to MyService
     4. **Note:** If you get an error, make sure you have enabled metadata in the **WorkflowWeb\web.config** file
     5. 
     6. **!HACK!**
     7. If you get an error Could not load file or assembly 'msshrtmi' or one of its dependencies
     8. edit the project file and remove the <PlatformTarget>AnyCPU</PlatformTarget>
     9. Delete msshrtmi.dll from the bin folder

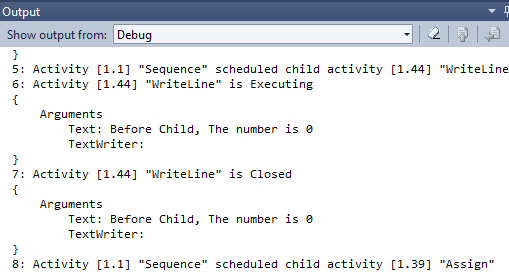
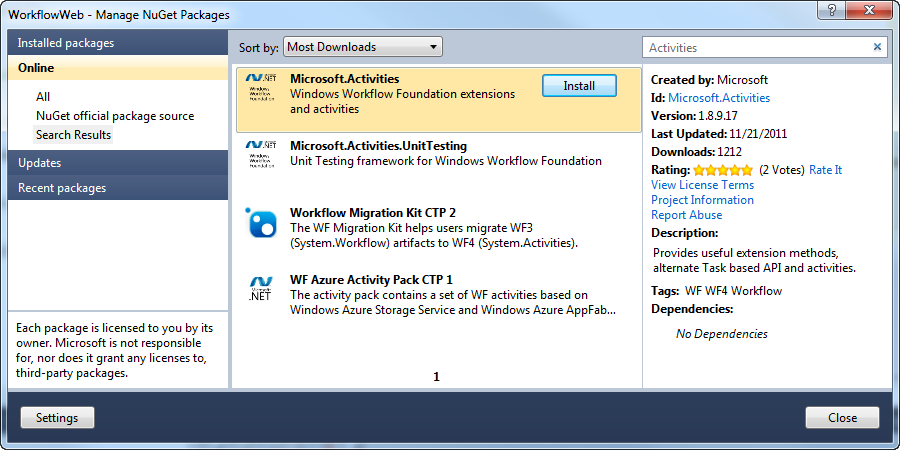


Task 5 – Add code to invoke the service

* 1. Right click on **Roles/WorkflowWeb** and select **Properties**
  2. On the Endpoints tab rename the endpoint to **WorkflowService**
     1. 
     2. Figure 10
     3. Naming the WorkflowService endpoint
  3. Open **HomeController.cs**
  4. Add the following namespace directives
  5. Modify the **[HttpPost] Index** method as shown
     1. C#
     2. [HttpPost]
     3. public ActionResult Index(GetDataModel model)
     4. {
     5. var endpoint = RoleEnvironment.CurrentRoleInstance.
     6. InstanceEndpoints["WorkflowService"].IPEndpoint;
     7. var uri = new Uri(string.Format(
     8. "http://{0}:{1}/MyService.xamlx",
     9. endpoint.Address,
     10. endpoint.Port));
     11. var proxy = new ServiceClient(
     12. new BasicHttpBinding(),
     13. new EndpointAddress(uri));
     15. try
     16. {
     17. this.ViewBag.Response = proxy.GetData(model.Data);
     18. proxy.Close();
     19. }
     20. catch
     21. {
     22. proxy.Abort();
     23. throw;
     24. }
     26. return this.View();

}

Task 6 – (Optional) Enable Workflow Tracking Output in Visual Studio

* + 1. If you want to see workflow trace information in the Visual Studio Output window do the following.
    2. 
    3. Figure 11
    4. Workflow tracking output displayed in Visual Studio
  1. Right click on **WorkflowWeb** and select **Manage NuGet Packages…**
  2. In the search box type **Activities** and press **enter**.
  3. Select **Microsoft.Activities** and click on **Install**
     1. 
     2. Figure 12
     3. Installing Microsoft.Activities
  4. Open **WorkflowWeb\web.config** and add the **extensions** element to the **system.serviceModel** section.
     1. XML
     2. <extensions>
     3. <behaviorExtensions>
     4. <add name="workflowServiceTrace" type="Microsoft.Activities.Diagnostics.WorkflowServiceTraceElement, Microsoft.Activities" />
     5. </behaviorExtensions>
     6. </extensions>
  5. Add the **workflowServiceTrace** behavior to the **serviceBehaviors**
     1. XML
     2. <behaviors>
     3. <serviceBehaviors>
     4. <behavior>
     5. <serviceMetadata httpGetEnabled="true"/>
     6. **<workflowServiceTrace />**
     7. </behavior>
     8. </serviceBehaviors>
     9. </behaviors>

Task 7 – Enable Azure Diagnostics

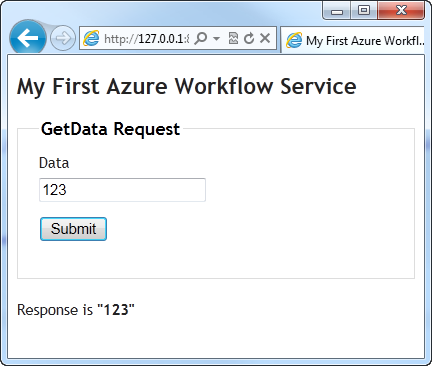
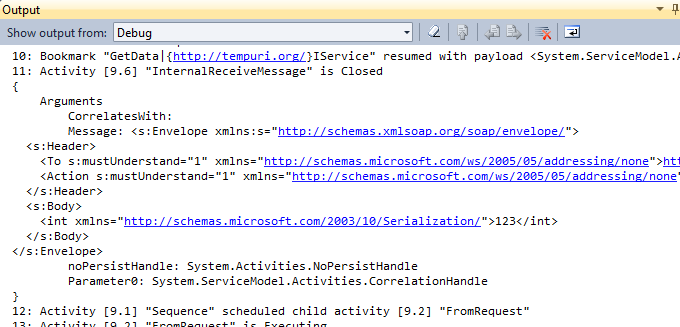
* 1. Open **WebRole.cs**
  2. Modify the **OnStart** method to enable Azure Diagnostics with log transfers every 10 seconds.
     1. C#
     2. public override bool OnStart()
     3. {
     4. CloudStorageAccount.SetConfigurationSettingPublisher(
     5. (configName, configSetter) =>
     6. configSetter(
     7. RoleEnvironment.GetConfigurationSettingValue(configName)));
     8. var diagnosticMonitorConfiguration =
     9. DiagnosticMonitor.GetDefaultInitialConfiguration();
     10. diagnosticMonitorConfiguration.Logs.ScheduledTransferPeriod =
     11. TimeSpan.FromSeconds(10);
     12. diagnosticMonitorConfiguration.Logs.ScheduledTransferLogLevelFilter
     13. = LogLevel.Verbose;
     14. DiagnosticMonitor.Start(
     15. "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString",
     16. diagnosticMonitorConfiguration);
     17. return base.OnStart();
     18. }
  3. Open **WebRole\web.config** and add the following to the system.webServer section to enable tracing of failed requests.
     1. XML
     2. <tracing>
     3. <traceFailedRequests>
     4. <add path="\*">
     5. <traceAreas>
     6. <add provider="ASP" verbosity="Verbose" />
     7. <add provider="ASPNET"
     8. areas="Infrastructure,Module,Page,AppServices"
     9. verbosity="Verbose" />
     10. <add provider="ISAPI Extension" verbosity="Verbose" />
     11. <add provider="WWW Server"
     12. areas="Authentication,Security,Filter,StaticFile,CGI,Compression,Cache,RequestNotifications,Module"
     13. verbosity="Verbose" />
     14. </traceAreas>
     15. <failureDefinitions timeTaken="00:00:15" statusCodes="400-599" />
     16. </add>
     17. </traceFailedRequests>
     18. </tracing>

### Exercise 1 Verification

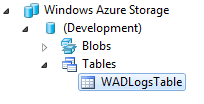
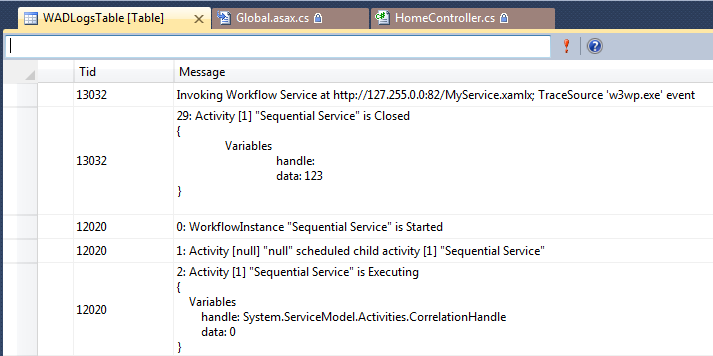
* 1. In order to verify that you have correctly performed all steps of exercise 1, proceed as follows:

#### Verification 1

In this verification, you will invoke the workflow service in the debugger using the Windows Azure Compute Emulator

* 1. From the menu select **Debug / Start Debugging**
  2. When the web page opens enter the value **123** and click submit
  3. The response should be **“123”**
     1. 
     2. Figure 13
     3. The completed application running in the Windows Azure Compute Emulator
  4. If you enabled tracking you should see tracking records in the Visual Studio Debug window
     1. 
     2. Figure 14
     3. Tracking records including message content

#### Verification 2

* 1. In this verification, you will view the log records from table storage to verify the workflow executed properly.
  2. In Visual Studio select **View / Server Explorer**
  3. Right click on **Windows Azure Storage** and select **Refresh**
  4. Expand **Windows Azure Storage** / **(Development)** / **Tables**
     1. **Note:** If there are no Tables yet, wait a minute and try again.
     2. 
     3. Figure 15
     4. Expand the Tables to view the logs
  5. Right click on the **WADLogsTable** and select **View Table**
  6. Expand the Message column to view messages from the HomeController and the Workflow Tracking records
     1. 
     2. Figure 16
     3. The message column with trace messages

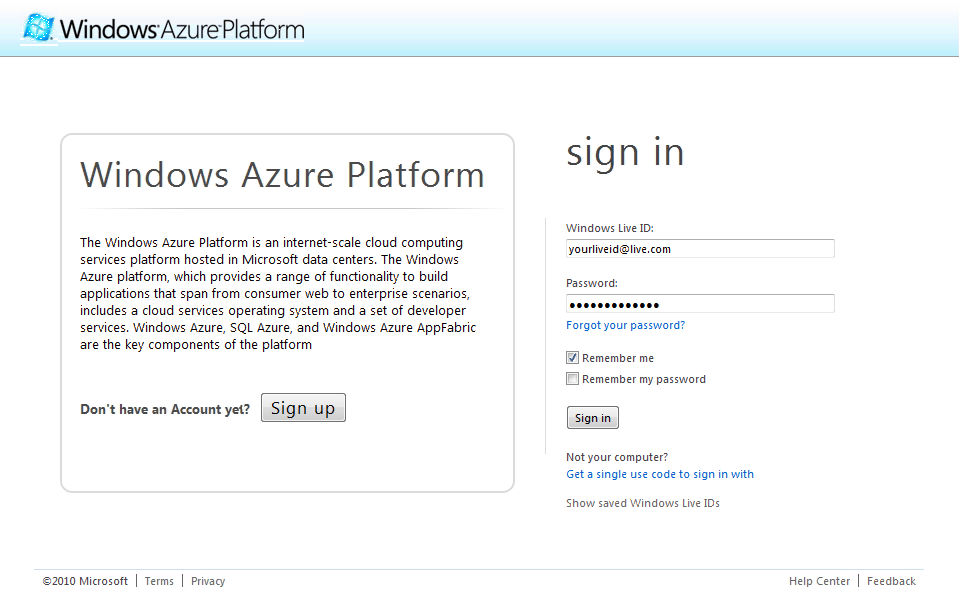
Exercise 2: Publishing your first Workflow Service to Windows Azure

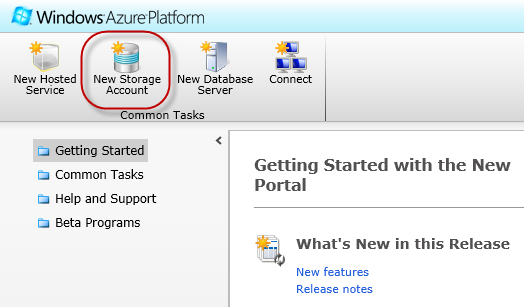
* 1. In this exercise, you will modify the application to run on Windows Azure.

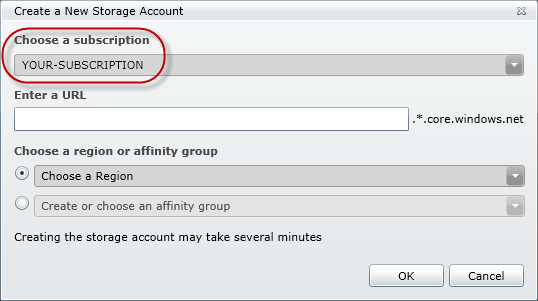
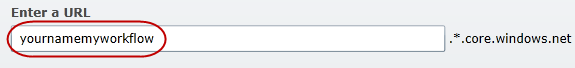
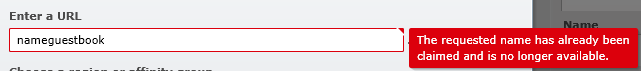
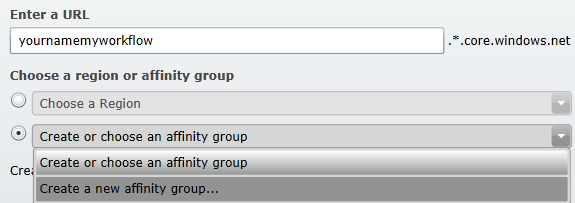
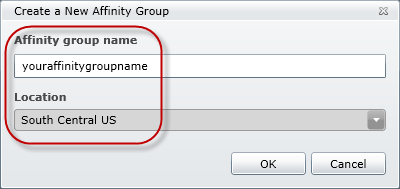
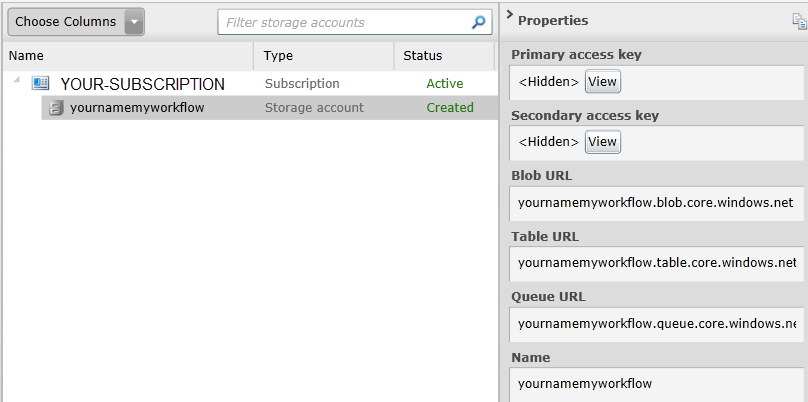
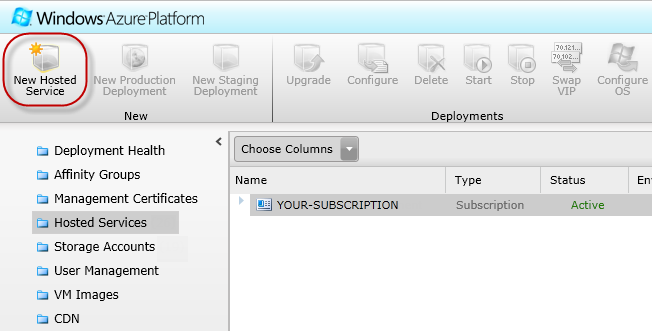
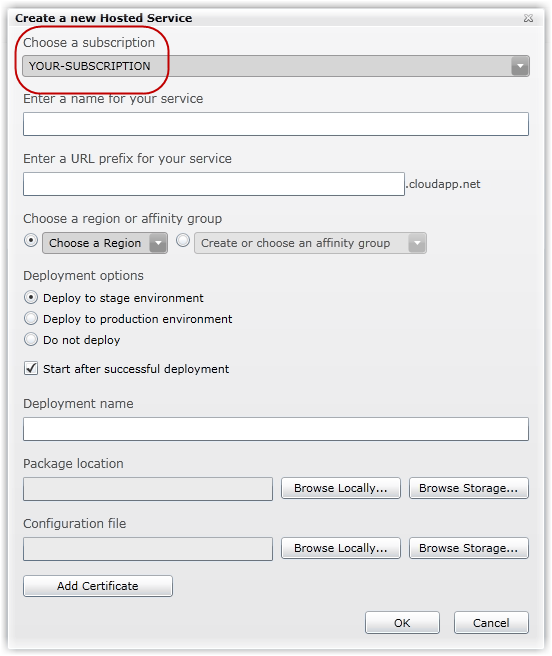
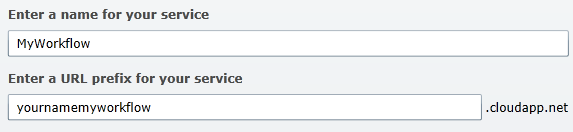
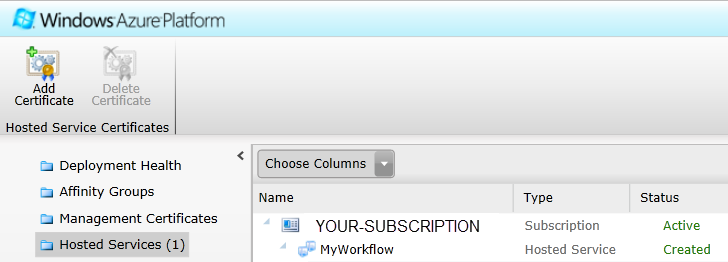
Task 1 – Preparing your app for Windows Azure

* 1. In the **Global.asax** file of the web role, declare the following namespaces.
     1. C#
     2. **using Microsoft.WindowsAzure;**
     3. **using Microsoft.WindowsAzure.ServiceRuntime;**
     4. Visual Basic
     5. **Imports Microsoft.WindowsAzure**
     6. **Imports Microsoft.WindowsAzure.ServiceRuntime**
  2. Next, find the **Application\_Start** method and insert the following (highlighted) code at the start of the method to initialize the configuration settings publisher.
     1. C#
     2. void Application\_Start(object sender, EventArgs e)
     3. {
     4. if (RoleEnvironment.IsAvailable)
     5. {
     6. CloudStorageAccount.SetConfigurationSettingPublisher(
     7. (configName, configSetter) => configSetter(  
        RoleEnvironment.GetConfigurationSettingValue(configName)));
     8. }
     9. ...
     10. }

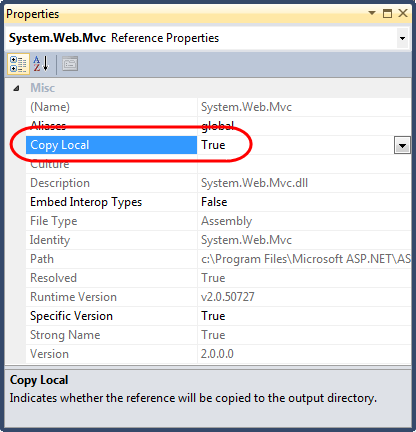
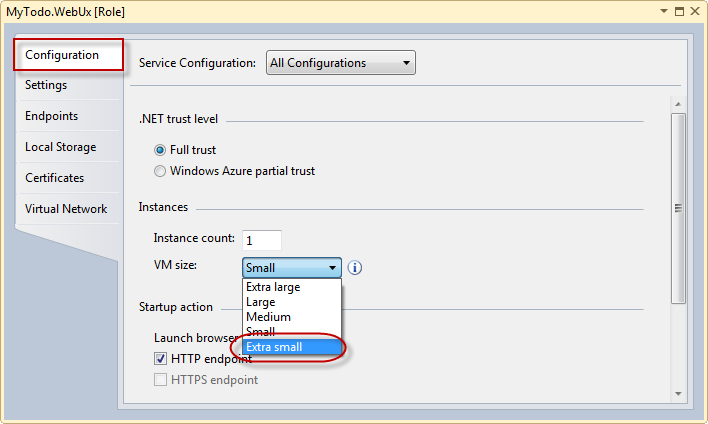
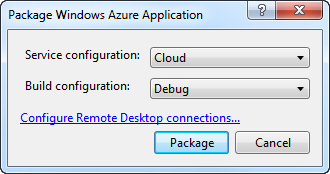
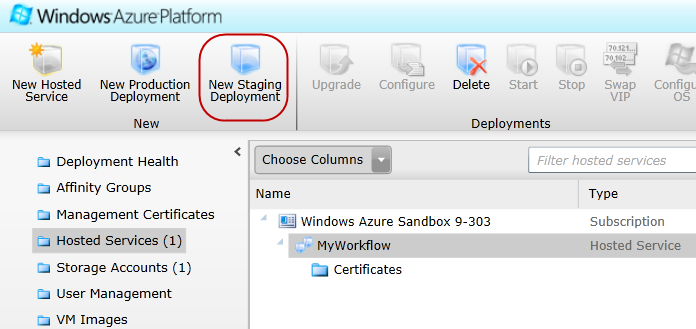
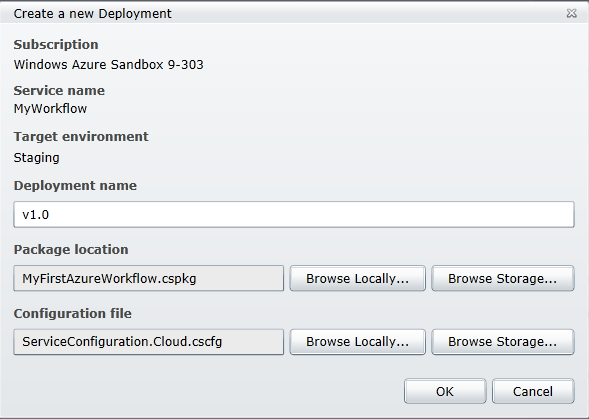
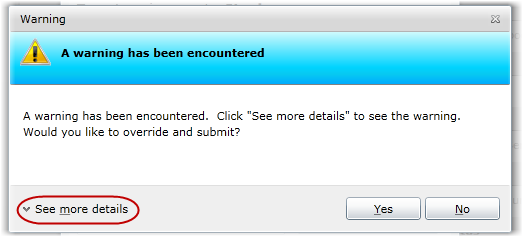
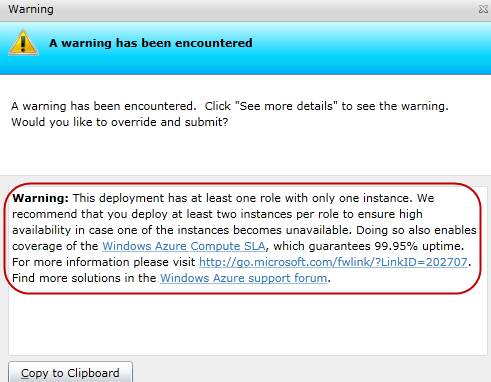
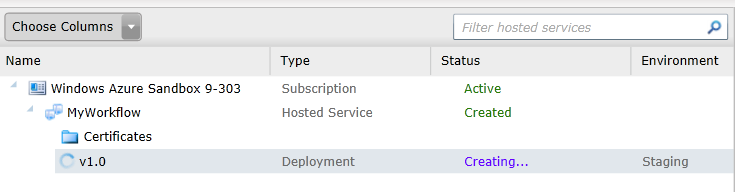
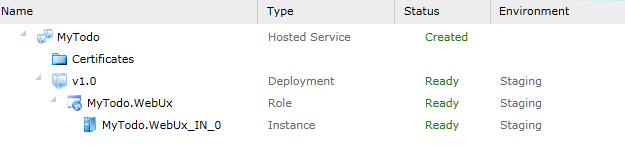
Task 1 – Creating a Storage Account and a Hosted Service Component

* 1. The application you deploy in this exercise requires both compute and storage services. In this task, you create a new Windows Azure storage account to allow the application to persist its data. In addition, you define a hosted service component to execute application code.
  2. Navigate to [http://windows.azure.com](http://windows.azure.com/) using a Web browser and sign in using the Windows Live ID associated with your Windows Azure account.
     1. 
     2. Figure 17
     3. Signing in to the Windows Azure Platform Management portal
  3. First, you create the storage account that the application will use to store its data. In the Windows Azure ribbon, click **New Storage Account**.



* + 1. Figure 18
    2. Creating a new storage account
  1. In the **Create a New Storage Account** dialog, pick your subscription in the drop down list labeled **Choose a subscription**.
     1. 
     2. Figure 19
     3. Choosing a subscription to host the storage account
  2. In the textbox labeled **Enter a URL**, enter the name for your storage account, for example, ***<yourname>myworkflow***, where *<yourname>* is a unique name. Windows Azure uses this value to generate the endpoint URLs for the storage account services.
     1. 
     2. Figure 20
     3. Choosing the URL of the new storage account
     4. **Note:** The name used for the storage account corresponds to a DNS name and is subject to standard DNS naming rules. Moreover, the name is publicly visible and must therefore be unique. The portal ensures that the name is valid by verifying that the name complies with the naming rules and is currently available. A validation error will be shown if you enter a name that does not satisfy the rules.
     5. 
  3. Select the option labeled **Create or choose an affinity group** and then pick **Create a new affinity group** from the drop down list.
     1. 
     2. Figure 21
     3. Creating a new affinity group
     4. **Note:** The reason that you are creating a new affinity group is to deploy both the hosted service and storage account to the same location, thus ensuring high bandwidth and low latency between the application and the data it depends on.
  4. In the **Create a New Affinity Group** dialog, enter an **Affinity Group Name**, select its **Location** in the drop down list, and then click **OK**.
     1. 
     2. Figure 22
     3. Creating a new affinity group
  5. Back in the **Create a New Storage Account** dialog, click **Create** to register your new storage account. Wait until the account provisioning process completes and updates the **Storage Accounts** tree view. Notice that the **Properties** pane shows the **URL** assigned to each service in the storage account. Record the public storage account name—this is the first segment of the URL assigned to your endpoints.
     1. 
     2. Figure 23
     3. Storage account successfully created
  6. Now, click the **View** button next to **Primary access key** in the **Properties** pane. In the **View Storage Access Keys** dialog, click **Copy to Clipboard** next to the **Primary Access Key**. You will use this value later on to configure the application.
     1. 
     2. Figure 24
     3. Retrieving the storage access keys
     4. **Note:** The **Primary Access Key** and **Secondary** Access **Key** both provide a shared secret that you can use to access storage. The secondary key gives the same access as the primary key and is used for backup purposes. You can regenerate each key independently in case either one is compromised.
  7. Next, create the compute component that executes the application code. Click **Hosted Services** on the left pane. Click on **New Hosted Service** button on the ribbon.
     1. 
     2. Figure 25
     3. Creating a new hosted service
  8. In the **Create a new Hosted Service** dialog, select the subscription where you wish to create the service from the drop down list labeled **Choose a subscription**.
     1. 
     2. Figure 26
     3. Choosing your subscription
  9. Enter a service name in the textbox labeled **Enter a name for your service** and choose its URL by entering a prefix in the textbox labeled **Enter a URL prefix for your service**, for example, ***<yourname>myworkflow***, where *<yourname>* is a unique name. Windows Azure uses this value to generate the endpoint URLs for the hosted service.
     1. 
     2. Figure 27
     3. Configuring the hosted service URL and affinity group
     4. **Note:** If possible, choose the same name for both the storage account and hosted service. However, you may need to choose a different name if the one you select is unavailable.
  10. Select the option labeled **Create or choose an affinity group** and then pick the affinity group you defined when you created the storage account from the drop down list.
      1. 
      2. Figure 28
      3. Choosing an affinity group
      4. **Note:** By choosing this affinity group, you ensure that the hosted service is deployed to the same data center as the storage account that you provisioned earlier.
  11. Select the option labeled **Do not Deploy**.
      1. **Note:** While you can create and deploy your service to Windows Azure in a single operation by completing the **Deployment Options** section, for this hands-on lab, you will defer the deployment step until the next task.
  12. Click **OK** to create the hosted service and then wait until the provisioning process completes.
      1. 
      2. Figure 29
      3. Hosted service successfully created
  13. Do not close the browser window. You will use the portal for the next task.

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

* 1. A hosted service is a service that runs your code in the Windows Azure environment. It has two separate deployment slots: staging and production. The staging deployment slot allows you to test your service in the Windows Azure environment before you deploy it to production.
  2. In this task, you create a service package for the MyWorkflow application and then deploy it to the staging environment using the Windows Azure Platform Management Portal.
  3. Ensure that the **System.Web.Mvc** assembly is included in the service package that you deploy to Windows Azure. To do this, for a Visual C# project, expand the **References** node in **Solution Explorer** for the **WorkflowWeb** project, right-click the **System.Web.Mvc** assembly and select **Properties**.   
       
     To add the assembly to the service package, in the **Properties** window for the **System.Web.Mvc** assembly, if **Copy Local** setting is set to *False* then change it to *True*.
     1. 
     2. Figure 15
     3. Including assemblies in the service package deployed to Windows Azure
     4. **Note:** In general, you need to set **Copy Local** = *True* for any assembly that is not installed by default in the Windows Azure VMs to ensure that it is deployed with your application.
  4. Next, change the size of the virtual machine that will host the application. To do this, in **Solution Explorer**, expand the **Roles** node of the **WorkflowWeb** project and then double-click the **WorkflowWeb** role to open its properties window. In the **Configuration** page, locate the **VM Size** setting under the **Instances** category and choose the **Extra small** size from the drop down list.
     1. 
     2. Figure 16
     3. Configuring the size of the virtual machine (VM) for the deployment
     4. **Note:** When you create your service model, you can specify the size of the virtual machine (VM) to which to deploy instances of your role, depending on its resource requirements. The size of the VM determines the number of CPU cores, the memory capacity, the local file system size allocated to a running instance, and the network throughput.
  5. To configure the storage before deploying the service, open **ServiceConfiguration.Cloud.cscfg** file located in **WorkflowWeb** service. Replace the placeholder labeled [YOUR\_ACCOUNT\_NAME] with the **Storage Account Name** that you chose when you configured the storage account in Task 1. If you followed the recommendation, the name should follow the pattern ***<yourname>myworkflow***, where *<yourname>* is a unique name.
  6. Next, replace the placeholder labeled [YOUR\_ACCOUNT\_KEY] with the **Primary Access Key** value that you recorded earlier, when you created the storage account in Task 1.
     1. XML
     2. <ServiceConfiguration serviceName="MyFirstAzureWorkflow" ...>
     3. <Role name="WorkflowWeb">
     4. <Instances count="1" />
     5. <ConfigurationSettings>
     6. <Setting
     7. name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
     8. value="DefaultEndpointsProtocol=https;
     9. AccountName=[YOUR\_ACCOUNT\_NAME];
     10. AccountKey=[YOUR\_ACCOUNT\_KEY]" />
     11. </ConfigurationSettings>
     12. </Role>
     13. </ServiceConfiguration>
     14. Figure 17
     15. Configuring the storage account connection string
  7. Press **CTRL + S** to save the changes to the service model.
  8. To create a service package, right-click the cloud service project and select **Package**.
  9. In the **Package Windows Azure Application** dialog, click **Package** and then wait until Visual Studio creates it. Once the package is ready, Visual Studio opens a window showing the folder that contains the generated files. Copy this path, you will need it later when you deploy the package.
     1. 
     2. Figure 19
     3. Creating a service package in Visual Studio
  10. Go to the Windows Azure Management portal and open the **Hosted Services** page for the project that you created in the previous task.
  11. At the portal, select the hosted service that you created in the previous step and then click **New Staging Deployment** on the ribbon.
      1. 
      2. Figure 20
      3. Hosted service summary page
  12. In the **Create a new Deployment** dialog, to select a **Package location**, click **Browse Locally**, navigate to the folder where Visual Studio generated the package in Step 10 and then select **MyFirstAzureWorkflow.cspkg**.
      1. **Note:** The *.cspkg* file is an archive file that contains the binaries and files required to run a service, in this case, the WorkflowWeb ASP.NET MVC application. Visual Studio creates the service package for you when you select **Build | Publish** for your Windows Azure project.
  13. Now, to choose the **Configuration File**, click **Browse Locally** and select **ServiceConfiguration.Cloud.cscfg** in the same folder that you used in the previous step.
  14. Finally, for the **Deployment name**, enter a label to identify the deployment; for example, use **v1.0**.
      1. 
      2. Figure 21
      3. Configuring service package deployment
  15. Click **OK** to start the deployment. Notice that the portal displays a warning message when you do this. Click **See more details** to review and understand the message.
      1. 
      2. Figure 22
      3. Reviewing the warnings
      4. **Note:** In this particular case, the warning indicates that only a single instance is being deployed for at least one of the roles. This is not recommended because it does not guarantee the service’s availability. In the next task, you will increase the number of instances to overcome this issue.
      5. ****
  16. Click **Yes** to override and submit the deployment request. Notice that the package begins to upload and that the portal shows the status of the deployment to indicate its progress.
      1. 
      2. Figure 23
      3. Uploading a service package to the Windows Azure Platform Management Portal
  17. Wait until the deployment process finishes, which may take several minutes. At this point, you have already uploaded the package and it is in a **Ready** state. Notice that the portal assigned a **DNS name** to the deployment that includes a unique identifier. Shortly, you will access this URL to test the application and determine whether it operates correctly in the Windows Azure environment, but first you need to configure it.
      1. 
      2. Figure 24
      3. Package successfully deployed