* 1. 

Readme

OData Service for Team Foundation Server V2 Beta

Lab version: 2.3.0

* 1. Last updated: 4/17/13

Contents

[Overview 3](#_Toc348500322)

[Configuring and Running the OData Service 6](#_Toc348500323)

[Introduction to OData Service Builds 7](#_Toc348500324)

[Initial OData Service Configuration 9](#_Toc348500325)

[Running the OData Service locally using IIS Express 10](#_Toc348500326)

[Running the OData Service locally using the Windows Azure Emulator 15](#_Toc348500327)

[Deploying the OData Service to IIS Server 17](#_Toc348500328)

[Configuring and Deploying the OData Service to Windows Azure 20](#_Toc348500329)

[Summary 29](#_Toc348500330)

[References and Further Reading 29](#_Toc348500331)

Overview

* 1. The purpose of this sample is to help developers work with data from Team Foundation Server on multiple device types (such as smartphones and tablets) and operating systems. OData provides a great solution for this goal, and has been embraced by numerous developers for building great device-specific applications. OData is accessible from any device and application stack which supports HTTP requests.
  2. The OData service interacts directly with the [TFS client object model](http://msdn.microsoft.com/library/bb130146.aspx), and will work with Team Foundation Service, Team Foundation Server 2010 and 2012, and CodePlex.

For the latest information about the OData Service and FAQ, please see the TechNet wiki page at <http://social.technet.microsoft.com/wiki/contents/articles/15039.odata-service-for-team-foundation-server-v2.aspx>.

Changelog

* 1. Version 2.3.0:
* Fixed bug/configuration issue where incorrect data could be returned when multiple, authenticated requests made identical queries
* Fixed bug where work item queries were not allowed to contain multiple ChangedDate or CreatedDate filter expressions (it is allowed now)
  1. Version 2.2.0 Beta:
  + Can now be configured to work against Team Foundation Service (tfs.visualstudio.com)
  + Work item queries performance improvements (for queries by project collection, project, and query)
  + IterationPath is now available (mirrors AreaPath functionality for now)
  + Simple TFS single-user lookup. Can specify a user name, see updated landing page for details on performing queries. For example, this enables you get the display name for the email address used by end-user when providing basic auth creds
  + All startup tasks only run in Azure environment (no longer the emulator as well)
  + Fixed bug where work item queries for project (and build) loses scope when filters are applied
  + Fixed bugs where work item updates were not working for all properties
  + Fixed bugs where filtering was not working as expected when multiple filters were applied
  + Fixed bug where ‘substringof’ filters were not being interpreted correctly (e.g. ‘substringof(‘asdf’,Title)’ should be interpreted as ‘substringof(‘asdf’,Title) eq true’)
  + Configured default solution to utilize the Azure .wadcfg way of configuring diagnostics
  1. Version 2.1.0 Beta:
  + Fixed bug in RequestWorkItemsByProject where incorrect WIQL was being generated in the case where more than one filter clause was provided. This corresponds to queries such as:
  + /DefaultCollection/Projects(‘prjName’)/WorkItems?$filter=Type eq ‘Bug’ or substringof(‘bug’,State) eq true
  1. Version 2.0.0 Beta:
  + Added WorkItems(X)/Links to API to return links for specific work item
  + Added additional WorkItem fields (Blocked, OriginalEstimate, RemainingWork, StoryPoints, BacklogPriority, BusinessValue, Effort, Size, CompletedWork, AttachedFileCount, HyperLinkCount, RelatedLinkCount)
  + Fixed bug where larger uploads would fail (configuration change)
  + Fixed bug where larger uploads would fail to an HTTPS endpoint (service configuration)
  + Fixed bug where $value command was not returning Attachments
  + Session state turned off to avoid intermittent bug
  + Changed to buffered output from WCF service (not chunked)
  + Improvements in some exception handling and logging

Prerequisites

To build the OData Service, you will need the following:

* + [Visual Studio 2012 with Update 1](http://www.microsoft.com/visualstudio/eng/products/visual-studio-overview)
    1. **Note:** Instructions in this document aren’t based on using an Express version of Visual Studio; therefore, some adjustments may be necessary if you go this route.
  + [Windows Azure SDK for .NET](http://www.windowsazure.com/en-us/develop/downloads/) (optional)
    - Necessary only if building for Windows Azure environment. See the sections specific to IIS Express and IIS Server for details on how to build without installing the SDK.
    - Version 1.8 used at time of release
    1. **Note:** If you want to run or debug using the Azure Emulator, you will also need to install the IIS feature with ASP.NET on your development machine.

To deploy the OData Service, you will need the following:

* + [Microsoft .NET Framework 4.5](http://www.microsoft.com/en-us/download/details.aspx?id=30653)
  + Team Foundation Server 2012 Update 1 Object Model Installer (included as part of ODataTFS.Web project)
  + IIS 7 or later with ASP.NET installed – this means Windows 7 and up, Windows Server 2008 and up
  + SSL certificate to secure HTTPS endpoint (if deploying publically)
  + Windows Azure account (optional)

Third-Party Dependencies

The OData Service uses the [WCF Data Services Toolkit](http://wcfdstoolkit.codeplex.com/) to help with common tasks associated with building an OData endpoint with WCF Data Services. This dependency is located in the *code\References* folder.

The ODataTFS.Tests project, which houses the included unit tests, depends on a mocking library called [Moq](http://code.google.com/p/moq/downloads/list). Version 4.0 of this dependency is included as part of the OData Service download.

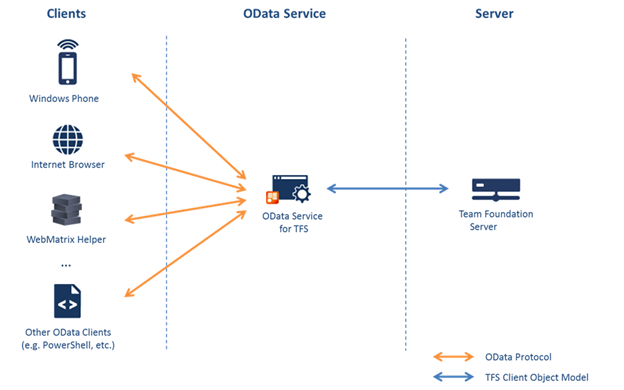
OData

* 1. The Open Data Protocol (OData) exposes a way to work with data over the web. It uses interfaces similar to REST, with powerful query syntax, so that you can programmatically consume and manipulate data from any device or application stack that supports HTTP requests.
  2. To learn more about the OData standard browse to <http://www.odata.org/>. This sample uses a subset of version 2.

Getting Started

* 1. Depending upon your goal for using the OData Service, use the links below to get started:
  + You can see and experiment with a version of the OData Service that has been configured to target Team Foundation Service by browsing to [https://](https://codeplexodata.cloudapp.net)tfsodata.visualstudio.com. This version of the service should be nearly identical in functionality, but be aware that this version may be updated independently when needed.
  + To run the OData Service locally, see the **Configuring and Running the OData Service** section. You can try it out by deploying to IIS Express from within Visual Studio, or you can deploy it to a local instance of IIS.
  + See the [Getting Started video](http://channel9.msdn.com/Blogs/briankel/OData-Service-for-Team-Foundation-Server-2010) on Channel 9.

Solution Description

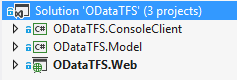
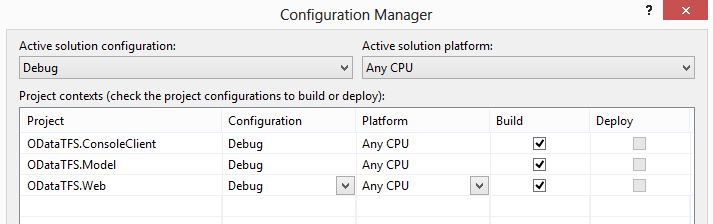
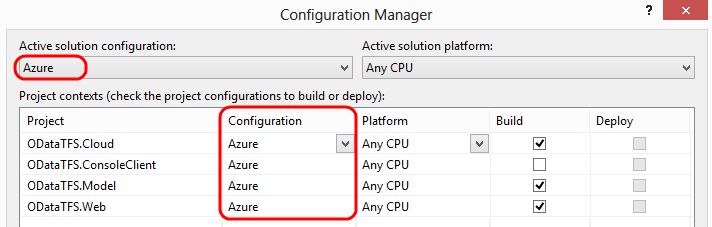
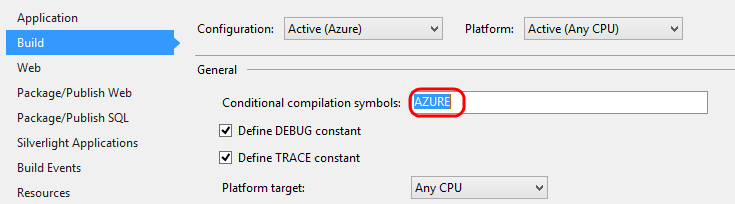
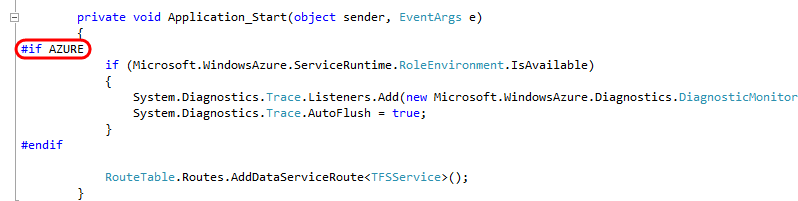
* 1. The main component of this project is the **OData Service for Team Foundation Server 2012**. Different client applications on different platforms (smartphones, tablets, web applications, etc.) can perform OData queries against the service to retrieve information from the configured Team Foundation Server. The service itself responds to the queries by communicating with Team Foundation Server using the Team Foundation Server client object model.
  2. The OData Service is implemented mainly in the **ODataTFS.Model** project, which includes all the necessary code to access the TFS API through the client object model and the classes to parse and execute the OData queries (this implementation is based on the [WCF Data Services Toolkit](http://wcfdstoolkit.codeplex.com/)). Additionally, a Web front-end for the service is provided in the **ODataTFS.Web** project.
  3. The service solution also includes an **ODataTFS.Cloud** project to make it easy to deploy your OData service to Windows Azure (you can also use the local Azure Compute Emulator). Alternatively, you can disregard the **ODataTFS.Cloud** project and deploy the service on a local server (you will find instructions on how to deploy the service on IIS in the Configuring and Running the OData Service section).
  4. 
  5. Figure 1
  6. OData Service

If you look in the Code folder, you will notice three different solution files. Use **ODataTFS.sln** if you plan to build and deploy to IIS, use **ODataTFS.Cloud.sln** if you plan to build and deploy to Windows Azure, and use **ODataTFS.Tests.sln** to work with and execute the provided unit tests. A console application is also included that shows how to reference and consume the OData Service.

Configuring and Running the OData Service

* 1. In this section, we will take a look at the steps necessary to configure and run the OData Service locally and how to build and deploy for IIS and Windows Azure.

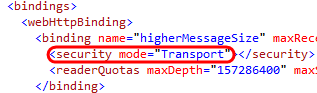
Introduction to OData Service Builds

* 1. Open the **ODataTFS.sln** solution file from the **Code** folder to load it in Visual Studio. The projects and their high-level functionality were described in the overview section.
     1. 
     2. Figure 2
     3. ODataTFS solution
  2. This is the solution that you want to use if you plan to build and deploy to IIS. Let’s take a look at the build configuration (**Build | Configuration Manager…** from the main menu).
     1. 
     2. Figure 3
     3. Configuration Manager
  3. By default, the solution is setup to build all projects using their **debug** configurations and to target the **Any CPU** platform. This may not initially seem to be worth pointing out, but it is important to note that the **ODataTFS.Web** project is designed to run in a normal, on-premises IIS environment as well as an Azure environment with little re-configuration or changes necessary.
  4. Open the **ODataTFS.Cloud.sln** solution file from the **Code** folder and then open the **Configuration Manager** window. Set the **Active Solution Configuration** to be **Azure**.
     1. 
     2. Figure 4
     3. Configuration Manager
  5. There are a couple of important differences to note here. The first is that we now we have an additional project named **ODataTFS.Cloud** that defines a single web role that can be deployed to Windows Azure. In addition, using the **Azure** solution configuration will build the ODataTFS.Web project with a conditional compilation symbol defined that is used to include some additional Azure-specific code. Close the Configuration Manager window without making any changes.
  6. **Right-click** on the **ODataTFS.Web** project in Solution Explorer and select the **Properties** option to view the property pages.
  7. Select the **Build** tab and note that there is a compilation symbol defined here named “**AZURE**”.
     1. 
     2. Figure 5
     3. Conditional compilation symbol for Azure build
  8. This conditional compilation symbol is used in a couple of locations in code. For example, open the **Global.asax.cs** file in the code editor and scroll down to the **Application\_Start** method. If we are building the solution to target Azure, the “if” code block will be compiled. In the case where we are targeting IIS, the Azure runtime will not be installed, so we do not want to attempt to resolve any Microsoft.WindowsAzure.\* assemblies.
     1. 
     2. Figure 6
     3. Conditionally compile code block
  9. Any additional build configuration is up to you and the environment in which you want to deploy. By default, the ODataTFS.Cloud solution will build a debug version of all projects. If you want to build and deploy a release version, that is entirely up to you.

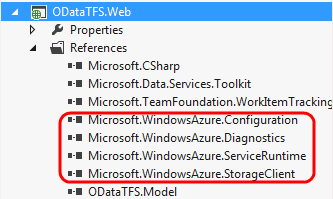
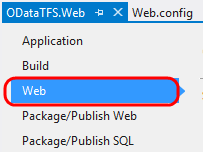
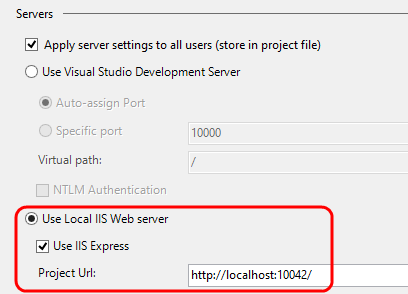
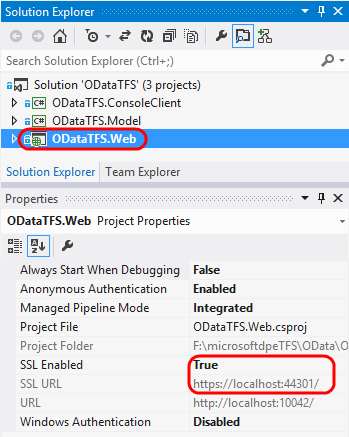
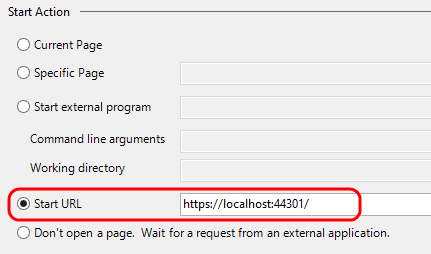
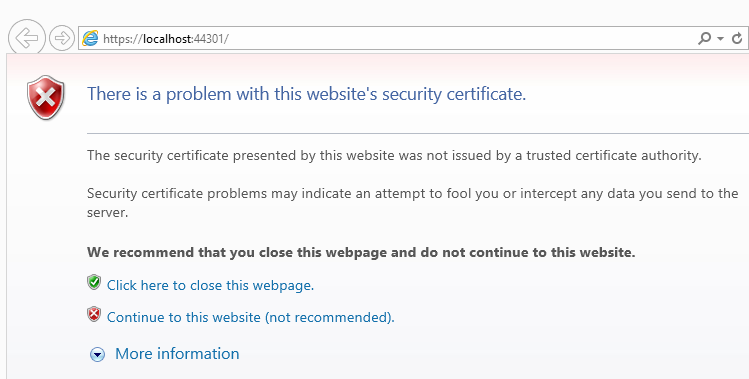
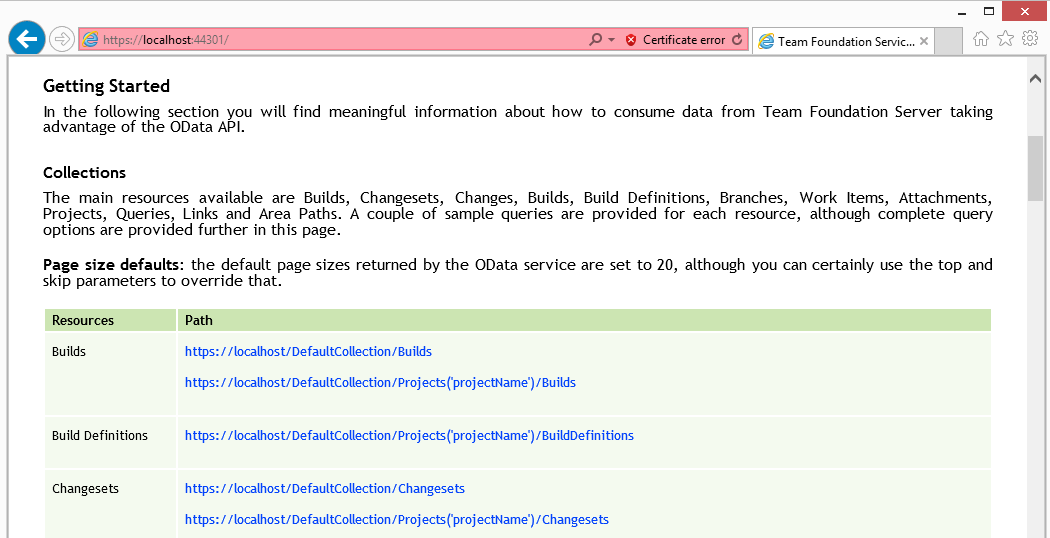
Initial OData Service Configuration

* + 1. There is a certain amount of configuration that you will likely need to do regardless of your desired hosting environment.
  1. Open the **ODataTFS.sln** solution file in Visual Studio.
  2. Open **Web.config** from the **ODataTFS.Web** project and navigate to the “appSettings” configuration section.
  3. The **ODataTFS.TfsServer** app setting value needs to be set to the URL for the TFS server that you wish to expose via OData endpoint. By default, this is set to the URL for the Team Foundation Service endpoint ([https://](https://tfs.codeplex.com/tfs)visualstudio.com). You can use this as-is to connect to Team Foundation Service, or you can change it to any TFS 2010 or TFS 2012 server that the OData Service will have network access to.
     1. 
     2. Figure 7

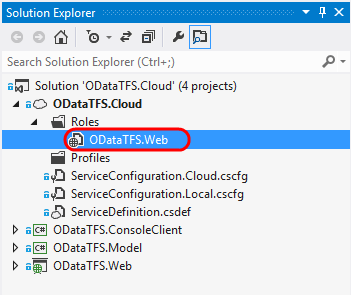
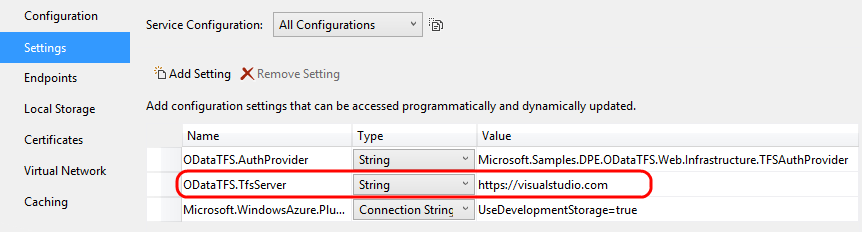
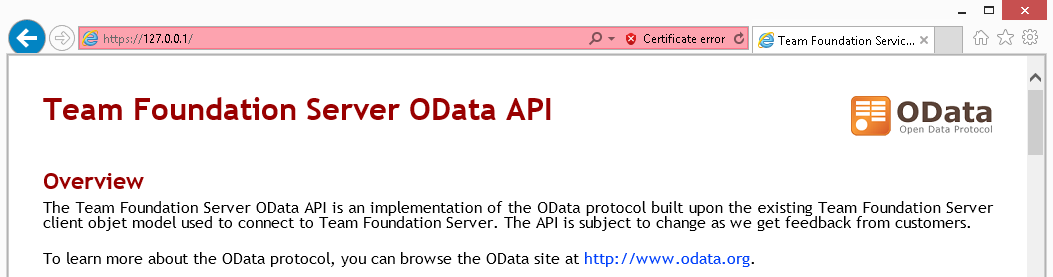
Configuring TFS Server

* 1. The second bit of critical configuration has to do with securing the connection from clients to the OData Service. This is done using SSL, which we will discuss in more detail later, but it is important to note here that the WCF endpoint needs to have its binding configured for this. If you take a look at the binding configuration in **Web.config**, you can see that the security mode is set to “**Transport**”.
     1. 
     2. Figure 8
     3. Security for WCF service
  2. The security mode was left set to Transport as a default because we want anyone trying out this sample to be fully aware of the implications of running without it. OData credentials are sent over the wire using Basic authentication, so securing the transport layer is necessary. If you try to run the service over HTTP, you will get an exception and have to explicitly change the security mode of the binding. **Only use HTTP for testing purposes within a sandboxed environment**.

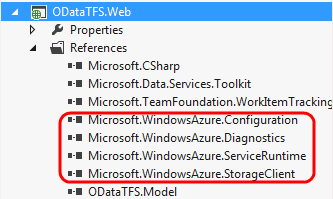
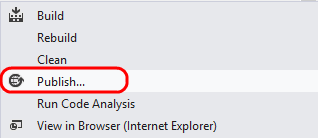
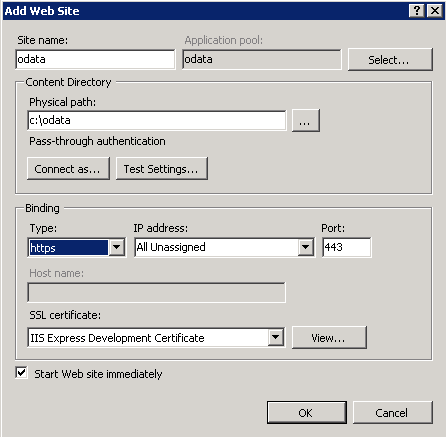
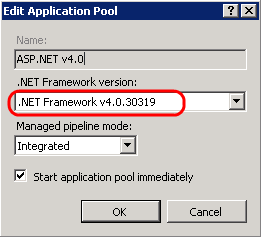
Running the OData Service locally using IIS Express

* + 1. After configuring the OData Service to point at your chosen TFS instance, you can run the service locally using IIS Express. This is most useful during development and debugging.
    2. **Note:** When testing against larger instances, some queries will take unnecessarily long when running in IIE Express as dynamic content compression is not enabled. In this case, it is recommended that you run the site on a normal IIS server (with dynamic content compression enabled for the site).
  1. Open the **ODataTFS.sln** solution file in Visual Studio (if not already open).
  2. If you already have the Azure SDK installed, go ahead and skip to the next section. If you want to avoid installing the Azure SDK and will ultimately be deploying to IIS, you can simply remove the Microsoft.WindowsAzure.\* references from the Web project.
     1. 
     2. Figure 9
     3. Removing unneeded references for on-premises deployment (optional)
  3. The **ODataTFS.Web** project is configured to use **IIS Express** by default when debugging locally. Open the project property pages and select the **Web** tab.
     1. 
     2. Figure 10
     3. Web properties tab for ODataTFS.Web project
  4. Under the **Servers** section, note that the “**Use Local IIS Web server**” and “**Use IIS Express**” options are selected and that a **Project URL** is set.
     1. 
     2. Figure 11
     3. IIS Express configuration
  5. Since we are forcing the WCF service to require SSL, we also need to make sure that IIS Express is configured to use it. The easiest way to do this is to single-click on the **ODataTFS.Web** project node in Solution Explorer and then view the **Properties** window (press **F4** if not visible).
     1. 
     2. Figure 12
     3. Configuring SSL for IIS Express
  6. SSL should already be enabled, but go ahead and do that if it’s not the case. Note that a **SSL URL** is listed here when enabled. Copy the SSL URL to the clipboard with **Ctrl+C**.
  7. Switch back to the **Web** tab of the project property pages and paste the SSL URL as the **Start URL** also (use **Ctrl+V**). This simplifies the process of building and testing the OData Solution.
     1. 
     2. Figure 13
     3. Start URL
  8. Assuming that the default port assignments for IIS Express are not already in use on your system, you should be able to start the service now by pressing **F5**. Once the Internet Explorer window opens, select the option to “**continue to this website**”.
     1. 
     2. Figure 14
     3. Manually bypass the security certificate warning
  9. You should now be looking at the default landing page that provides some information about the OData Service and a summary of the available queries and operations.
     1. 
     2. Figure 15
     3. OData Service landing page

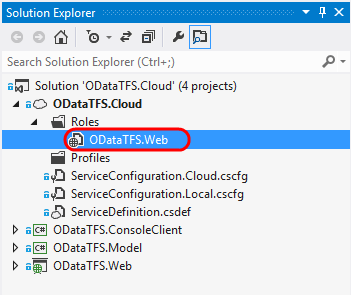
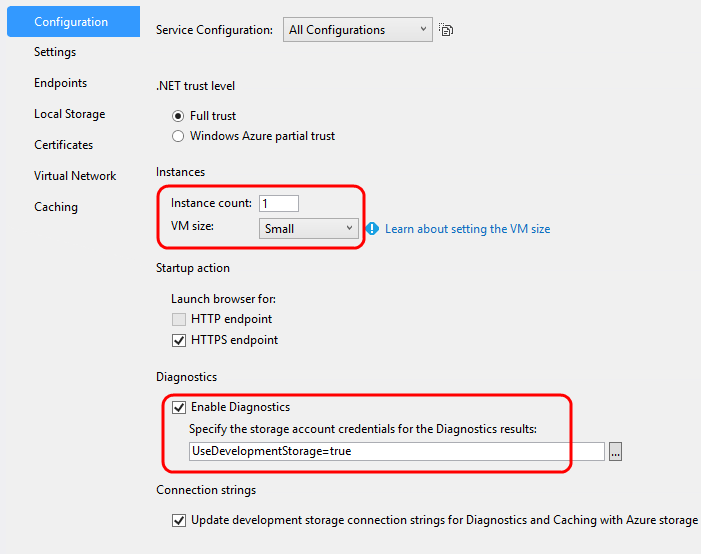
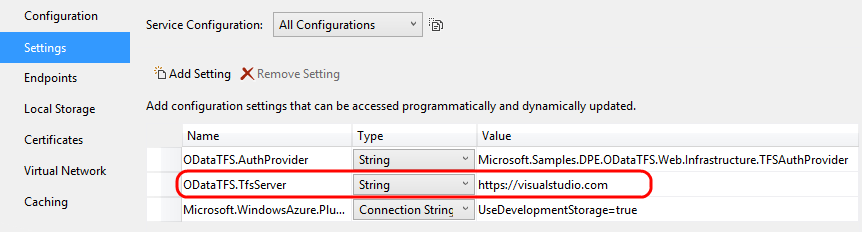
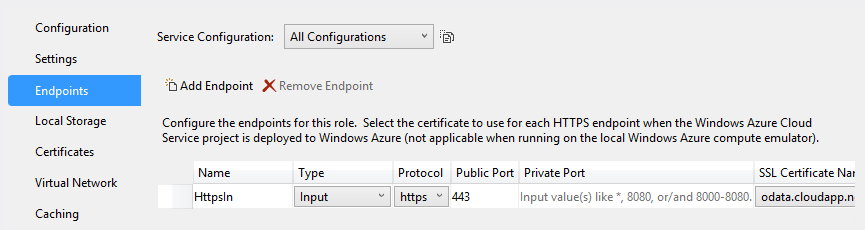
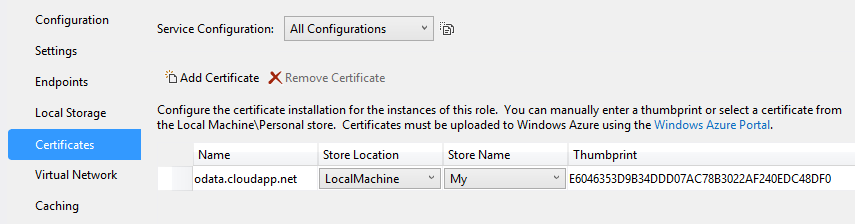
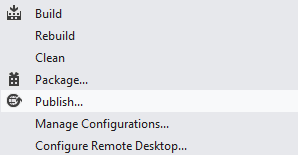
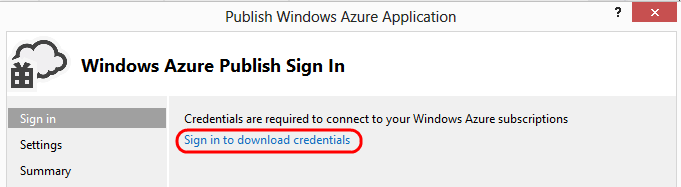
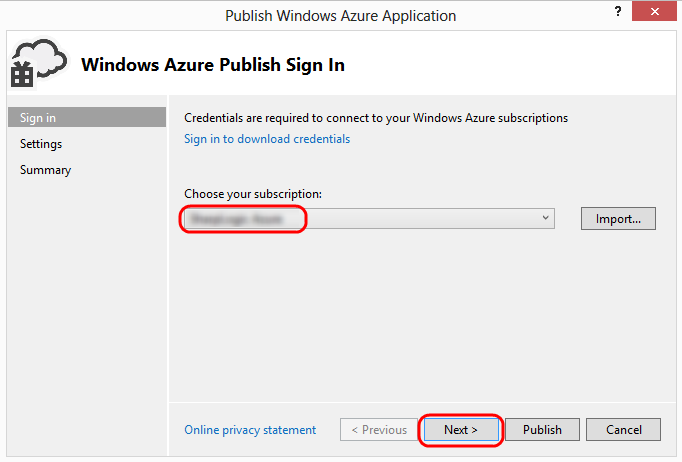
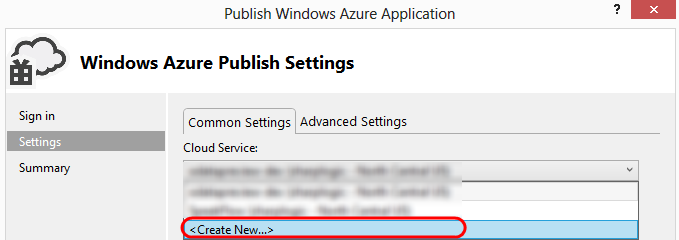
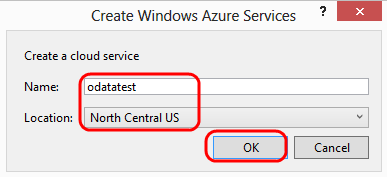
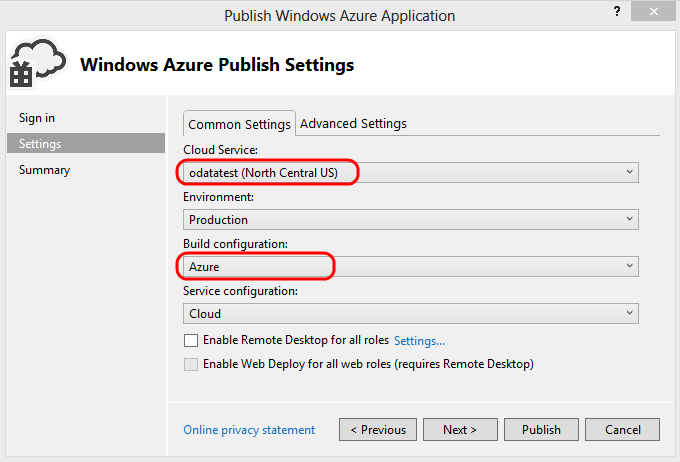
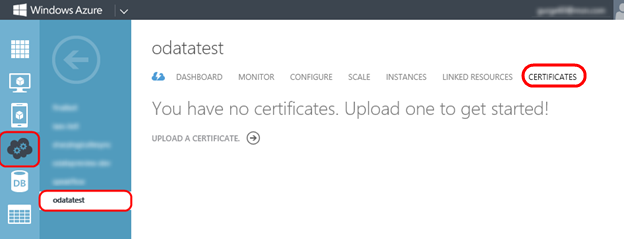
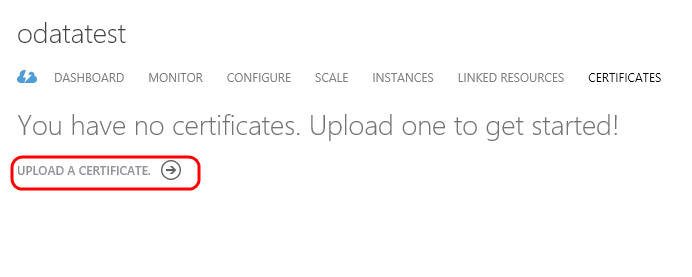
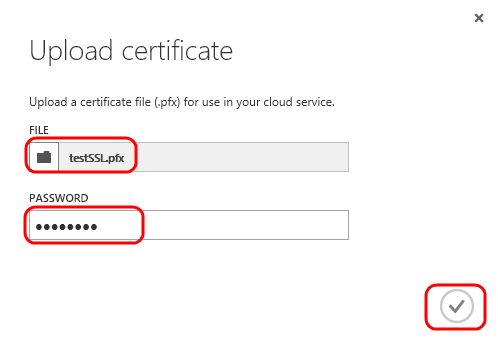
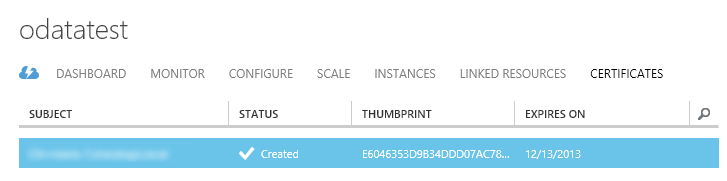
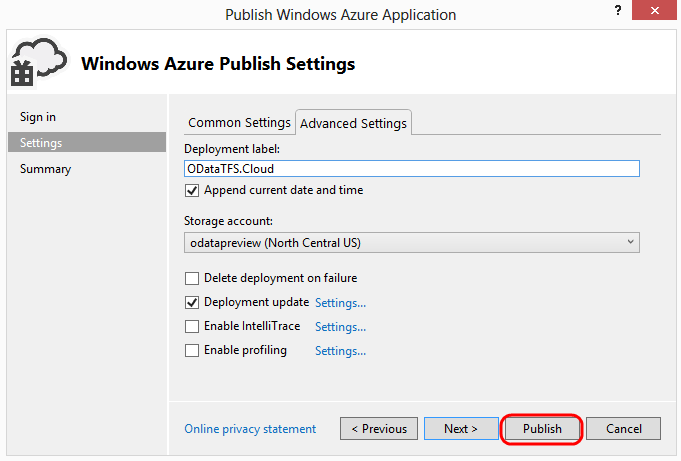
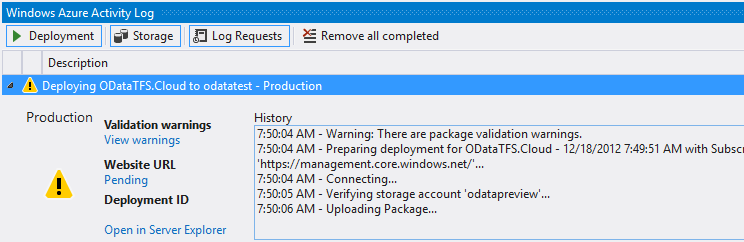
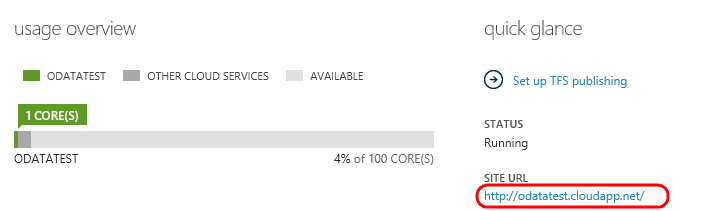
Running the OData Service locally using the Windows Azure Emulator

* + 1. To get a better idea of how the service will behave when it runs in Windows Azure, you can also take advantage of the emulator assuming that you have the Windows Azure SDK and tools installed.
    2. **Note:** You will need to install the IIS feature on your development machine to use the emulator.
  1. Open **Visual Studio 2012** as an **elevated** administrator. Hint: right-click on Visual Studio 2012 icon and select ‘Run as administrator’.
  2. Open the **ODataTFS.Cloud.sln** solution file in Visual Studio. Note that the ODataTFS.Cloud project is set as the **startup project**.
  3. Set the **Active Solution Configuration** to be **Azure**.
  4. Open the property pages for the **ODataTFS.Web** role definition by double-clicking on it in Solution Explorer.
     1. 
     2. Figure 16
     3. ODataTFS.Web role definition
  5. Load the **Settings** property page and update the **ODataTFS.TfsServer** setting to your TFS URL.
     1. 
     2. Figure 17
     3. Configuring TFS URL
  6. You should be able to press **F5** and have the OData Service deployed to the emulator straight away, although this process may take a few minutes to complete.
  7. Once the web role is deployed to the emulator, an instance of Internet Explorer will open to the HTTPS endpoint of the OData Service. Once you dismiss the security certificate warning, you will see the default landing page.
     1. 
     2. Figure 18
     3. OData Service landing page

Deploying the OData Service to IIS Server

* + 1. The steps below were written and verified using new installs of both Windows Server 2008 R2 Standard SP1 and Windows Server 2012 Standard, therefore you may need to adapt the steps to match up with your environment if there are any differences. If installing on a production server, make sure that administrators are fully aware of the changes that you plan to make.
  1. Install the **Web Server (IIS)** role and include the **ASP.NET role service**. Also add in the dependent role services when prompted. In Server 2012, this selection will be **Application Development | ASP.NET 4.5**. Also include the **Dynamic Content Compression** option.
  2. Once IIS is installed, install the **.NET Framework 4.5**. This is a pre-requisite for the Team Foundation Server client object model install. If you are running Server 2012, this will already have been installed when you added ASP.NET 4.5.
     1. **Note:** If the .NET Framework is installed before IIS, you will need to use the ASP.NET IIS Registration tool that comes with the framework. See <http://msdn.microsoft.com/en-us/library/k6h9cz8h(v=VS.100).aspx> for more details.
  3. It is suggested that you check for Windows Updates after installing the .NET Framework. Be aware that some updates may require a restart.
  4. Now we can build and publish ODataTFS.Web to the server. Open the **ODataTFS.sln** solution file in Visual Studio
  5. If you already have the Azure SDK installed, go ahead and skip to the next section. If you want to avoid installing the Azure SDK and will ultimately be deploying to IIS, you can simply remove the Microsoft.WindowsAzure.\* references from the Web project.
     1. 
     2. Figure 19
     3. Removing unneeded references for on-premises deployment
  6. Select **Build | Rebuild Solution** to ensure everything is up-to-date.
  7. **Right-click** on the **ODataTFS.Web** project in Solution Explorer and select the **Publish** option. You can use any publish method that you want, but the remaining instructions in the section are based on the **File System** method.
     1. 
     2. Figure 20
     3. Publishing the website
  8. Copy the published website files to the server.
  9. Run the **InstallTFSObjectModel.cmd** script from the *\bin\SetupFiles* folder of the published website. This will install the Team Foundation Server client object model assemblies to the GAC.
  10. Run the **SetupIIS.cmd** script from the *\bin\SetupFiles* folder of the published website to create the Team Foundation Server client object model cache folder and also set permissions for the IIS\_IUSRS group to the cache folder.
  11. Depending upon where you copied the website files, you will also need to ensure that the **IIS\_IUSRS** group has **full permissions** to that location also. This can be done via command line or by going to the folder properties and security tab in a Windows Explorer window.
  12. Now we need to create and configure a new Web Site in IIS. Most of the site details such as name, port, and host name are up to you and your environment, just make sure that you use HTTPS and your SSL certificate for the binding.
      1. 
      2. Figure 21
      3. Adding a new Web Site to IIS
      4. **Note:** For testing purposes, you can use a self-signed certificate. One way to do this using IIS Manager is to navigate to the machine node, open Server Certificates, and select Create Self-Signed Certificate. To learn more about SSL and IIS, please see <http://learn.iis.net/page.aspx/144/how-to-set-up-ssl-on-iis-7/>.
  13. Once the web site has been created, we also need to make sure that the associated **application pool** is configured to use **.NET Framework 4**. It should be okay if the minor version is greater than that showed in the screenshot.
      1. 
      2. Figure 22
      3. Application Pool configuration
  14. At this point, the OData Service should be ready to go.

Configuring and Deploying the OData Service to Windows Azure

* + 1. If you would like to host a version of the OData Service in Windows Azure, follow these general steps.
  1. Open the **ODataTFS.Cloud.sln** solution file in Visual Studio.
  2. Set the **Active Solution Configuration** to be **Azure**.
  3. Let’s make sure that everything is setup and ready to go for deployment. Open the property pages for the **ODataTFS.Web** role definition.
     1. 
     2. Figure 23
     3. ODataTFS.Web role definition
  4. Load the **Configuration** property page and set the instance count and VM size as appropriate for your deployment. It is generally recommended that you deploy 2 instances for availability reasons, but that is up to you. You can also (optionally) setup the diagnostics storage account here.
     1. 
     2. Figure 24
     3. Configuring the Azure Web Role
  5. Load the **Settings** property page and update the **ODataTFS.TfsServer** setting to your TFS URL. The web application will pull its settings from here (rather than web.config) when running in a Windows Azure environment.
     1. 
     2. Figure 25
     3. Configuring the Azure Web Role
  6. Load the **Endpoints** property page and note that there is a single public endpoint using HTTPS and public port 443. The **SSL Certificate Name** refers to the name of the certificate to use for SSL. If you change the name of the certificate you will need to update this property as well.
     1. 
     2. Figure 26
     3. Configuring the Azure Web Role
  7. Load the **Certificates** property page to view the certificate that is configured for use by the HTTPS endpoint that we just looked at. You will need to replace this certificate definition with the one you plan to use with the service. If you plan on deploying with a self-signed certificate for testing, you can follow the instructions from this article: <http://msdn.microsoft.com/en-us/library/ff803361.aspx>.
     1. 
     2. Figure 27
     3. Configuring the Azure Web Role
  8. Once you have the initial service configuration ready to go, you can go ahead and deploy it to Azure as you normally would. There are a number of ways to do this, but if you continue to follow the steps in this procedure we will walk through using the **Publish** wizard.
  9. **Right-click** on the **ODataTFS.Cloud** project node and select the **Publish** option.
     1. 
     2. Figure 28
     3. Starting the Publication wizard
  10. If this is your first time loading the Publish wizard, you will need to setup credentials in order to connect to your Windows Azure subscriptions. If you click the link to “**Sign in to download credentials**” you will be able to initiate the process of downloading a **Publish Settings File** (.publishSettings extension).
      1. 
      2. Figure 29
      3. Download credentials from the Azure Management Portal
  11. Once you have downloaded your **Publish Settings File** and imported it using the Publish wizard, you can then select the **subscription** you would like to use. Select the **Next** button once you have made your subscription selection.
      1. 
      2. Figure 30
      3. Selecting a subscription
  12. In the **Settings** step of the wizard, select the **Cloud Service** drop down and then select the **Create New** option.
      1. 
      2. Figure 31
      3. Creating a new cloud service
  13. Create a new cloud service by choosing a **name** and **location**. Select the **OK** button when done.
      1. 
      2. Figure 32
      3. Creating a new cloud service
  14. For the **Build Configuration** option, select the **Azure** configuration. Remember that this configuration includes the Azure conditional compilation symbol during the build process.
      1. 
      2. Figure 33
      3. Setting up publication options
  15. Additional configuration options are up to you – for example, you could enable **Remote Desktop** or **IntelliTrace** to help with debugging. Before we can publish, however, we need to upload the SSL cert that we are using for HTTPS using the [Windows Azure Portal](https://manage.windowsazure.com).
  16. Once you are logged into the management portal for Windows Azure, navigate to the newly created service and then select the **Certificates** tab.
      1. 
      2. Figure 34
      3. Uploading a certificate to the service
  17. Select the **Upload a Certificate** link.
      1. 
      2. Figure 35
      3. Link to upload certificate
  18. Select the exported **PFX file** that has your certificate, enter the **password**, and finally select the **OK** button to upload it.
      1. 
      2. Figure 36
      3. Uploading certificate to service
  19. Once the certificate is uploaded it will show up in the portal. Note that the thumbprint shown here should match the thumbprint in the Certificates tab of the role properties. Double-check this before continuing.
      1. 
      2. Figure 37
      3. Certificate successfully uploaded
  20. Once you are ready to start the publication process, select the **Publish** button.
      1. 
      2. Figure 38
      3. Starting the publication of the OData Service
  21. As the publication process proceeds, you can view status updates in the Windows Azure Activity Log window and at the Azure management portal site. Note that there are some package validation warnings, and if you investigate them you will see warnings about Team Foundation Server client object model assemblies not being in the package. This is okay, however, since there is a startup task that is responsible for installing the object model on startup.
      1. 
      2. Figure 39
      3. Publishing to Azure
  22. Once the deployment is complete, you should be able to load the OData Service default page. On the **Dashboard** tab for the service (in the management portal), scroll down to find the **SITE URL**. Copy and paste this link into a browser window, change the scheme to be **https** (we only setup a secure endpoint so HTTP will not work), and enjoy.
      1. 
      2. Figure 40
      3. Location of site URL

Summary

* 1. This sample OData Service provides a starting point that you can use to interface with your Team Foundation Server and expose useful data to clients. As such, this sample only exposes a small amount of the surface area exposed by the Team Foundation Server client object model. You are encourage to experiment and reference the description of available queries and operations listed on the service’s landing page to learn more.

References and Further Reading

* + For the latest information about the OData Service, FAQ, please see the TechNet wiki page at <http://social.technet.microsoft.com/wiki/contents/articles/15039.odata-service-for-team-foundation-server-v2.aspx>.
  + To learn more about WCF Data Services, please see <http://msdn.microsoft.com/en-us/library/cc668792(v=VS.110).aspx>.
  + To learn more about the WCF Data Services Toolkit, visit the CodePlex site at <http://wcfdstoolkit.codeplex.com/>. To learn more about how the Toolkit works, see the blog article at <http://lostintangent.com/post/3189655590/you-want-to-wrap-odata-around-what>.

To give feedback please write to [TFSOData@Microsoft.com](mailto:TFSOData@Microsoft.com)

Copyright © 2013 by Microsoft Corporation. All rights reserved.