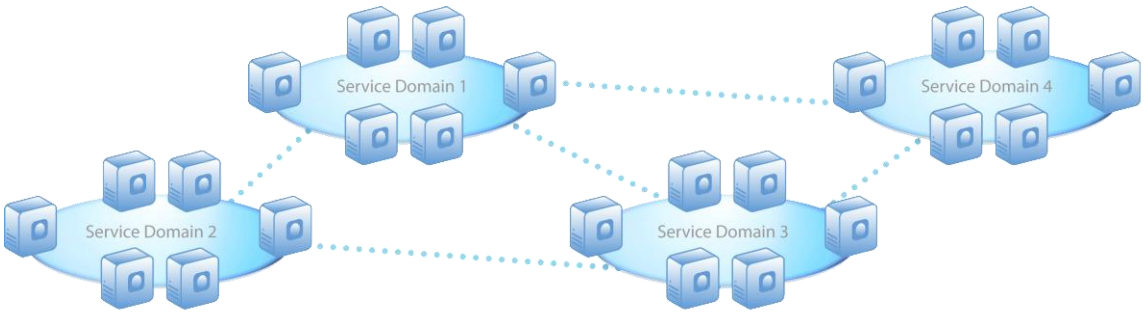
.NET StockTrader 5.0 Installation and Configuration

Reconfiguring .NET StockTrader and Configuration Service 5.0 Overview

6/9/2011



Microsoft .NET Windows Azure Microsoft SQL Azure

THIS IS NOT A PRODUCT SPECIFICATION.

This document and related sample code supports Windows Server® 2008 R2, Windows Azure, SQL Azure, Azure AppFabric and the Microsoft .NET Framework 4.0 as a redistributable sample application kit.

The information contained in this document represents the current view of Microsoft Corp. on the issues disclosed as of the date of publication. Because Microsoft must respond to changing market conditions, this document should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Microsoft grants you the right to reproduce this guide, in whole or in part.

Microsoft may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering subject matter in this document, except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

Microsoft, Windows Server, Windows Azure, SQL Azure, SQL Server, the Windows logo, Windows, Active Directory, Windows Vista, Visual Studio, Internet Explorer, Windows Server System, Windows NT, Windows Mobile, Windows Media, Win32, WinFX, Windows PowerShell, Hyper-V, and MSDN are trademarks of the Microsoft group of companies.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

Introduction to ConfigWeb

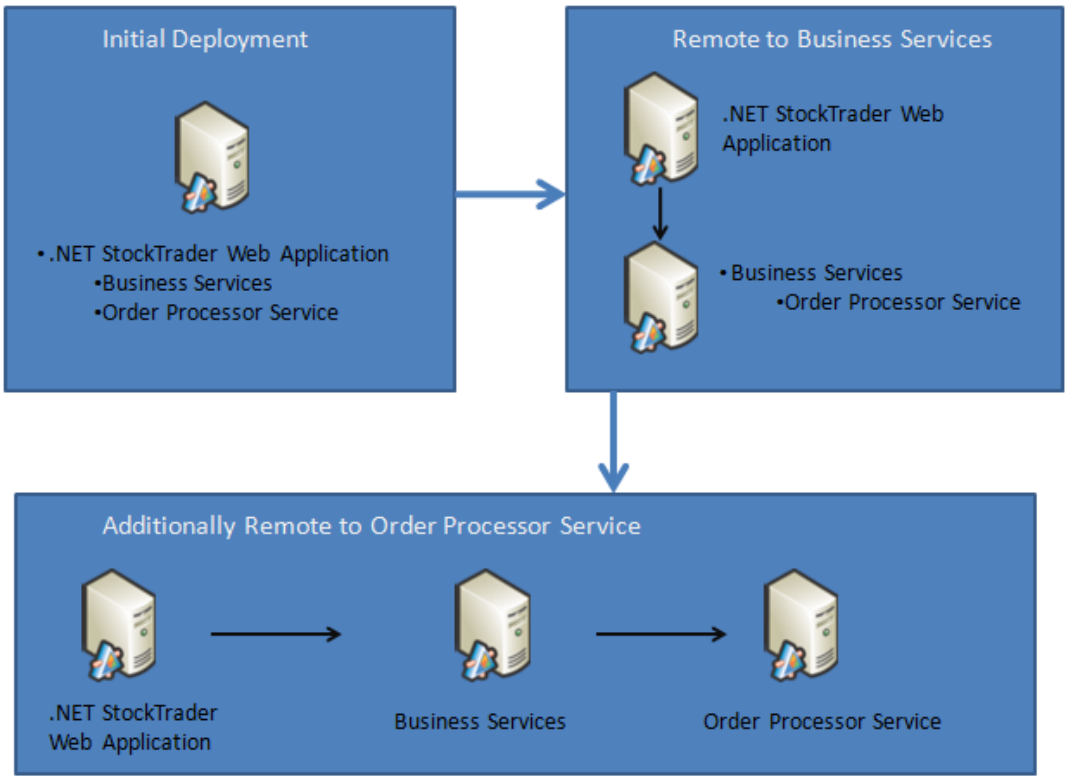
This document provides the basic steps used to reconfigure the .NET StockTrader 5 using the ConfigWeb management site. ConfigWeb is a separate Web application installed with .NET StockTrader that can remotely attach to any service or application that implements the Configuration Service 5.0. To access ConfigWeb, simply click on the ConfigWeb menu item within the .NET StockTrader Web application (http://localhost/trade), or the StockTrader Windows Start menu group. The configuration system utilized by StockTrader is contained in separate base classes and assemblies such that any developer, if desired, can implement the system in their own applications or services to help manage distributed application configuration data, and achieve load balancing and failover at the service operation level. A Visual Studio template and wizard is provided, and you can read the related document *Using the Configuration Service 5.0 Visual Studio Template*.

The Basic Deployment Options

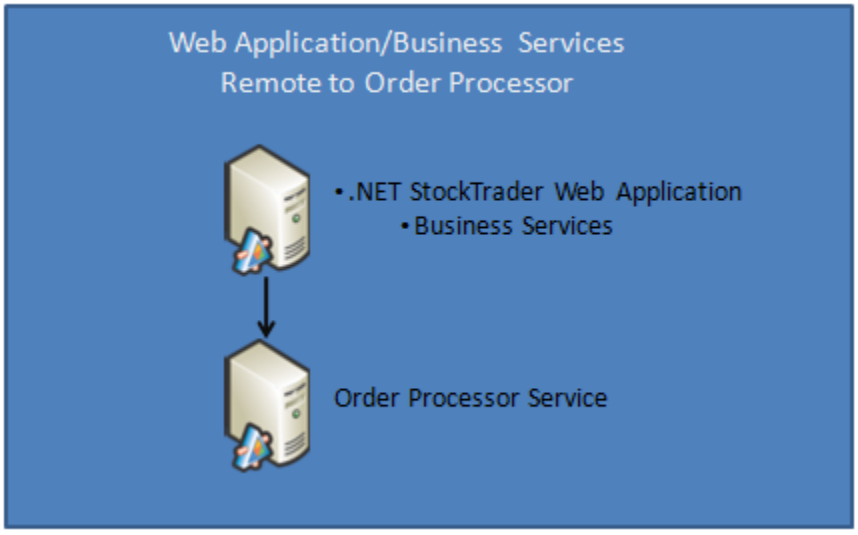
When logging into ConfigWeb, you can directly specify an address to any of the core StockTrader components that are running, including:

- .NET StockTrader Web application
- .NET StockTrader Business Services (hosted in IIS or in a .NET self-host program, accessible via the Windows Start menu)
- .NET StockTrader Order Processor Service (hosted in a .NET self-host program, accessible via the Windows Start menu)

These elements are initially configured on a default install to **run within a single ASP.NET process, without utilizing any remote services,** although the tiers are still logically partitioned within the host ASP.NET worker process. The relationships that can be changed look as follows (you can explore all of these options using a single computer, as each service host is configured to utilize separate ports):



Additionally, the following deployment option is possible:



Remote Modes and Advanced Web Service Security

In .NET StockTrader 5.0, two optional modes demonstrate WS-* security. Activating and using both modes are illustrated in step-by-step procedures in a different document, see http://msdn.microsoft.com/stocktrader for details. Neither of these modes is active by default for an on-premise application. However, for the Azure deployment, one of these secure modes is always used.

Securing Business Services and Order Processor with Service Certificate and Custom Username Validation

The first option demonstrates using an X.509 service certificate over a ws2007HttpBinding or netTcpBinding with a security mode of TransportWithMessageCredentials. Client credentials are set to Username (no certificate is required on the client; messages are encrypted within the transport layer), and this mode demonstrates authenticating users using a custom username store, in this case directly using the Configuration Service Users table to authenticate users connecting to the business service layer (BSL). A base class provided within the Configuration Service does the validation, using the standard .NET overrides (System.Security) to perform custom validation of the user name/password. By default, this base class validates the incoming credentials against the Configuration Service Users table (which is a convenient way to store such credentials for authentication); however this can easily be replaced with other logic to use other credential stores such as Active Directory, custom LDAP stores, etc.

Securing Order Processor with Service Certificate and Custom Client Certificate Validation

The second option illustrates client credentials set to Certificate (service and client each require their own X.509 certificate; client certificate is used for identification/authorization to the remote services). The client certificate is validated using a custom certificate validator that checks the thumbprint of the certificate, only allowing access to the secured endpoint for specific client certificates specified in the validator class. The validator class, like the UserName validator, is provided as a base class with the Configuration Service, using the standard .NET X.509 framework classes to do the validation.

Remote Modes and Interoperability

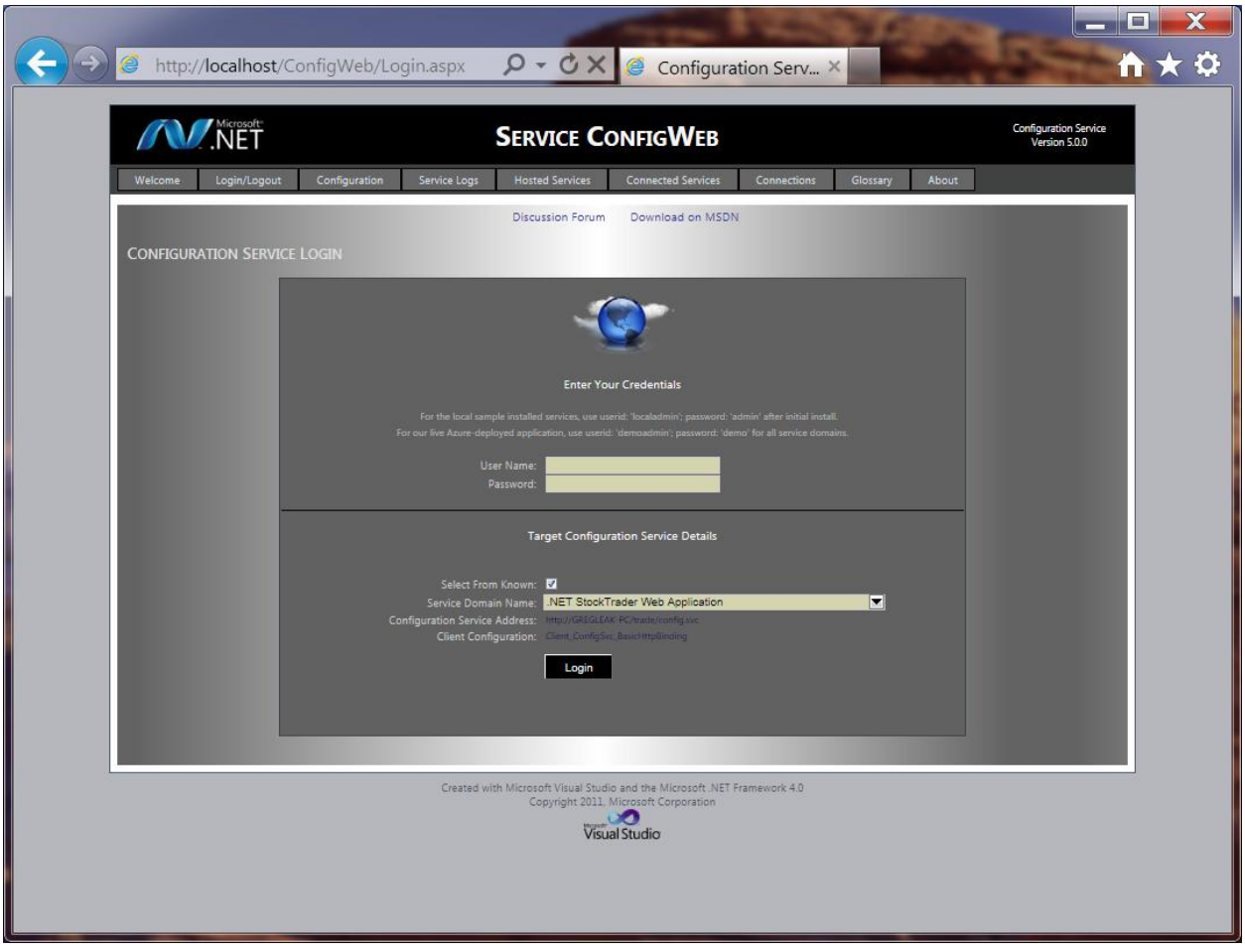
Several additional Connected Service definitions ship with the default install of .NET StockTrader 5.0 that demonstrate interoperability with non-Microsoft platforms. Third-party elements are required for these modes to operate:

- .NET StockTrader ASP.NET Web Application and/or .NET StockTrader smart client to IBM WebSphere 7.x Trade Business Service Layer over SOAP. This mode requires installation of IBM WebSphere 7 with the Trade 7 sample installed. Note that the IBM WebSphere Trade 7 Java Server Pages Web application can also be used seamlessly with the .NET middle tier services. The Java-based Trade 7 application is included in the StockTrader download.

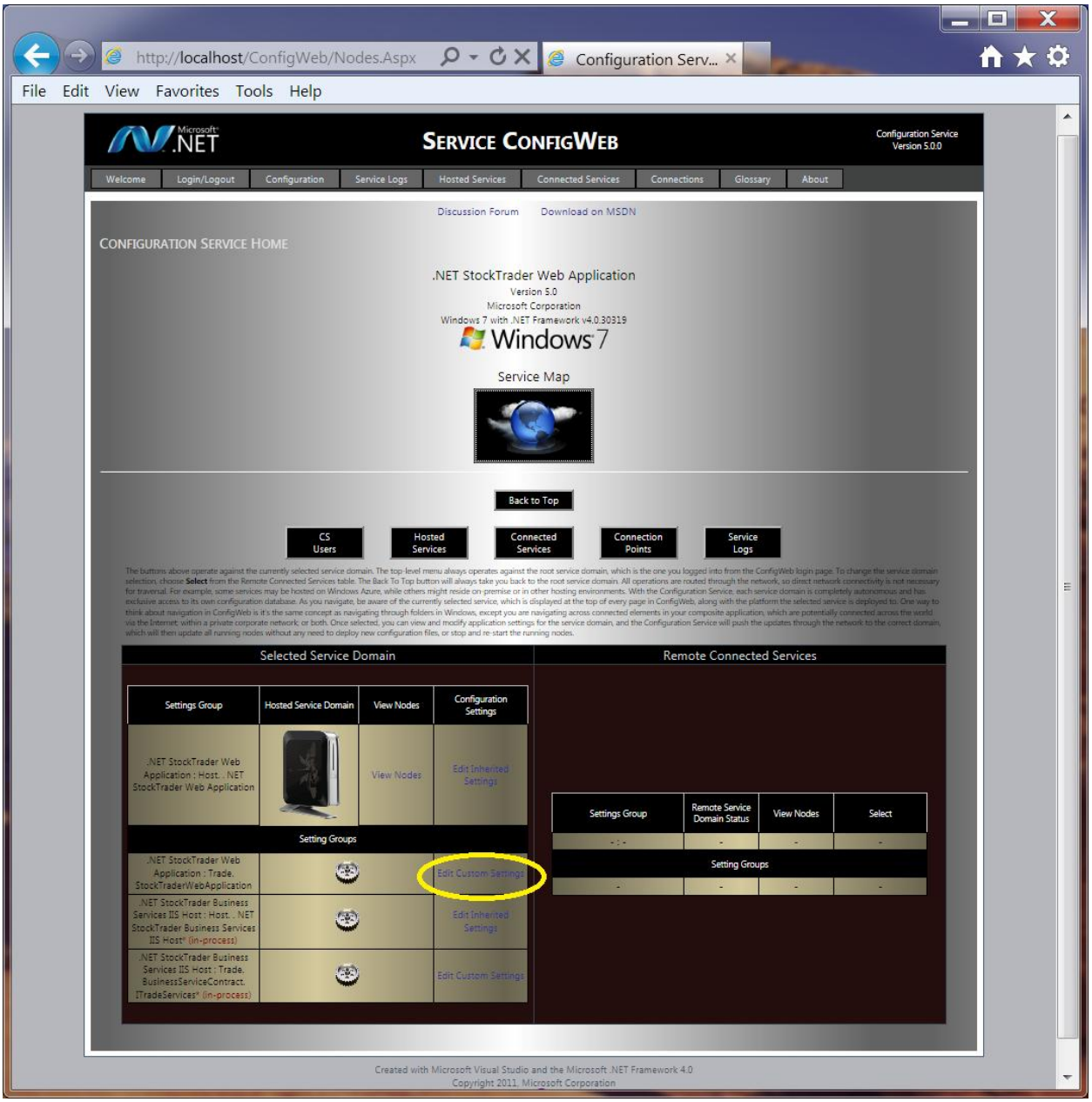
Initial ConfigWeb Login and Reconfiguration Steps

To get started, we will use StockTrader to provide an overview of using the Configuration Service and ConfigWeb for basic changes to the modes in which the composite application operate. Later, we will reconfigure services for the advanced security modes. Note that after the initial install of .NET StockTrader, the application is fully functional (operating in a no-services mode as a standalone Web application); so you can already run the .NET StockTrader Web application at http://localhost/trade before completing any steps in this guide. You can also run the smart client, accessible via the Windows Start menu.

To reconfigure .NET StockTrader to use a remote service tier for the business service layer, log in to ConfigWeb (http://localhost/configweb) by pointing to the **StockTrader Web application configuration service**. You can login as username '**localadmin'**, password '**admin'**. The drop-down selection (which can be customized) shows configuration addresses for the StockTrader services and other samples included with the download, and the first address listed is pre-specified to point to the .NET StockTrader Web application Configuration Service. You can change the server name by deselecting **Select from Known List**, otherwise by default the server hosting ConfigWeb will be listed.

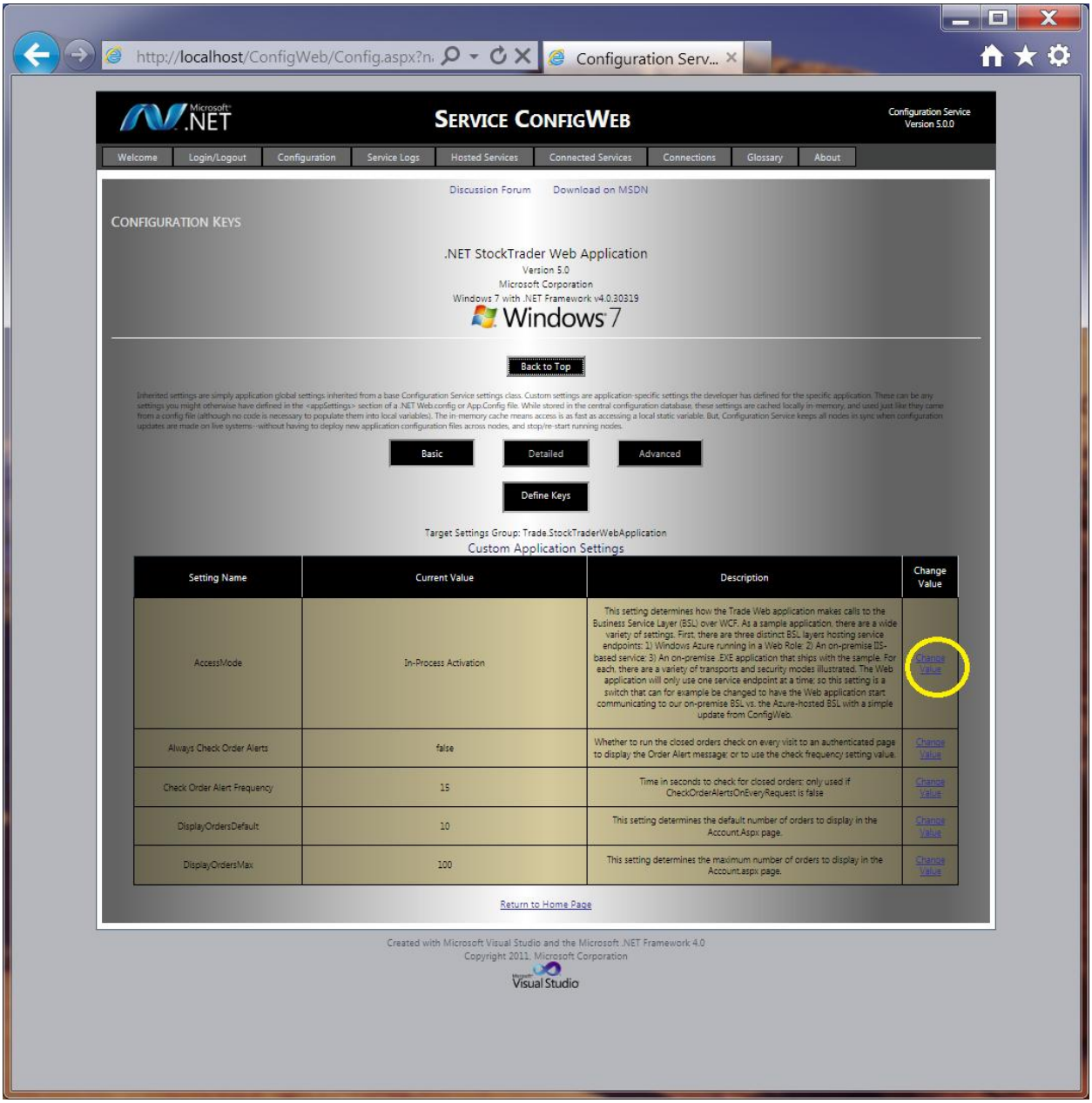


On login you will see the configuration home page, as follows:



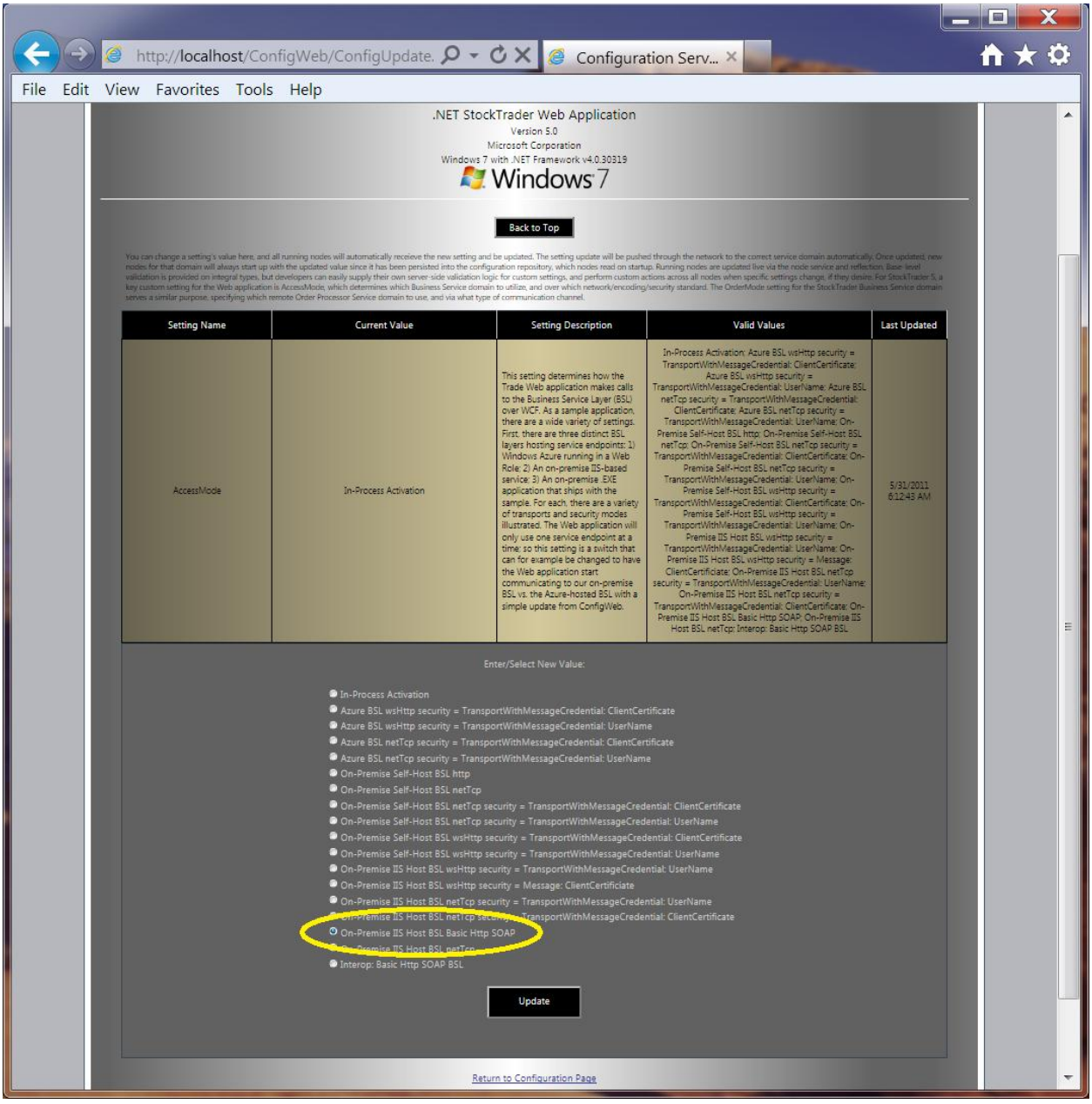
In the following steps, we will reconfigure the StockTrader Web application to connect to remote Business Services, hosted in IIS. To do so, click on the Edit Configuration link for the StockTrader Web application circled in the image above.

This will bring you to the main configuration settings for the StockTrader Web application. These are depicted below. Settings are separated into three groupings: Basic, Detailed and Advanced.


http://localhost/ConfigWeb/Config.aspx?n
Configuration Serv...
Microsoft .NET
SERVICE CONFIGWEB
Configuration Service Version 5.0.0
Welcome
Login/Logout
Configuration
Service Logs
Hosted Services
Connected Services
Connections
Glossary
About
Discussion Forum
Download on MSDN
CONFIGURATION KEYS
.NET StockTrader Web Application
Version 5.0
Microsoft Corporation
Windows 7 with .NET Framework v4.0.30319
Windows 7
Back to Top
Inherited settings are simply application global settings inherited from a base Configuration Service settings class. Custom settings are application-specific settings the developer has defined for the specific application. These can be any settings you might otherwise have defined in the <appSettings> section of a .NET Web.config or App.Config file. While stored in the central configuration database, these settings are cached locally in-memory, and used just like they came from a config file (although no code is necessary to populate them into local variables). The in-memory cache means access is as fast as accessing a local static variable. But, Configuration Service keeps all nodes in sync when configuration updates are made on live systems--without having to deploy new application configuration files across nodes, and stop/re-start running nodes.
Basic
Detailed
Advanced
Define Keys
Target Settings Group: Trade.StockTraderWebApplication
Custom Application Settings
Setting Name
Current Value
Description
Change Value
AccessMode
In-Process Activation
This setting determines how the Trade Web application makes calls to the Business Service Layer (BSL) over WCF. As a sample application, there are a wide variety of settings. First, there are three distinct BSL layers hosting service endpoints: 1) Windows Azure running in a Web Role; 2) An on-premise IIS-based service; 3) An on-premise .EXE application that ships with the sample. For each, there are a variety of transports and security modes illustrated. The Web application will only use one service endpoint at a time; so this setting is a switch that can for example be changed to have the Web application start communicating to our on-premise BSL vs. the Azure-hosted BSL with a simple update from ConfigWeb.
Change Value
Always Check Order Alerts
false
Whether to run the closed orders check on every visit to an authenticated page to display the Order Alert message; or to use the check frequency setting value.
Change Value
Check Order Alert Frequency
15
Time in seconds to check for closed orders; only used if CheckOrderAlertsOnEveryRequest is false
Change Value
DisplayOrdersDefault
10
This setting determines the default number of orders to display in the Account.Aspx page.
Change Value
DisplayOrdersMax
100
This setting determines the maximum number of orders to display in the Account.aspx page.
Change Value
Return to Home Page
Created with Microsoft Visual Studio and the Microsoft .NET Framework 4.0
Copyright 2011, Microsoft Corporation
Visual Studio

Click on the **Change Value** link circled above for the AccessMode setting. You will next be able to update the configuration to one of several different ways you can connect to remote Business Services. The AccessMode setting determines which connection types to make active and utilize while users are using the StockTrader Web application (http://localhost/trade).

For now, we will change the setting to IIS-Hosted .NET 4.0 services, corresponding to a setting of **On-Premise IIS Host BSL Basic Http SOAP**, which you should select as depicted. Note that note all modes are initially active. Also note that if you deploy the StockTrader Business Services to Windows Azure, this is where you could link your on-premise StockTrader Web application to cloud-hosted services on Azure.

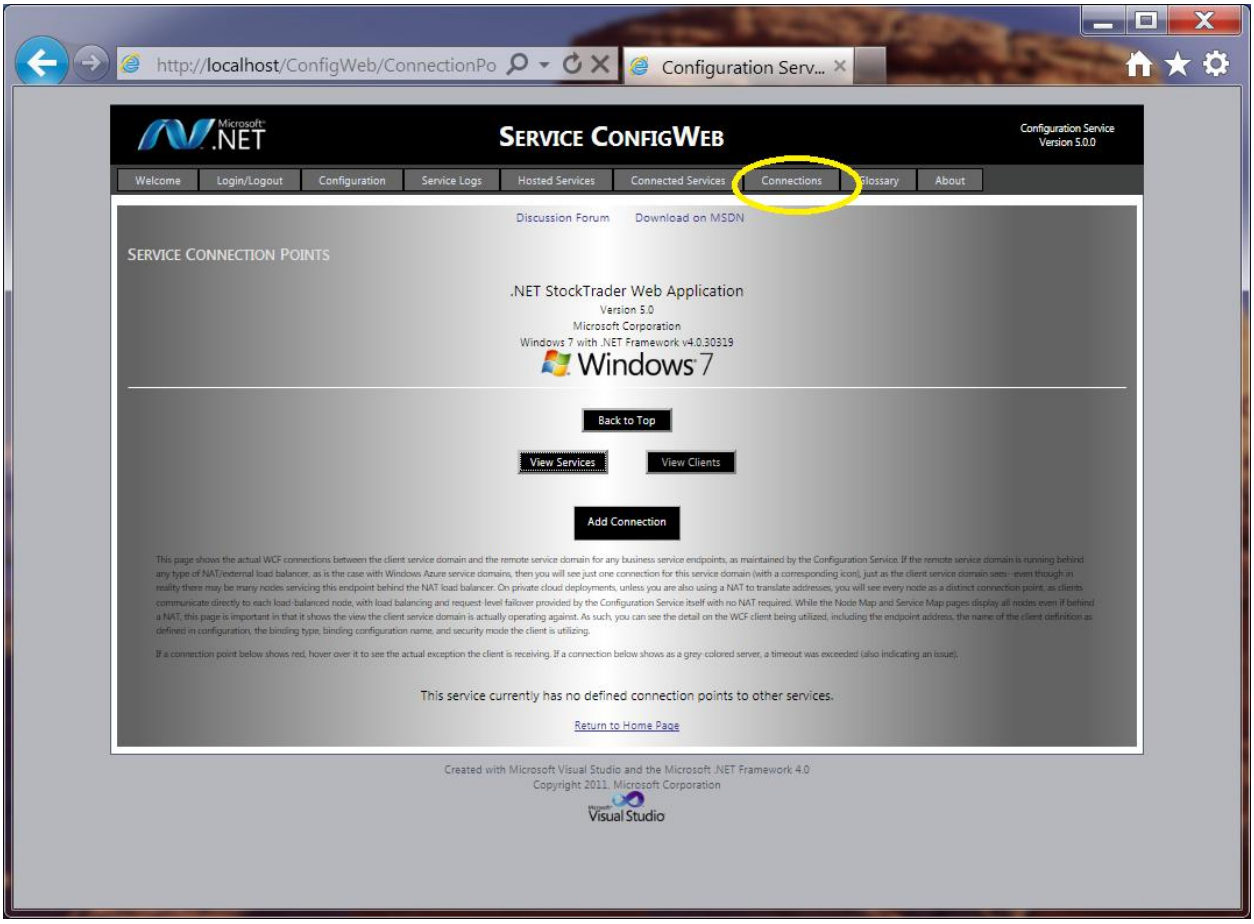


Click on the **Update** button to make the setting active. If your StockTrader Web application has been deployed to many nodes, or is scaled out on Windows Azure, all nodes will be immediately synchronized with the new setting.

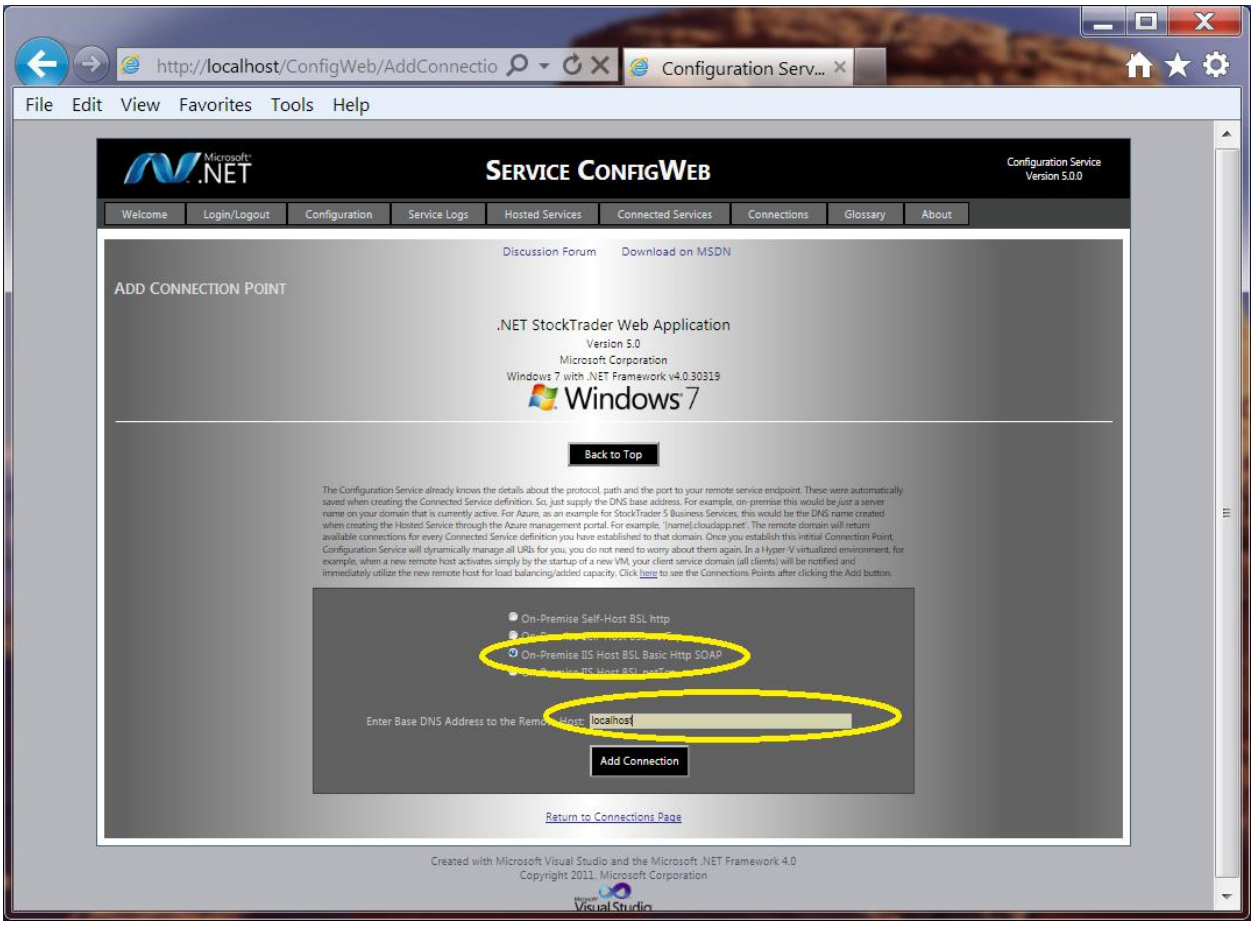
Next, click on the menu item **Configuration** to return to the main ConfigWeb page:

You will notice that we now have Business Services (and Order Processor Service) no longer showing up under the root StockTrader Web application, because all business services (and all connectivity to the StockTraderDB database) are now set to be remotely activated by a separate business service tier running in a different **physical** tier. However, there are no 'Remote Connected Services' showing up either. The .NET StockTrader Web application needs to be connected to a running instance of the

remote Business Services. To do so, you will click on the **Connection** menu item to add an initial connection point:



Click on **Add Connection**, and you will be presented with a list of options of the different Business Services you can connect to (these options are available because they have already been defined as Connected Services in the StockTrader Web application repository):

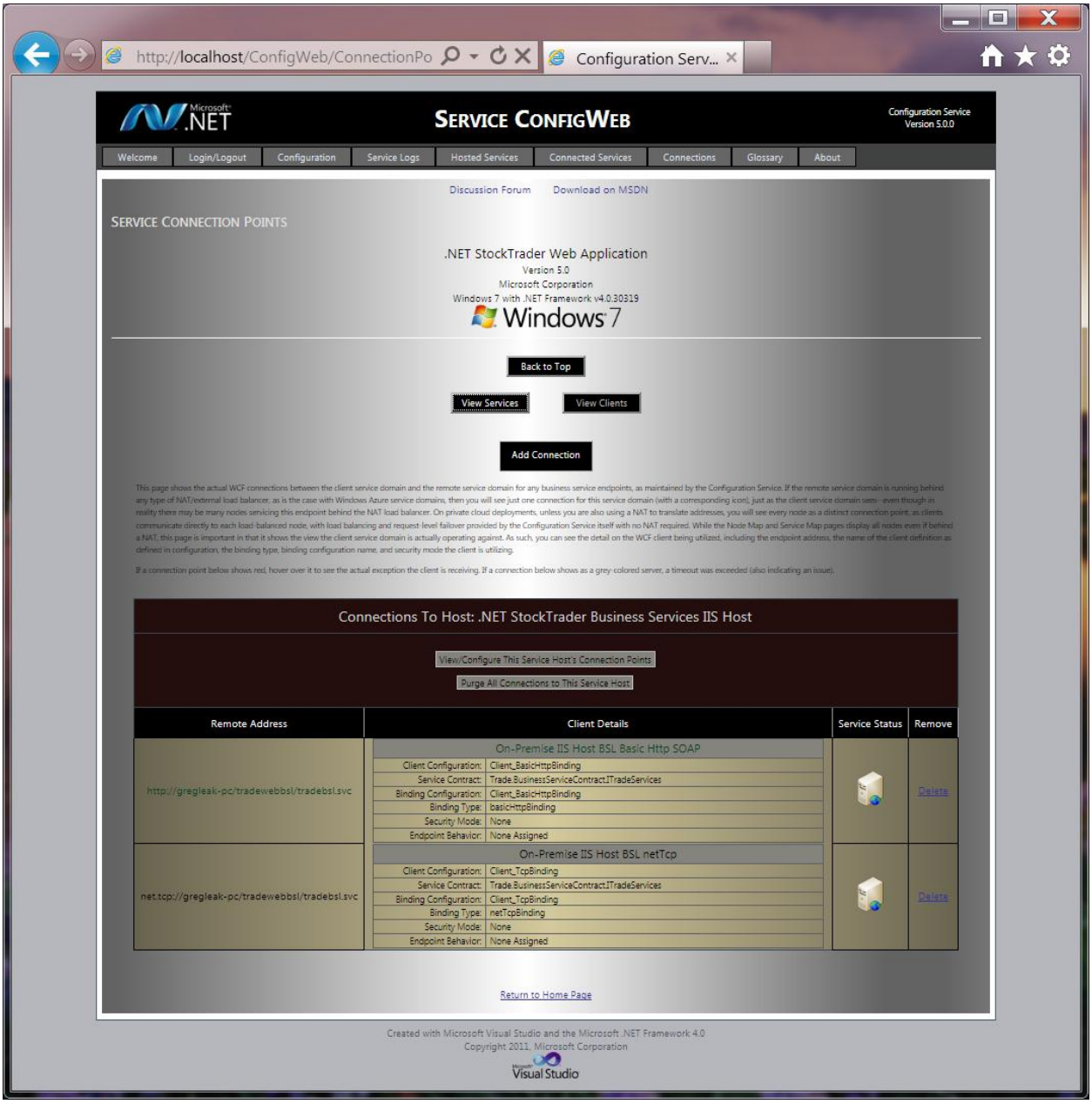


Select the option **.NET StockTrader Business Services IIS Host Http/Soap**.[1]

Next, simply type in the name or address of any machine you installed StockTrader on. The repository already stores the logical URI (path, port) information, so you just need the server name or IP address here. If remote business services (the /TradeWebBSL Web application) are running on multiple physical servers (nodes), the Configuration Service will be aware of this, but you still need to initially point to a single node running IIS-hosted business services to establish the initial dynamic clustering relationship.

After clicking **Add Connection**, click on the **Return** button to go back to the **Connections** page:

[1] Note that the .NET StockTrader Business Services IIS Host Tcp/Binary endpoint will only be viable on installations of Business Services on IIS 7.0, as IIS 6 (Windows Server 2003) and IIS 5 (Windows XP/SP2) only support http-based endpoints. The StockTrader setup only activates tcp endpoints for IIS when installing on Windows Vista/above or Windows Server 2008/above, accordingly, although the connection options will still appear in this menu. The self-hosted tcp endpoint (.NET StockTrader Business Service Tcp/Binary) will, however, be viable on Windows Server 2003 and Windows XP/SP2, as the self-host program (not hosted in IIS) can host endpoints on non-http endpoints.



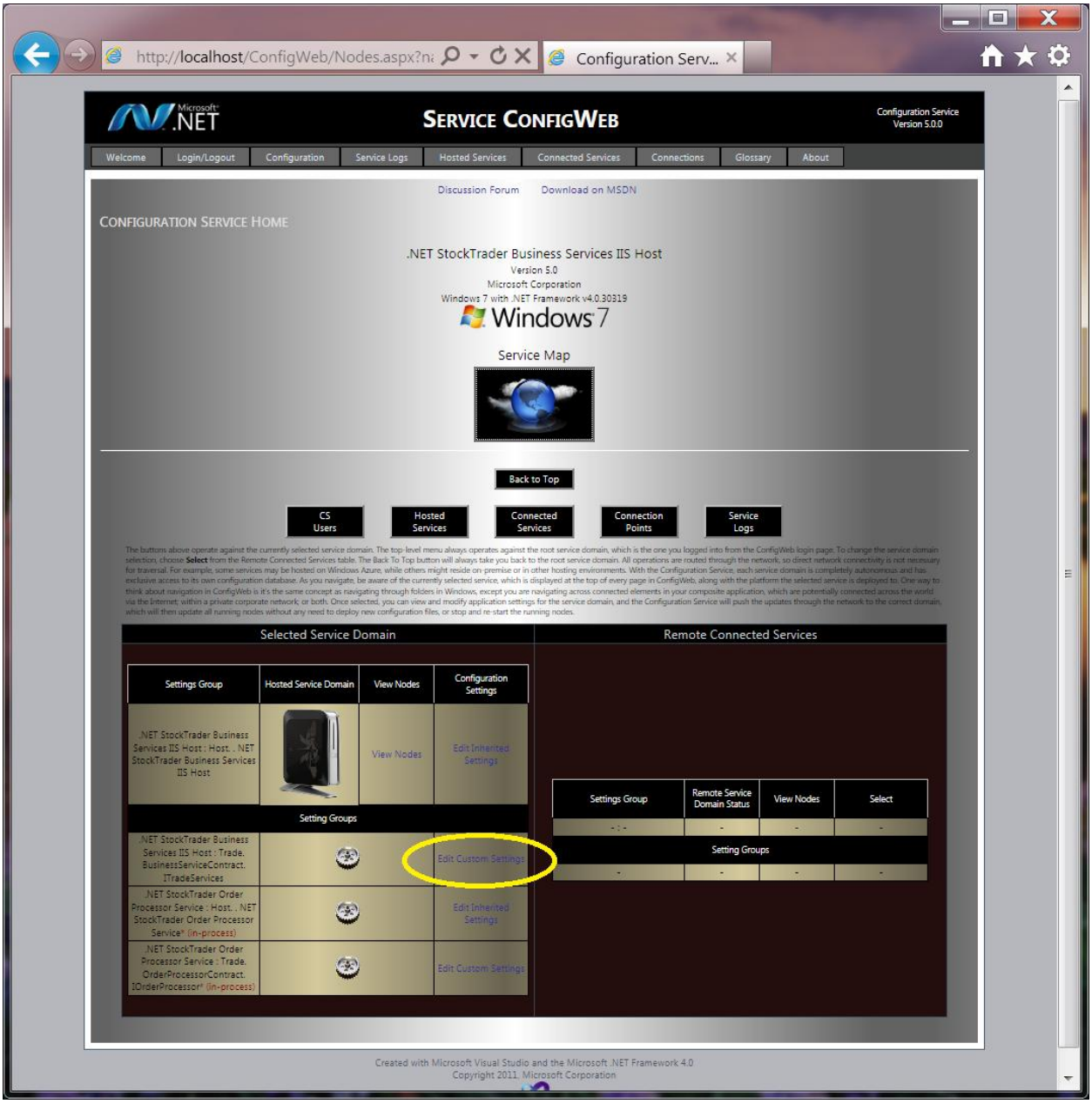
Note that on IIS7 (Windows Vista or Window Server 2008), two connections will be added. This is because the StockTrader Web Application has two Connected Service definitions in its repository, both hosted by the same IIS TradeWebBSL host application: one is the HTTP/SOAP endpoint, the other is a TCP/binary endpoint. Since it is subscribed to both, the Configuration Service will automatically create Connection Points to both, even though only one will be active at a time, based on the AccessMode setting, per the design of the StockTrader Web application. If you are running on an earlier version of IIS (IIS 5 on Windows XP or IIS 6 on Windows Server 2003), you will only see the HTTP/SOAP Connection Point, since these earlier versions of IIS are not capable of hosting non-Http service endpoints. The active Connection Points will be highlighted in green.

You can now login to the actual StockTrader Web application at http://localhost/trade (vs. ConfigWeb) as a registered site user, and now instead of running business services within the same process as the ASP.NET Trade Web application (the user interface tier), all activations are happening remotely to the separate IIS-hosted TradeWebBSL Web application via WCF/Web Services. Of course, you will not notice a difference as a StockTrader user in the StockTrader Web application itself. In this way, business services could be configured, for example, to run behind a second firewall, with all business logic and all database connectivity protected behind this firewall.

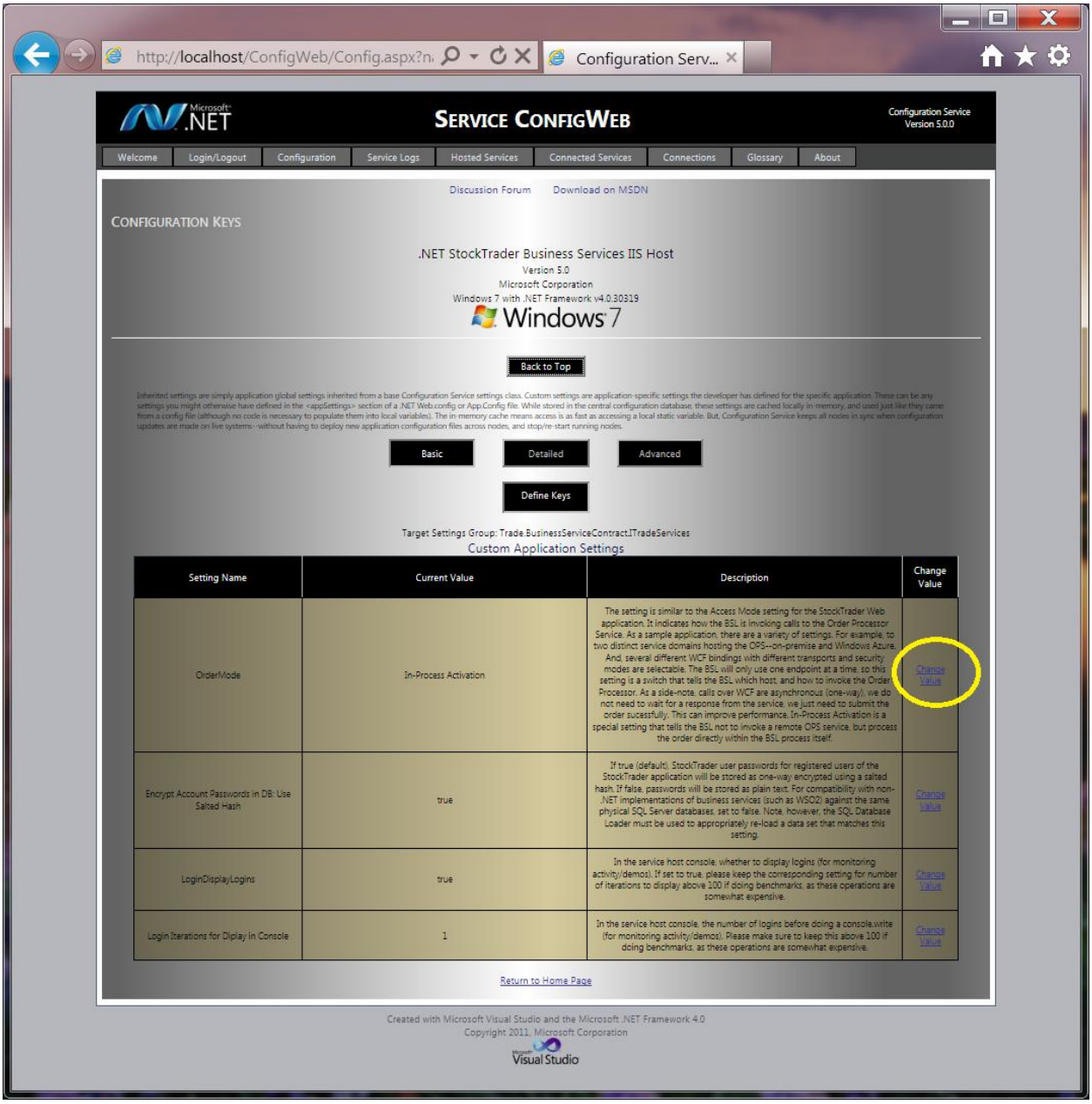
The next (optional) step to demonstrate further physical partitioning into distinct service layers will be to configure the Business Services to also remotely invoke, on a third tier, order processing to occur via a separate service via a service-to-service interaction. To configure how orders are processed, you will be configuring the StockTrader Business Service IIS Host, not the StockTrader Web application since the Web application is blind to anything beyond the Business Service tier. This is where the connected nature of the Configuration Service between service domains comes into play, using Linked Services. Instead of logging out, you can select the IIS Business Service tier directly, since the '**compositeadmin'** user is also registered as an administrator of this remote service within its own repository.

Next, go back to the main page by clicking on the **Configuration** menu item and select the Business Service IIS tier as the *top node* within ConfigWeb, by clicking as indicated below:

At this point, you are looking at the application from the Business Service tier down, as follows (the top node is always on the left, with its remote services shown on the right, much like Windows folder navigation):

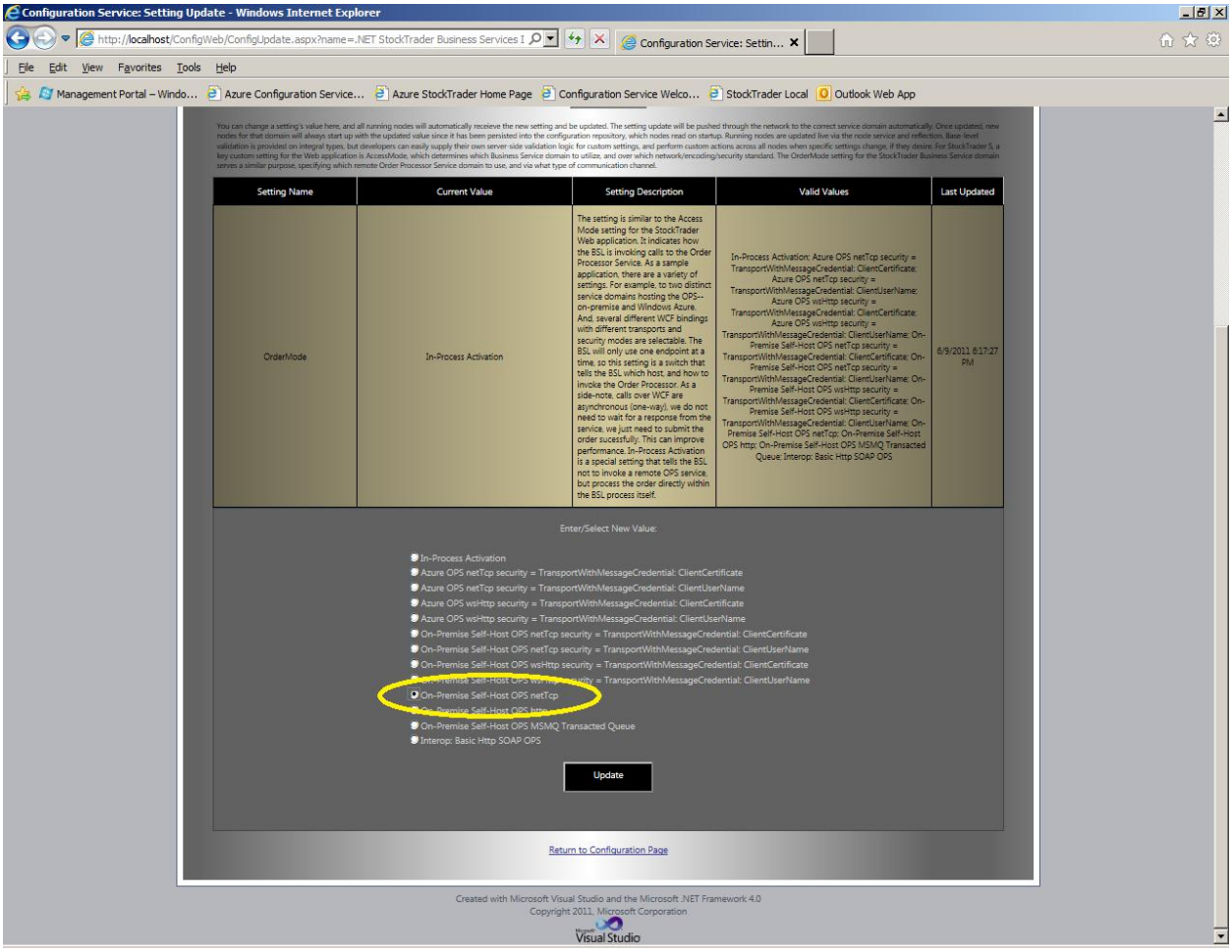


Now simply follow the same process completed for re-configuring the business services to be remotely activated by the Web application, except this time you will be configuring the IIS Business Service host to remotely activate the Order Processor Service. This is done via the **OrderMode** configuration setting for business services. Click on the **Edit Configuration** link as circled above. You can now change the OrderMode setting for business services:



Click on the **Change Value** link as circled above. Business services also has several different ways to connect to the Order Processor Service. We will connect here via asynchronous net.tcp messages.

Later, you can also activate the transacted MSMQ endpoint using ConfigWeb, and use it instead. See the notes later in this section about MSMQ, an endpoint that is not active by default but can be activated in just a few clicks via ConfigWeb. You need to make sure you have installed the base MSMQ service as a Windows feature, before you do this. The queues are created automatically by the OPS host when it starts (as a console application, windows host, or NT Service: all three host types are provided).

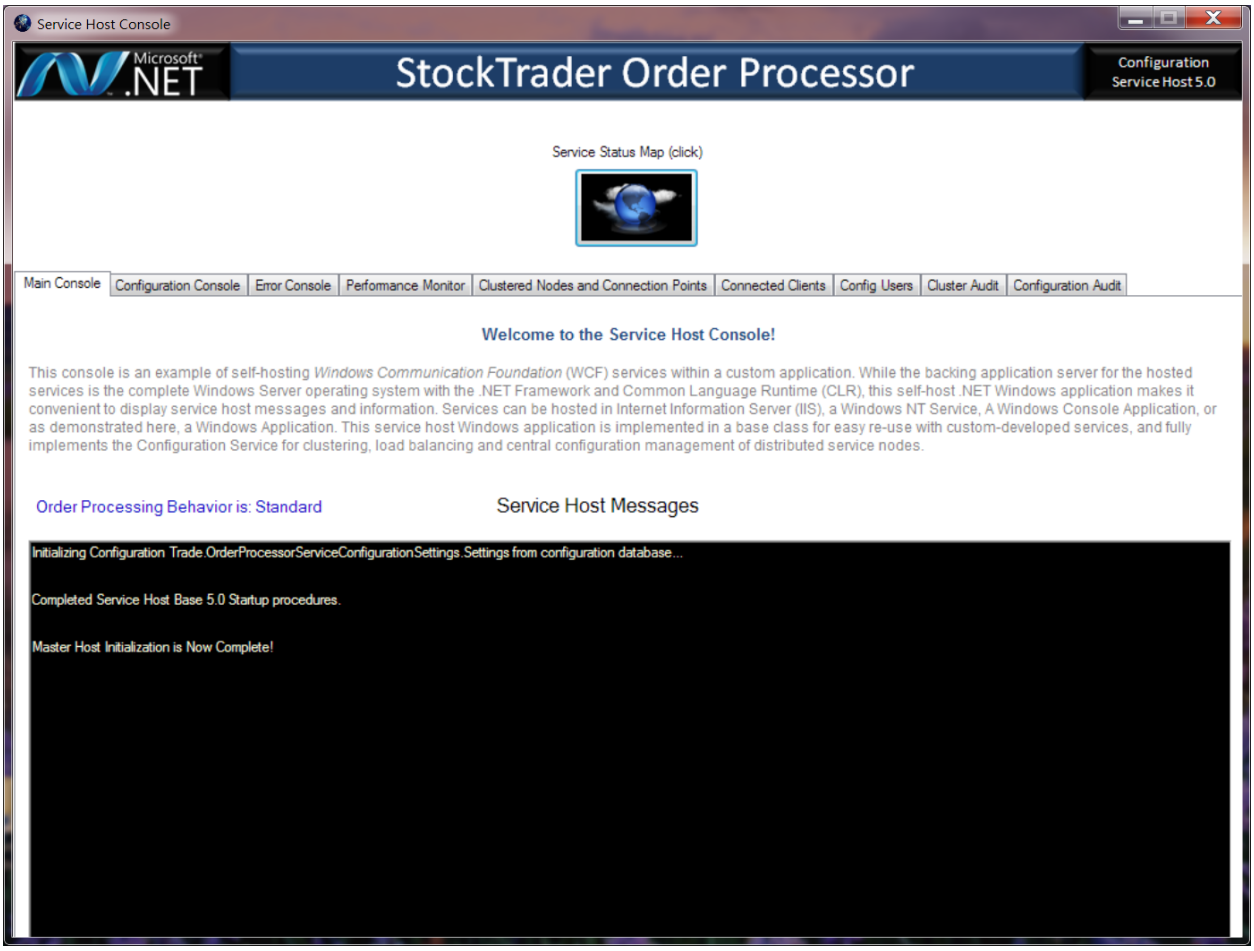


- Choose the On-Premise Self-Host OPS netTcp mode.
- Click the **Update** button to change the configuration to this remote mode.

The Order Processor Service is hosted in a separate .NET host program (or "self-hosted"): **Trade.OrderProcessorServiceHost.exe,** accessible via the StockTrader Start menu group in Windows. Order Processor Services **are not hosted in IIS** (although they could be hosted in IIS 7 via WAS hosting). But, as with the Business Services self-host, a Windows NT Service host type is provided as well, if you want to explore hosting services within an NT Service. The following interchangeable host programs are provided:

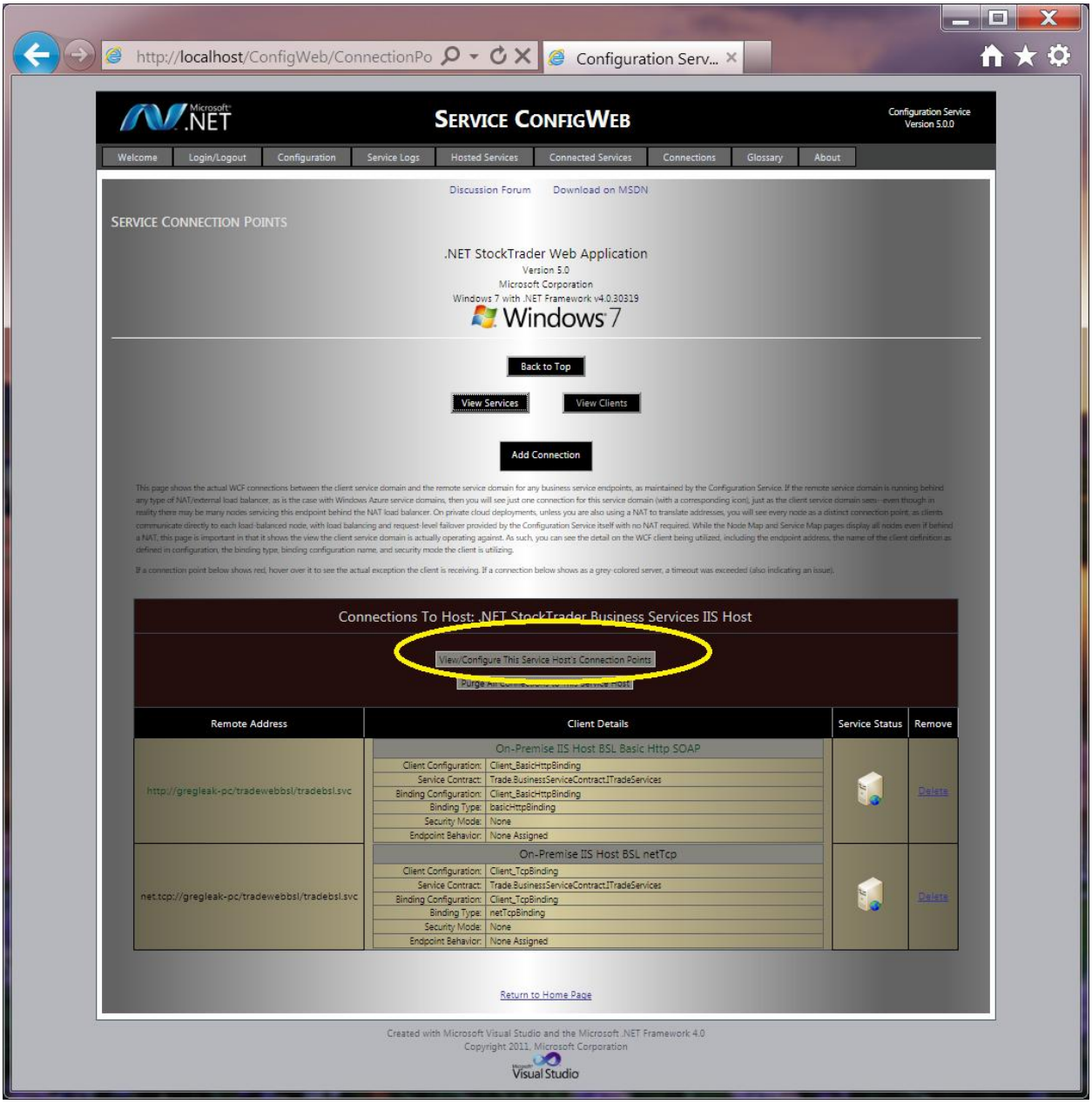
- Windows Host (inherits from base Config Service .NET Windows Forms class)
- Console Host
- NT Service Host (run \stocktrader install dir\setup\util**installOPSService.cmd** from a CMD prompt with –i to install and –u to install/uninstall as an NT Service. It will install in auto-start mode; so do not forget it will be active on re-boots unless you change this.

As before with the Web application, you will need to add an initial connection within business services to the Order Processor Service so that orders can actually be placed by users in the StockTrader Web application. First, however, you will need to startup the Order Processor Service host program. This can be found in the **Windows Start** menu items for .NET StockTrader, under **.NET StockTrader Self Hosts**. Start **Trade.OrderProcessorServiceHost.exe**:[2] Make sure to choose "Run As Administrator", or else WCF will not have the correct permissions to register and host its endpoints.

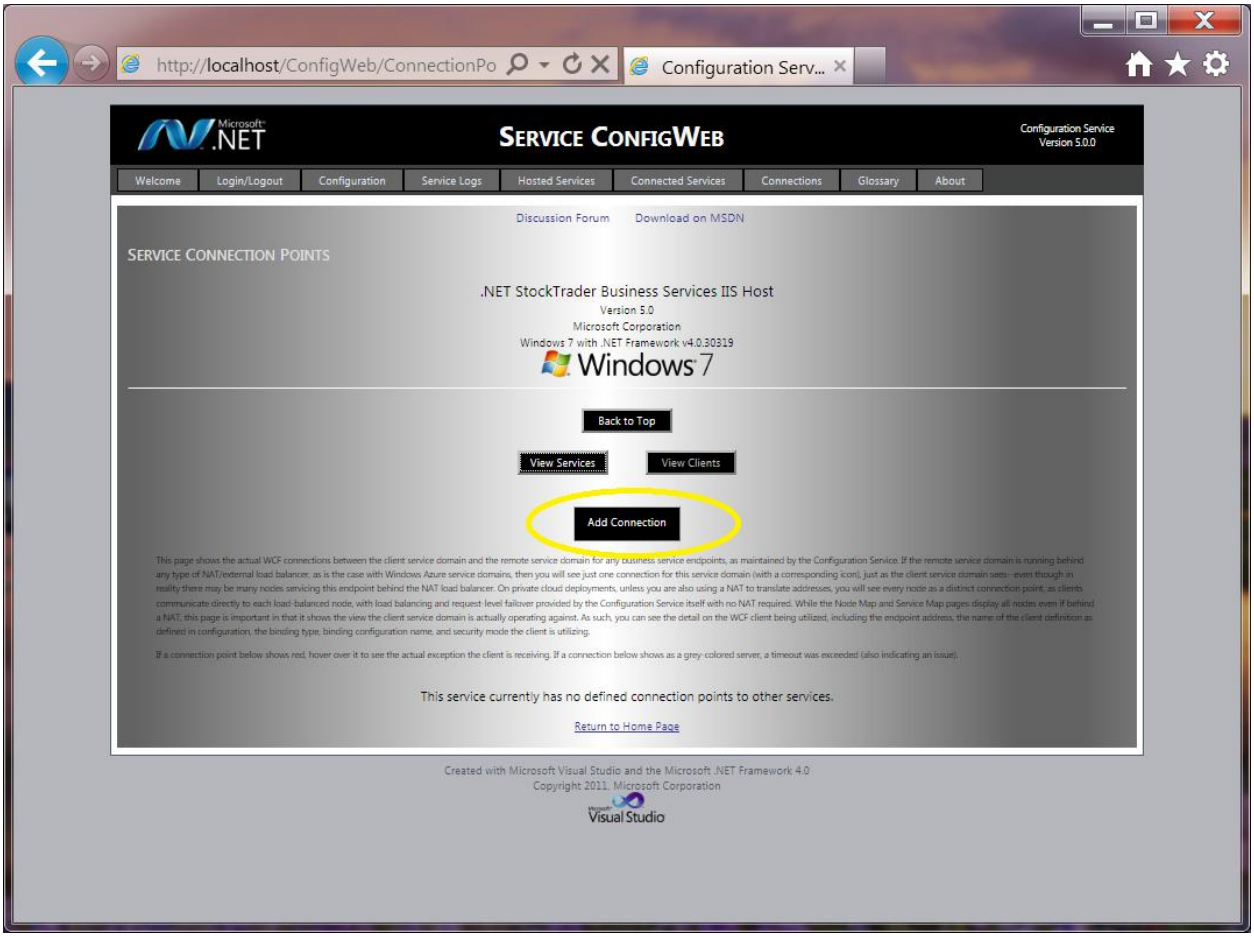


At this point, you are ready to add an initial connection from the Business Services-IIS Host layer to the running remote Order Processor Service. To do so, in ConfigWeb you will go back to the Connections page:

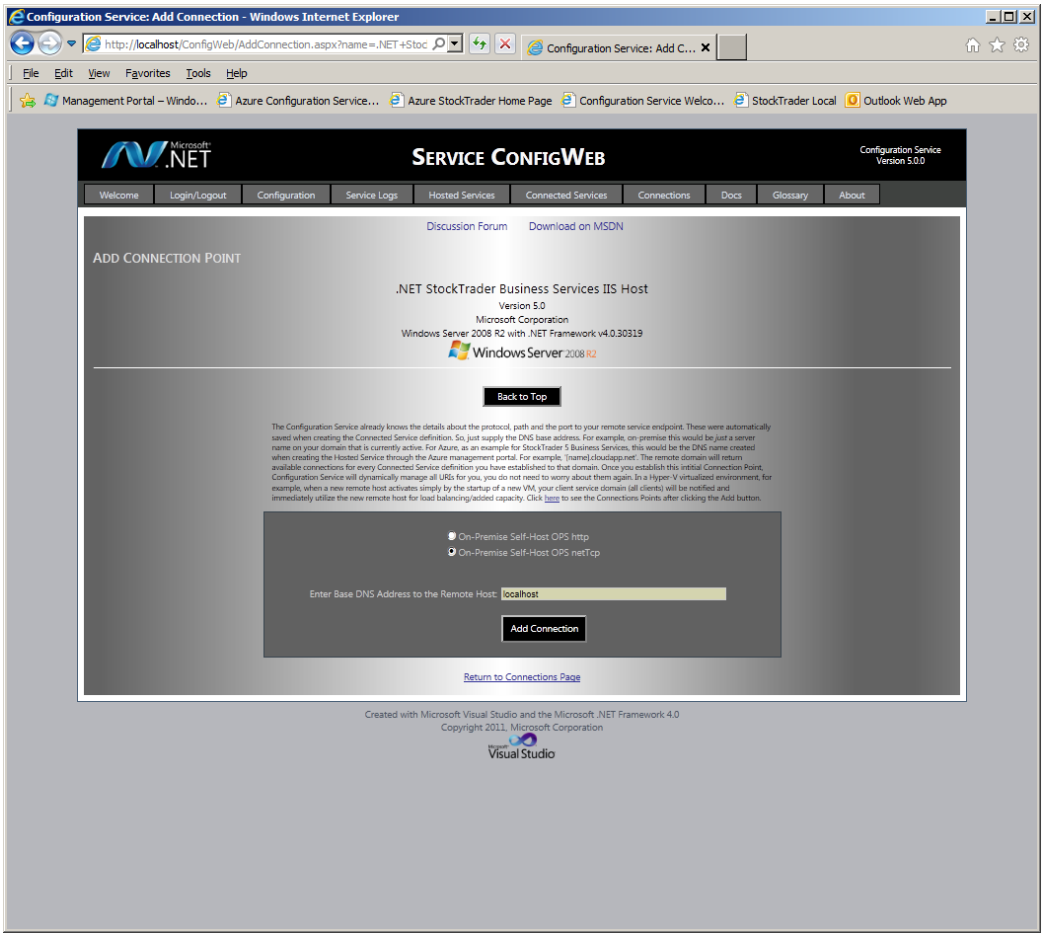
[2] Also, all executable programs are installed, pre-compiled, in [stocktrader install directory]\Builds. You can also optionally start the console host implementation, but for this example we will start the Windows host. Note this Windows host program is itself a re-usable base class that is part of the Configuration Service, and can be easily used in your own host programs without writing any Windows .NET code.



You are now at the Connections page for the Web application, but via the inter-connected nature of the configuration services, **we can directly drill-down into the .NET StockTrader Business Services IIS Host** connections from here, by clicking on the link as circled above.



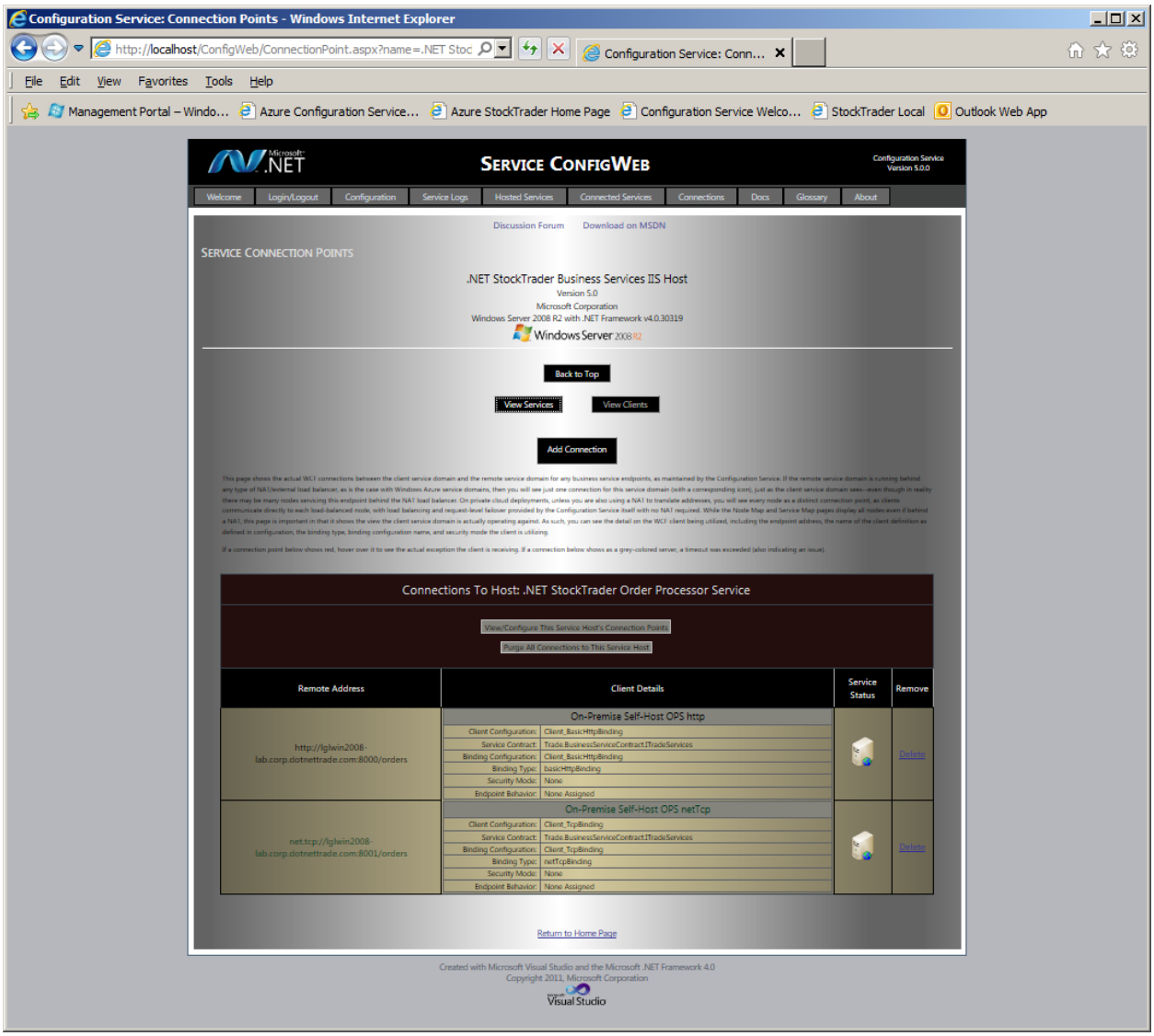
Click on the **Add Connection** button in this page, and you will see the connection options for business services to connect to the remote Order Processor Service. Choose the **On-Premise Self-Host OPS netTcp option.**



Enter the name of the server running the Order Processor Service (or **localhost** for the local computer). Again, you just need a server name here, since the Business Service IIS Host repository already has a Connected Service Definition to this endpoint that holds the logical URI information (path and port).

Click the **Add Connection** button.

You can now click on the **Return** button to go back to see the active connections. Note if you click on the Connections menu tab instead, you will be positioned back at the StockTrader Web application and will need to drill down again to see Business Services IIS Host connections. You will notice more than one connection has been returned. That is because the Order Processor Host is hosting the Order Processor Service on two different protocols simultaneously, and the Business Services IIS Host is subscribed to both types of endpoints; these Connected Service definitions are pre-set on install. In both cases, orders are not only remotely processed, but they will be placed asynchronously from business services, because of the way the Order Processor Service contract is defined (it is defined as a **one-way** contract).



Finally, click on the **Configuration** menu item to go back to the main ConfigWeb home page. Initially, you will be re-positioned with the Web Application as the top node.

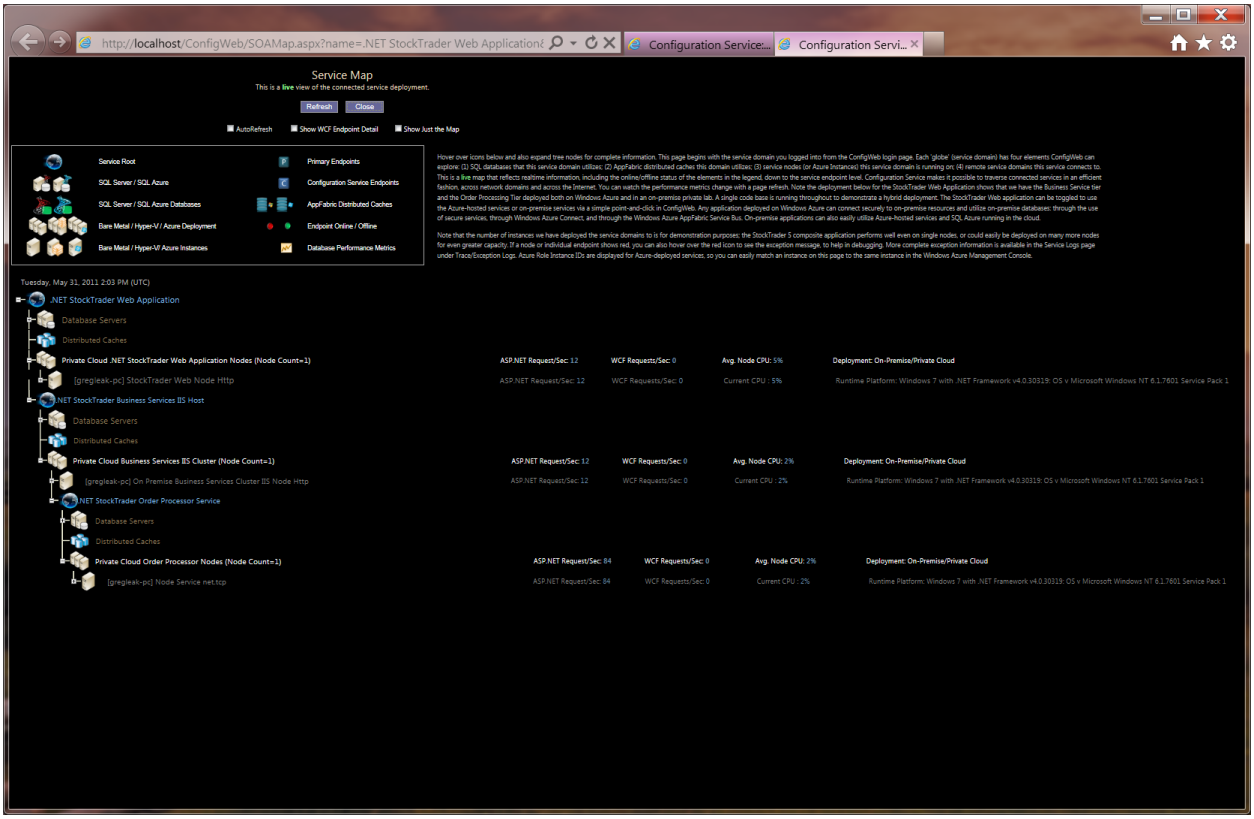
You can now select business services as the root node, by clicking where circled above. Business services will be shown on the left, with its newly added connections to the Order Processor Service.

Similarly, you could now optionally 'drill-down' one layer further, and position Order Processor Service as the root node, to configure its settings.

The Service Map View

The Service Map will display the entire physical topology of connected services making up an application, including all clustered servers each ***virtual service host*** is running on. You can bring up the Service Map by clicking on the button as shown below:

You will see the following:



You can see the three-tiered nature of the deployment. Note you can expand any node to view its individual endpoints. Also, hover over icons to see additional information the Configuration Service gathers. This view gives a near-instantaneous read on the online offline status of clustered servers and their individual service endpoints, for an entire composite application. Green icons indicate online, red, offline. If an icon shows as red, hover over it to view the exception message. If other clustered service nodes are started (for any tier in the application: Web; IIS Host Business Services; Order Processor Service), they will be populated in the view above as new clustered nodes for the virtualized service. The Configuration Service is automatically providing this clustering/load balancing.

In the diagram above, only one server node shows for each tier, since each tier is only running on one server (the same local PC). You can use **AutoRefresh** to poll every few seconds for updates automatically; and **Endpoint Detail** to see the actual URI information vs. the established "friendly names" stored/defined in the configuration repository for each host.

Back to StockTrader as a User

Now you can log into the StockTrader Web application as a user of the composite application at http://localhost/trade (vs. administrator using ConfigWeb), and try placing a buy order. This will now be processed by the remote OPS service, asynchronously.

Later in this document there are optional steps to use the MSMQ endpoint instead for placing orders with a durable, transacted message queue.

Transaction Flow If You Activate and Use the MSMQ Endpoint for Order Processing

If you choose to use the MSMQ endpoint (not active by default, see steps later in this documen), the flow of the buy order will be:

1. The Web application places a synchronous http Web-service submit to a remote Business Service Layer (IIS-hosted). This layer may be virtualized across many load-balanced servers.
2. Business Services running in IIS creates an order header in the database, but places an asynchronous submission of the order body as a message to the Order Processor Service, which will be responsible for processing the order.
3. The message is actually sent not to the Trade.OrderProcessorServiceHost.exe service host; but rather WCF automatically (by nature of an MSMQ 'loosely-coupled' binding) sends it to the MSMQ message queue on the remote server. The Order Processor Service and its associated message queue may also be virtualized across many clustered, load balanced servers, based on the dynamic clustering built into the Configuration Service.
4. WCF infrastructure will automatically notify the host program (Trade.OrderServiceHost.exe) and trigger its service contract method to process an order when it arrives in the message queue. Under heavier load, five orders will be processed at a time based on the TransactedBatching Service Behavior set on the Msmq service endpoint.
5. The order is read via a transacted queue by the Order Processor Service, and processed to the database. The user in the Web application will get a notification when the order is processed.
6. If the database part of the processing logic fails, the distributed transaction will automatically rollback and the order will not be lost, it will stay in the queue. Since the transacted queue is set to enforce ordering, it will automatically be retried n-times on a delay (based on WCF service binding settings in TraderOrderProcessorHost.exe.config); and if it still cannot be processed, it will automatically be sent to a Poison Message queue so it is not lost, but does not interrupt normal processing.
7. You can trigger such a distributed transaction test by getting a quote for symbol **ACIDBUY**, and attempting to buy the stock. While the order will be successfully submitted to the MSMQ transacted message queue; the Order Processor Service (with the acid test logic built in) will trigger a simulated exception processing the order. WCF infrastructure logic will retry the operation, but quickly trigger a fault exception as a detected poison message. At this point, the Order Processor service, with a custom service fault handler, will move the order to a custom-created **poison message queue**; perhaps for later auditing or processing (which is not implemented; for now the order will simply remain pending for the user, and stay in the poison message queue unless you implement such additional logic to process poison messages).

The Business Service Self Host Program and TCP/Binary Service Endpoints

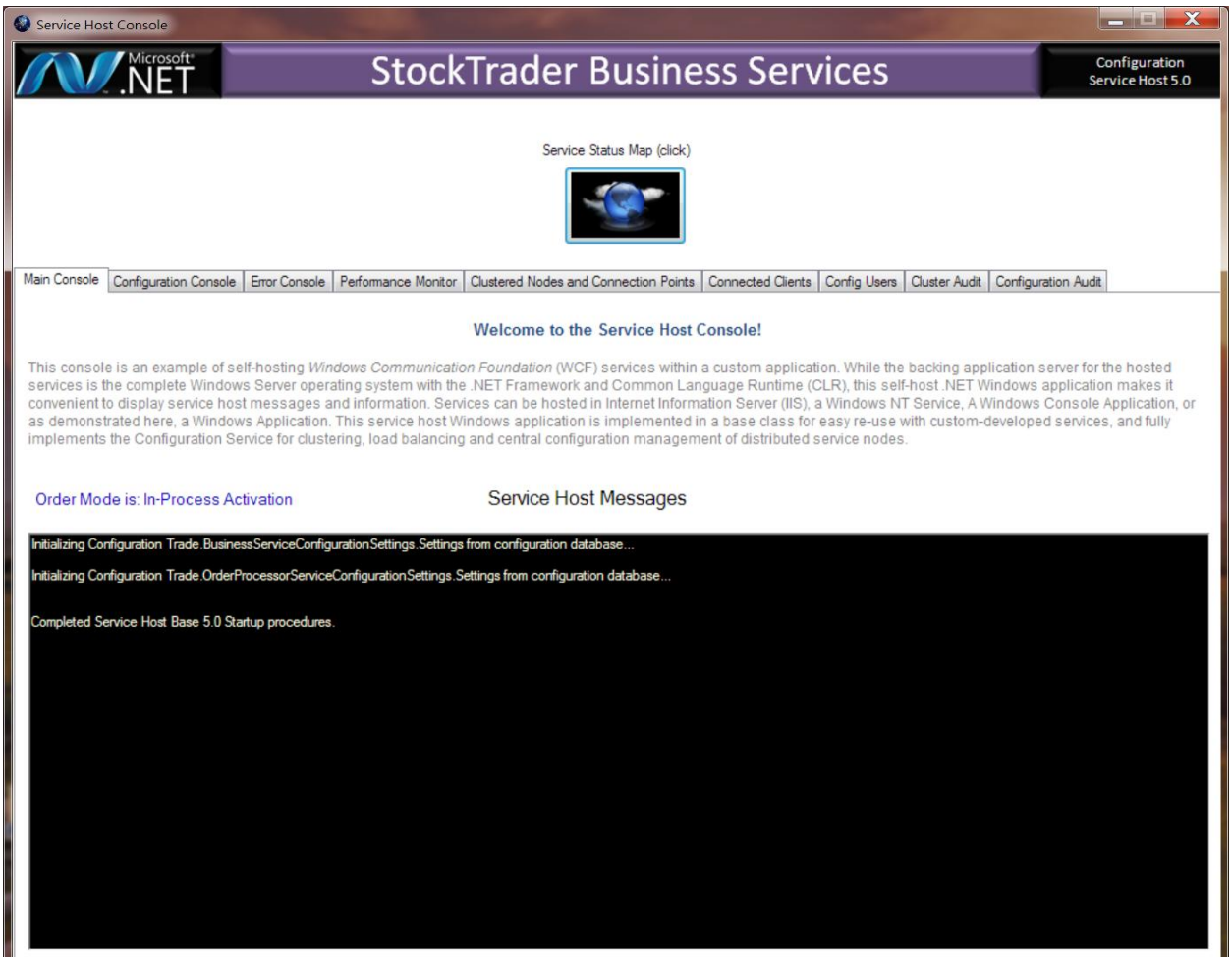
While the above instructions showed how to connect to the IIS-hosted Business Service layer from the Web application, another option is to run the business service self-host application, and connect to it instead. These are the Web application AccessMode settings named with "On-Premise Self-Host xxx":

You set these as you did for connecting to the IIS Hosted Business Services. However, as with the Order Processor Service, make sure to start the Business Service Self Host program first, from the .NET

StockTrader Windows Start menu items, under the StockTrader Self-Hosts subfolder; or you can also run from the \builds directory. There are three-self host environments available, all share the same configuration repository and only one can be active per node at a time (however, they are interchangeable and you can run different host types on different nodes if you want):

- Windows Host (inherits from base Config Service .NET Windows Forms class)
- Console Host
- NT Service Host (run \stocktrader install dir\setup\util**installBSLService.cmd** from a CMD prompt with –i to install and –u to install/uninstall as an NT Service. It will install in auto-start mode; so do not forget it will be active on re-boots unless you change this.

To get started, launch the **Trade.BusinessServiceHost.exe Windows** self-host program on your PC. Make sure to choose "Run As Administrator".



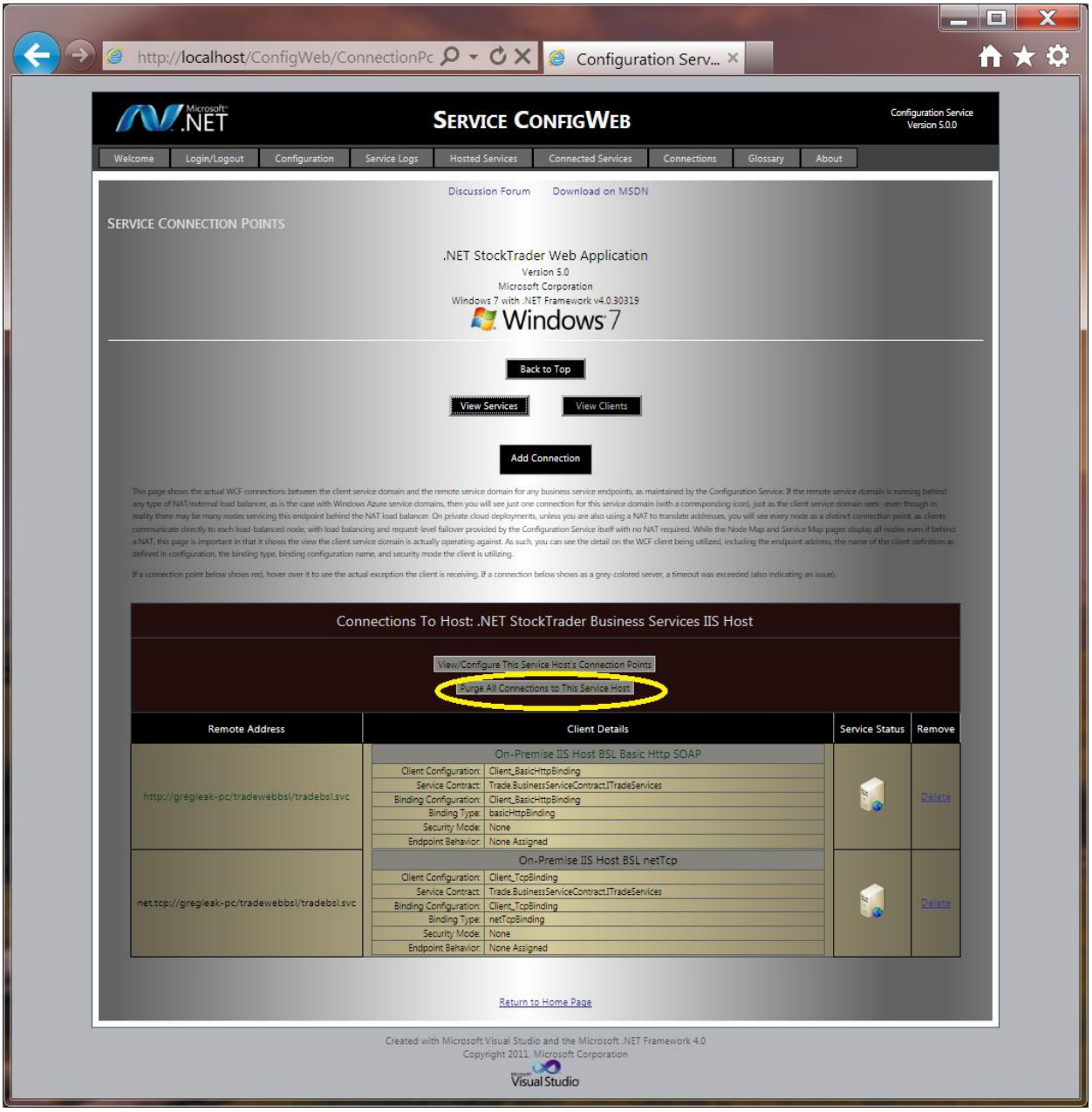
With the self-host program, you have two options: you can connect via HTTP/XML-encoding, or via TCP/Binary-encoding. TCP/Binary can offer performance advantages vs. HTTP-XML encoding; given lower serialization costs and network. Multiple clients can connect simultaneously over all the bindings a service is supporting.

Note that IIS 6.0 can only host HTTP-based bindings; however this is not a limitation with IIS 7.0, so you will see possible TCP endpoints for IIS-Hosted services with StockTrader 5.0 as well (not just for the self-host version of the BSL); however IIS-hosted Tcp/binary endpoints will only be active/working if running on IIS 7.x (Windows Vista/above or Windows Server 2008/above), as Setup deactivates these endpoints on Windows XP and Windows Server 2003 automatically since IIS 6 will not recognize them.

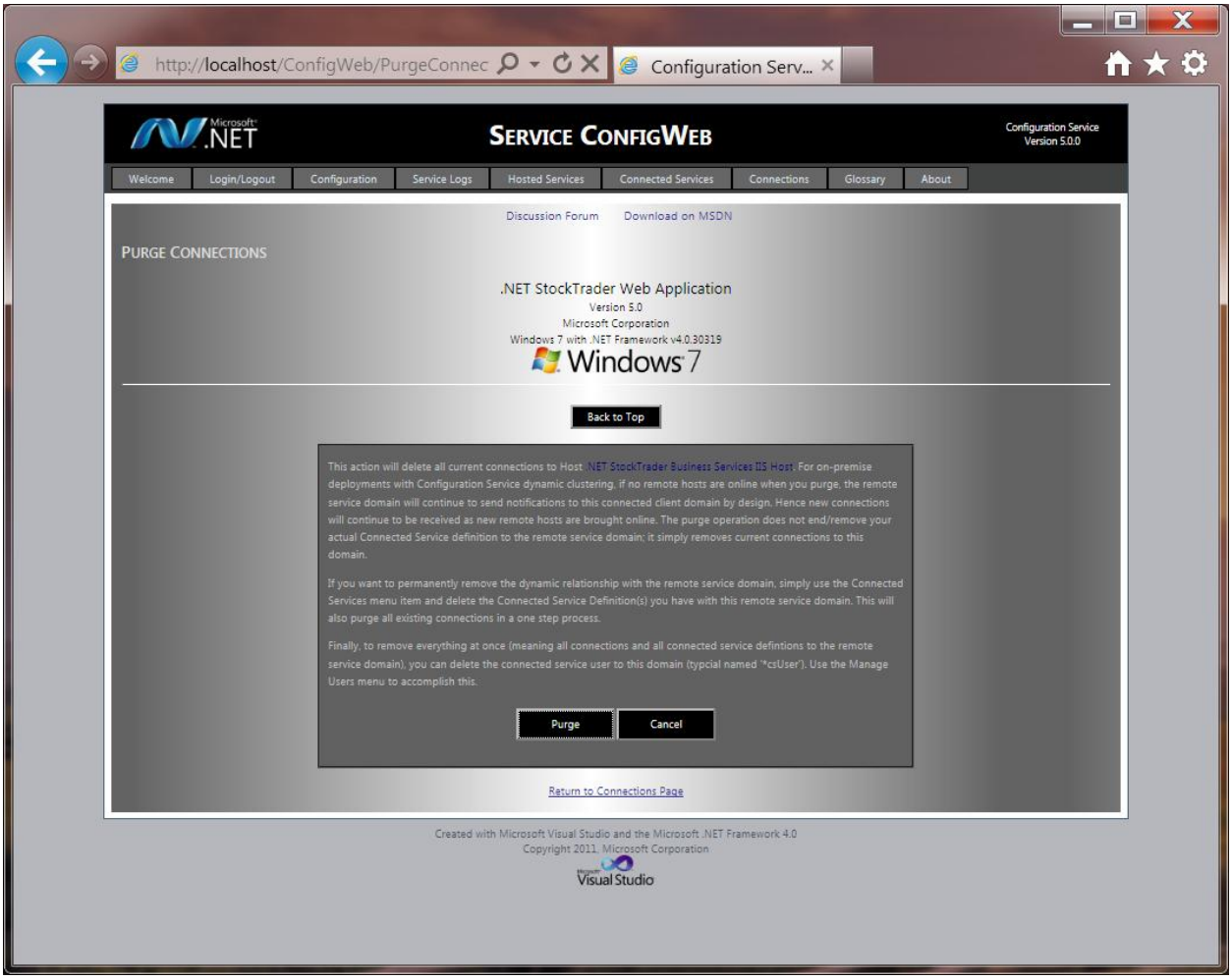
To change the StockTrader Web Application to use the self-hosted business services in the Windows host program, go back to the home page of ConfigWeb by selecting the **Configuration** menu item.

First, we will delete all connections to the IIS Host business services. This step is not required, as the Configuration Service can track/maintain relationships to any number of service hosts hosting any number of service endpoints from a client application/service. However, it will make this tutorial a bit clearer for first time users, and less likely to accidently configure the wrong remote host in later steps in this tutorial.

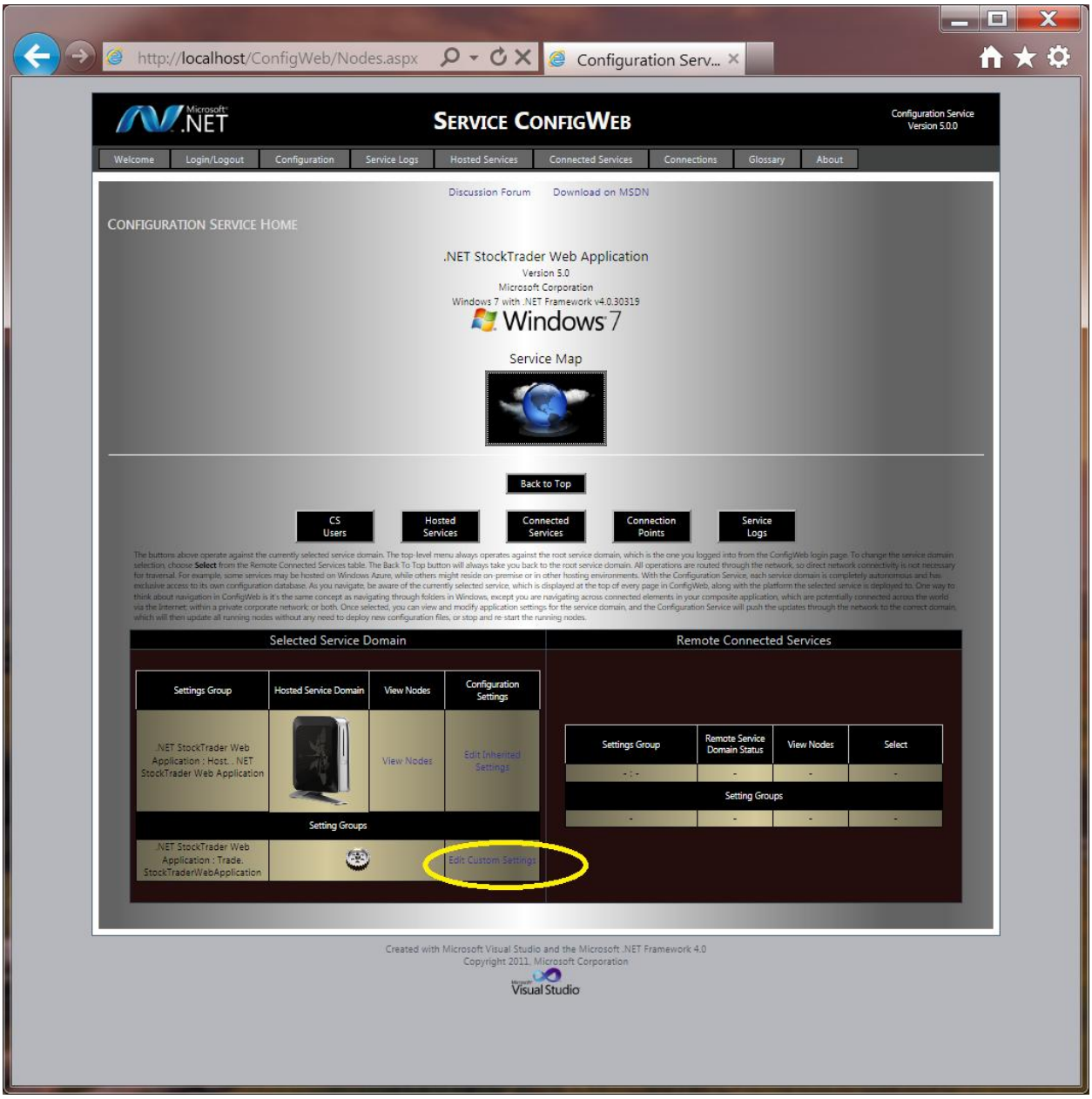
Select the **Connections** menu item.



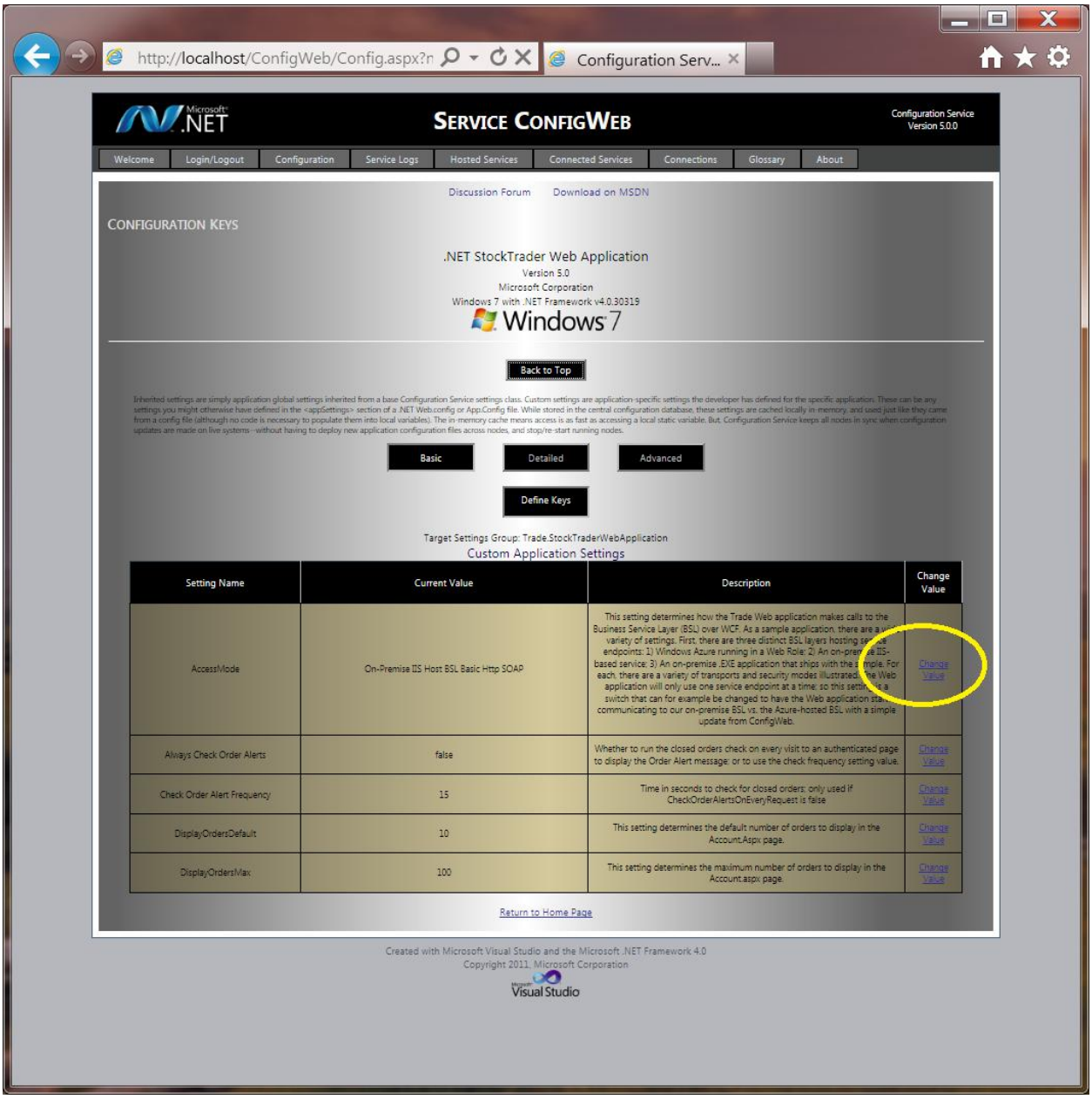
Now select **Purge All Connections to This Service Host**.



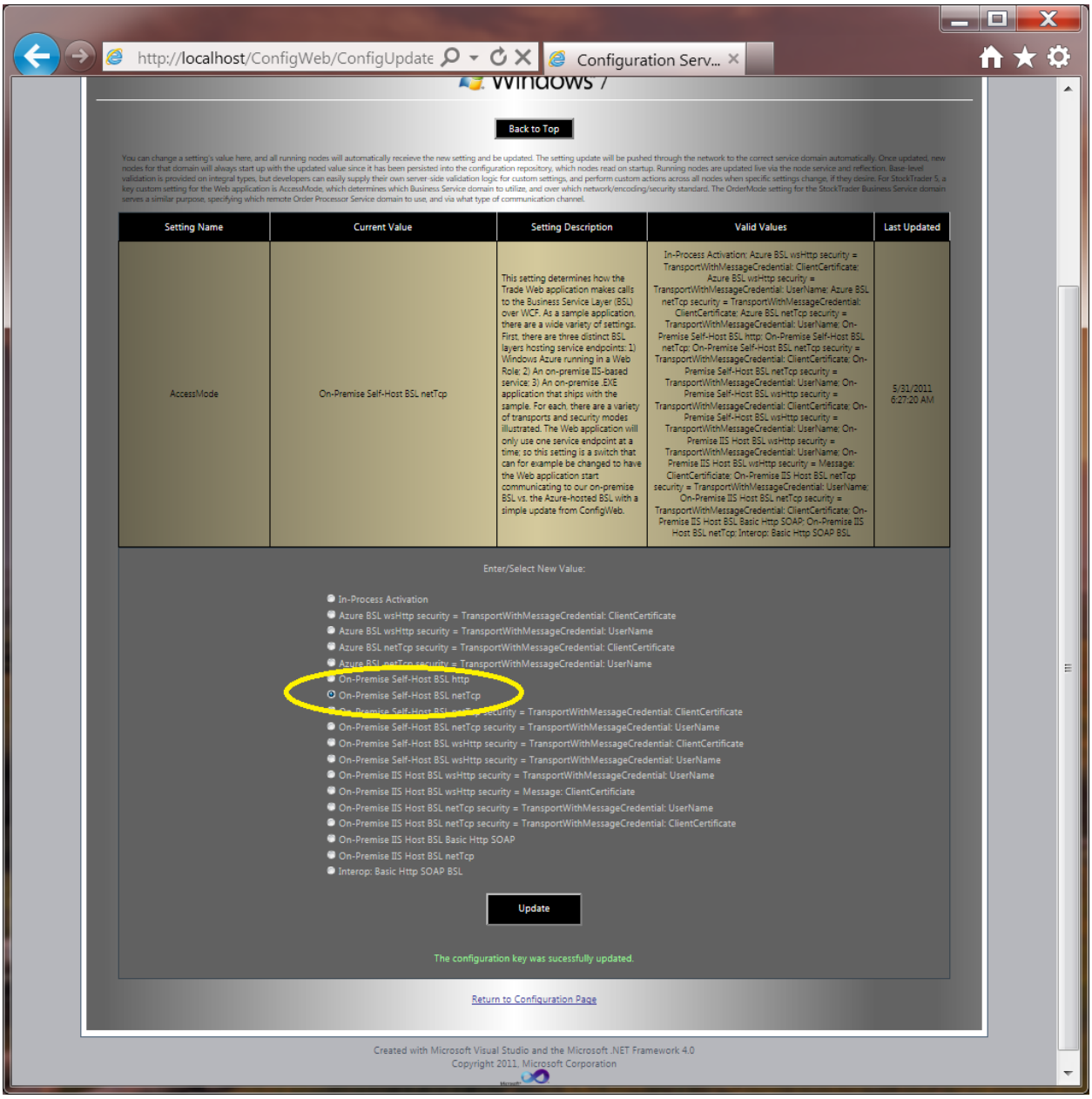
Click on the **Purge** button. After the connections have been purged, click on the main **Configuration** menu item to go back to the ConfigWeb home page:



From here, we will now select **Edit Configuration** above to change the StockTrader Web application AccessMode yet again.



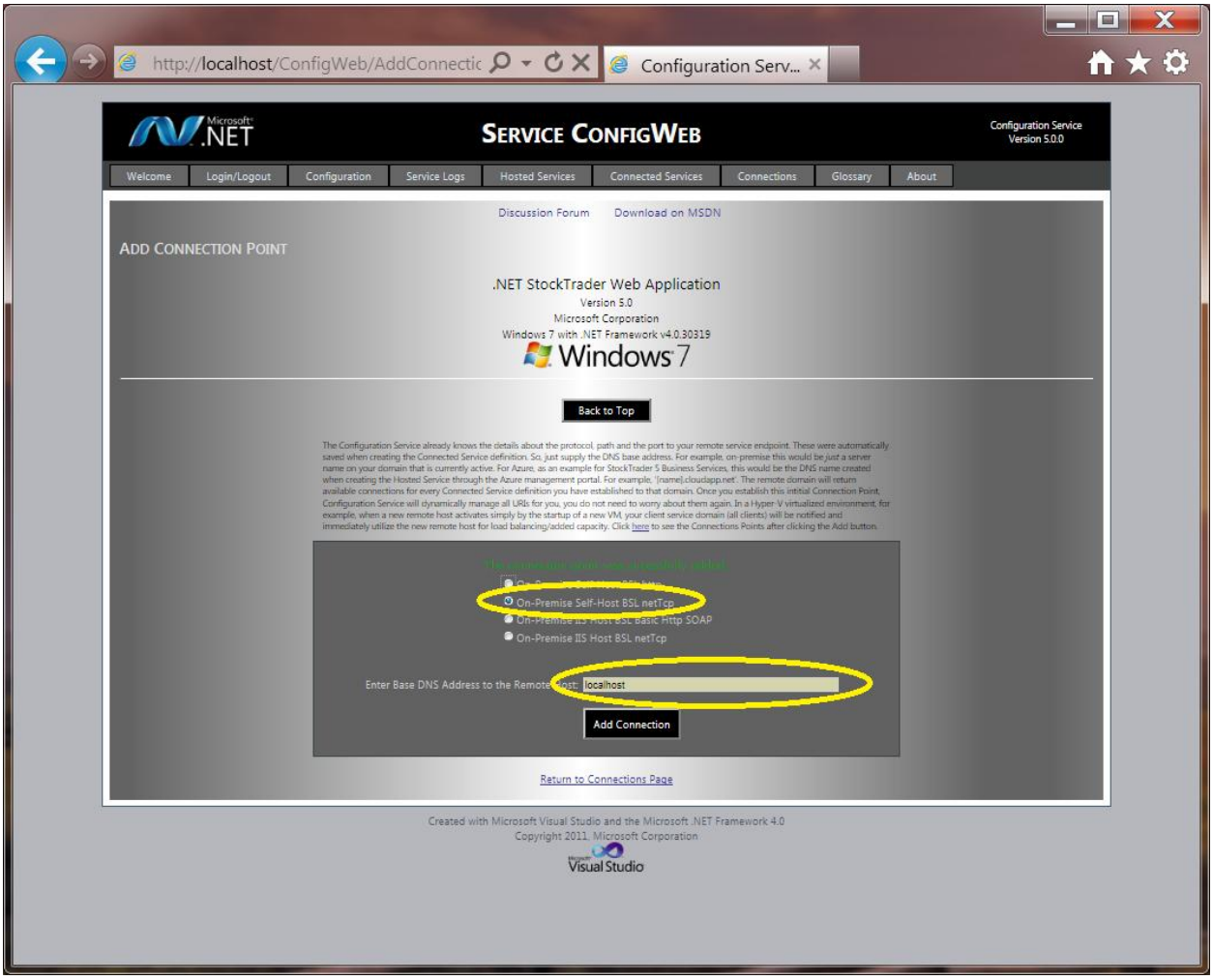
Select **Change Value** as indicated above.



Select **Tcp_On-Premise Self-host BSL netTcp** and click **Update**. This equates to activating remote business services via TCP/Binary encoding vs. Http with XML encoding. Next, we need to establish an initial connection to the running instance of the self-hosted business service program, Trade.BusinessServiceHost.exe.

First, go back to the home page for ConfigWeb by selecting the **Configuration** menu item. Now choose the **Connections** menu item again:

Select Add Connection, and then choose the .**NET StockTrader Business Services Tcp/Binary** endpoint**.**



Make sure to type in the server name hosting the running Business Service self host program, and click Add Connection. Next, click on the main **Configuration** menu item.

As with IIS hosted services, we now see the self-hosted business services being depicted in the right hand pane.

You can log into the http://localhost/trade StockTrader Web application as a site user. The StockTrader Web application is again using a remote service tier, but this time is connected to a different physical host process- the Windows self-host program.

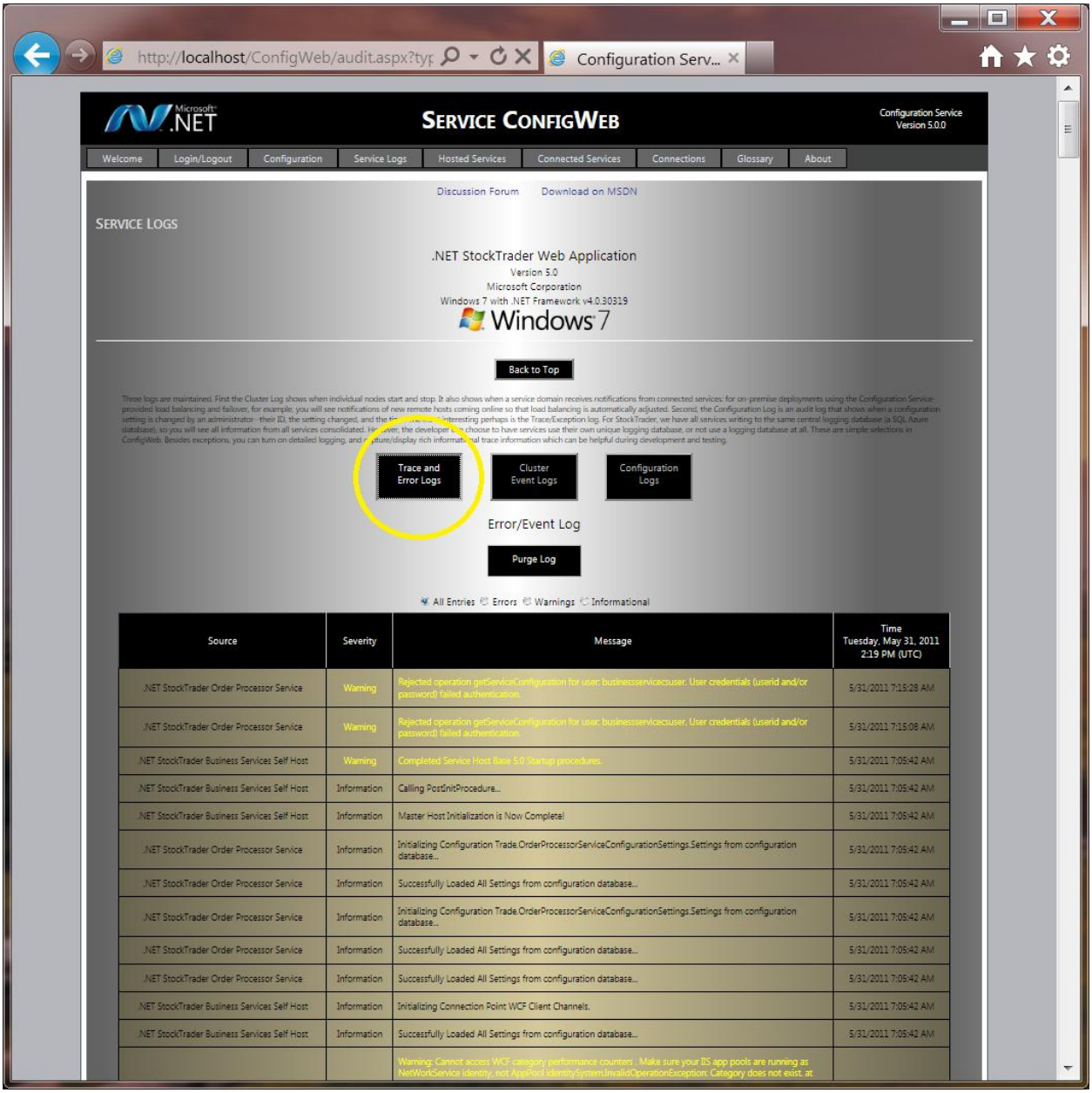
Different Hosts—Different Configuration Repositories

It is very important to keep in mind that the configuration repositories **are specific to the service host** hosting the services. Hence, while Business Services are hosted in both IIS and in the separate self-host

Trade.BusinessServiceHost.exe program, each maintains its own repository. Making changes to settings for one service host will not affect the other service host—they are physically distinct. When you are running business services in-process (AccessMode=InProcess) with the Web application, and you edit the business services configuration directly, you will be effecting the IIS-Hosted Business Service Repository only, based on the way the in-process mode is initially configured. Hence, if you now change your AccessMode to Http_WebService or Tcp_WebService from the Web application, the Trade.BusinessService.exe self host program will not yet be configured for **Msmq OrderMode**. You can optionally do so, or leave OrderMode as **InProcess**. If you change it, make sure you also follow the procedure as before to add an initial Connection Point from the Business Services self-host program to the Order Processor Service.

Monitoring Exception and Trace Logs

Configuration 5.0 allows any application or service to log exception and/or trace information to a logging database. You can then use ConfigWeb to view these logs.



Please note that information is also sent to each node's Event Log as setup in the ConfigWeb as an application-level setting. The log is very important for understanding issues, including exceptions your application might have. For WCF services, a Fault Behavior is automatically applied, so you do not actually have to write code to log exceptions, it all happens automatically. However, you can also write to the log using the ConfigurationUtility writeConfigConsoleMessage method.

You can view rich tracing information for Configuration Service 5.0 by turning on Detailed Logging and/or Log Node Notifications. This is a setting for each service domain in ConfigWeb. For performance reasons, you should keep detailed logging off for production applications.

Azure StockTrader: Re-Configuring for Remote Modes

By default, just like the on-premise version of .NET StockTrader, the Azure StockTrader Web Application will be operating as a standalone ASP.NET Web application, and not using the remote modes. The steps to reconfigure for remote modes are **almost** the same as those above for on-premise .NET StockTrader, but with a couple of additional steps. For Azure StockTrader, the default install does not create Connected Service Definitions to Business Services or the Order Processor Service. So, you will need to do this first before adding connections from the Web application to the Business Services; and from the Business Services to the Order Processor Service. Here is how.

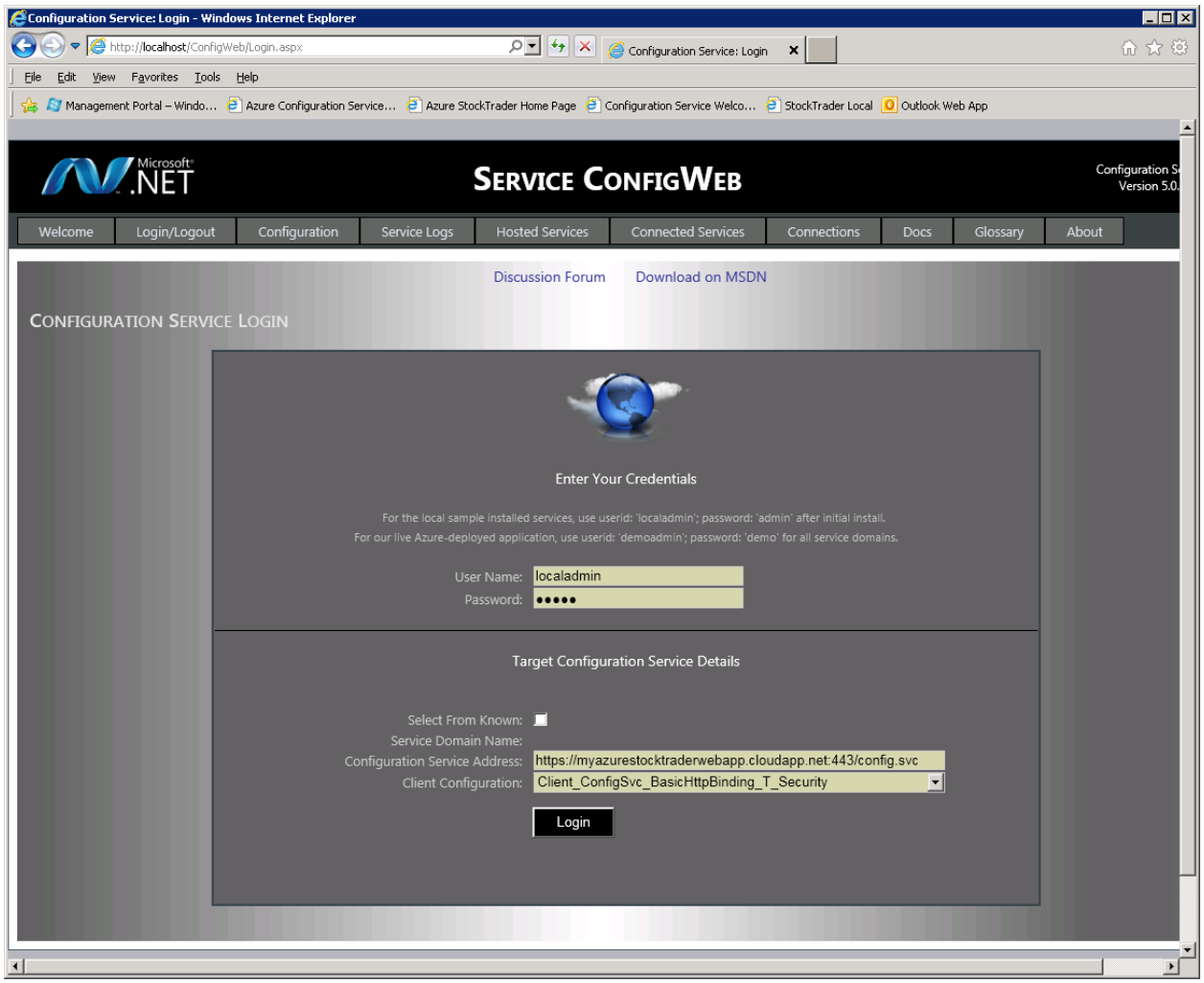
- Compile and deploy the three Visual Azure StockTrader Solutions per the AzureReadme file in the Windows Start menu. Make sure you deploy each to the correct Hosted Service via the Windows Azure Management Console.

- Bring up ConfigWeb. You can either use your own on-premise ConfigWeb for these steps (http://localhost/configweb), deploy AzureConfigWeb to your own Azure subscription as a Hosted Service, or, for convenience, use our Azure ConfigWeb at: https://azureconfigweb.cloudapp.net.

Note

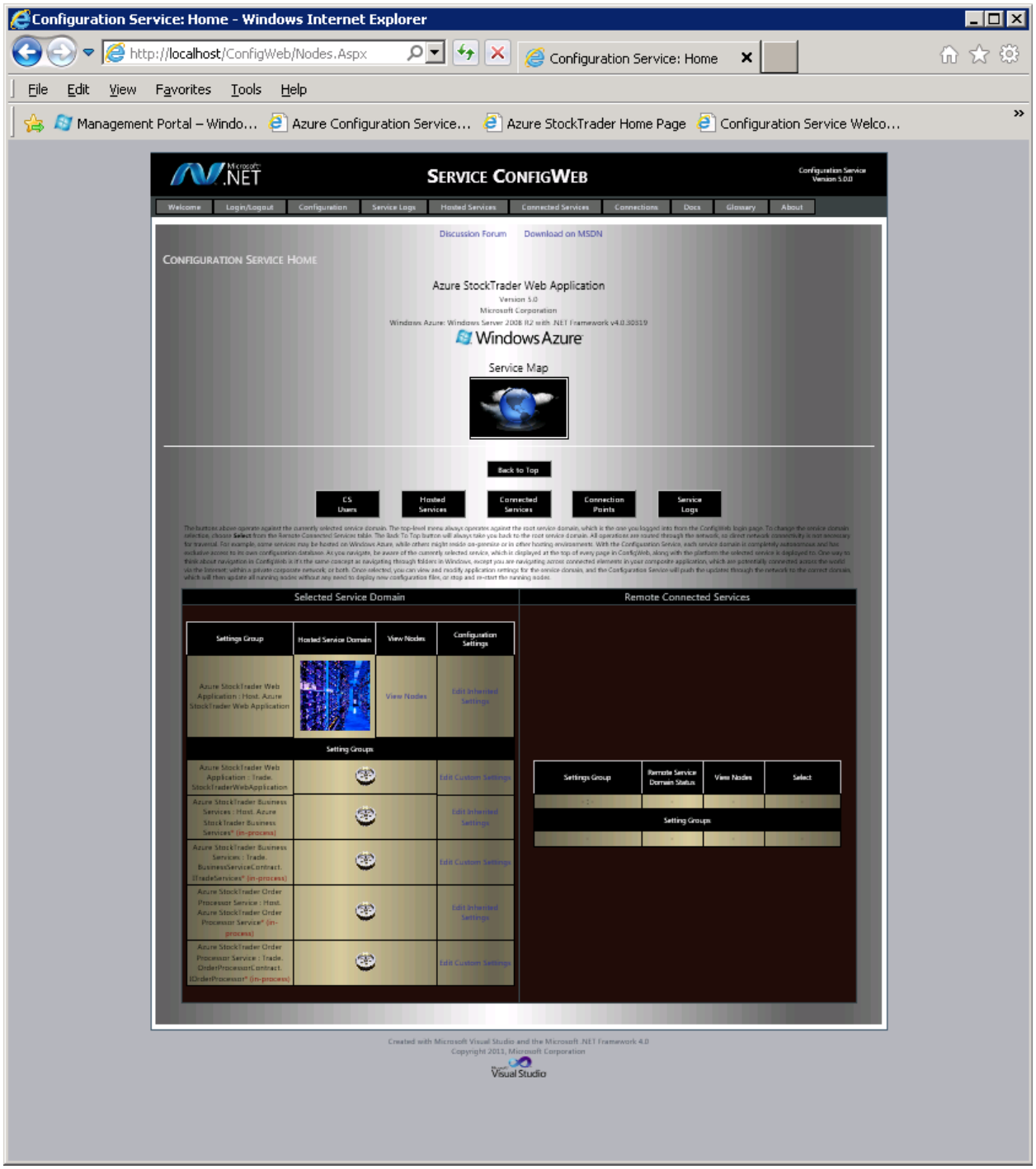
The login page for Azure ConfigWeb (as provided in your download and also as used at the link above) has several preset logins that are selectable. These Azure addresses point to OUR (Microsoft's) hosted StockTrader application service domains. As you complete the following sections, you will want to make sure to enter the addresses to YOUR StockTrader Hosted Services. You can edit the Web.Config of the AzureConfigWeb Visual Studio project (and/or your on-premise ConfigWeb application), to change the default addresses that appear in this drop-down list, so they point to your service domains for the various elements of your Azure StockTrader deployment vs. to our hosted services. Then you will not have to enter addresses on every ConfigWeb login, or accidently login to our services (which will only let you in as a demo administrator with no change privileges). So, now might be a good time to do this, using the base DNS addresses of your Azure service domains for the StockTrader Web Application, StockTrader Business Services and Order Processor Service (the three separate Azure Hosted Services/deployments). Near the top of Web.Config you will see the list, and you can substitute your DNS addresses for ours for each of the three services. The ports and the client configurations will remain the same. Change the information carefully.

- Login to the Azure StockTrader Web application. Deselect "Select from Known" on the login page and make sure to enter the address to YOUR StockTrader hosted service as follows. Note you can always use the Azure Management Portal to find the base DNS address for any of your Hosted Services if you forgot the address.



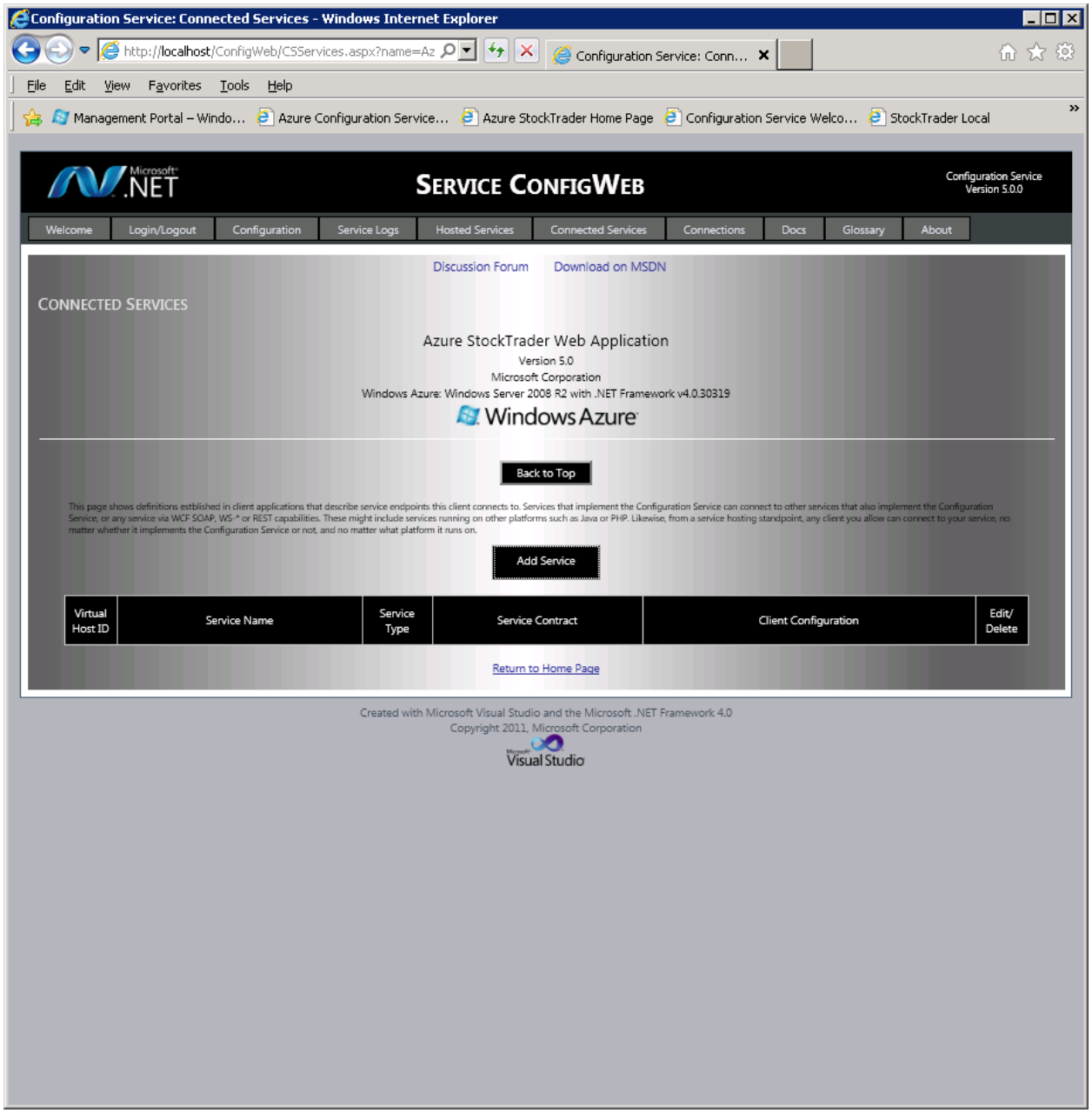
- Above, you will replace '**myazurestocktraderwebapp.cloudapp.net**' with the address you created when you first created your Hosted Service (in the Windows Azure Management Console) via the steps in the AzureReadme.html. You have a unique base DNS address for your own site that is different than the one used above.

- Make sure you go over https, and select the **Client_ConfigSvc_BasicHttpBinding_T_Security** Client Configuration.

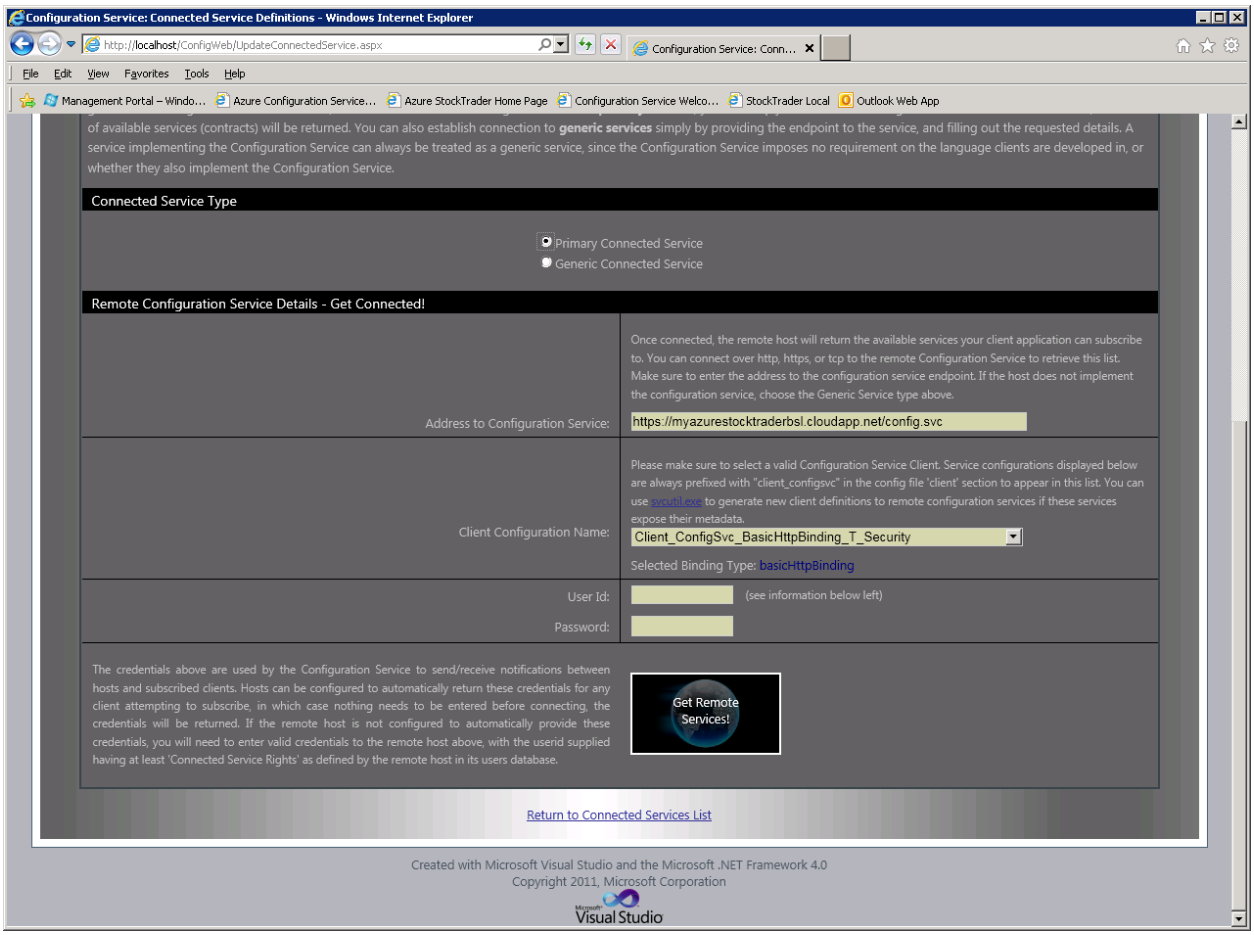


From here, you will notice that the application is running in Collapsed “in-process” mode, as a standalone Web application. But, you cannot use Add Connection Point just yet, this time (unlike for the on-premise steps before), you will first need to create initial Connected Service Definitions via ConfigWeb.

- Click on the Connected Services button in ConfigWeb.



- Next, click **Add Service**.



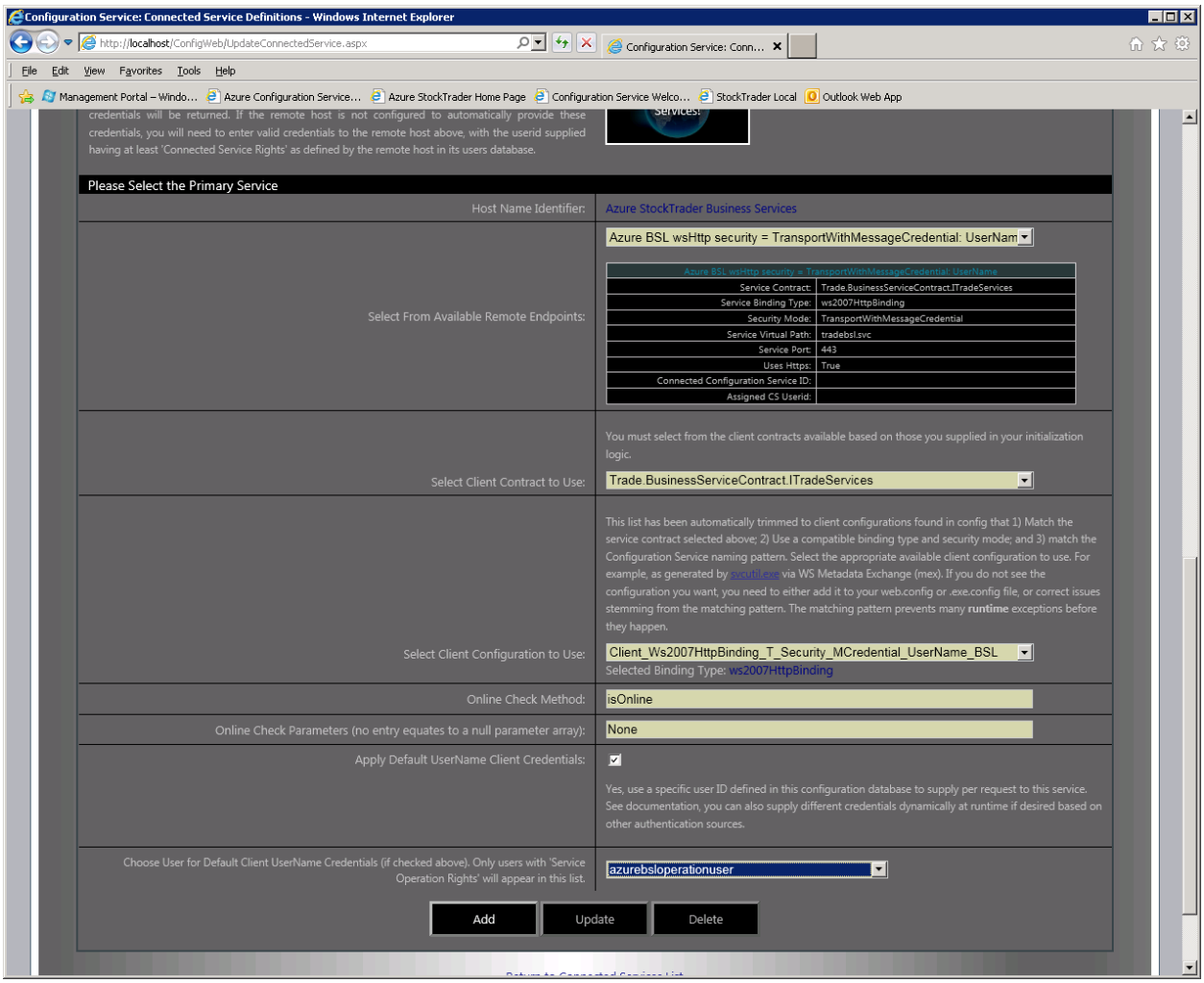
- Now, you will enter the address to your Azure StockTrader Business Services configuration service endpoint. This will be:

  https://[myazurestocktraderbsl.cloudapp.net]/config.svc

But you will replace [myazurestocktraderbsl.cloudapp.net] with the base DNS address you created for your Azure StockTrader Business Service Hosted Service in the Windows Azure Management portal.

- Make sure you select the https/secure Client Configuration, or you will not be able to connect. Select **Client_ConfigSvc_BasicHttpBinding_T_Security** (the 'T' stands for *transport* security, just a naming convention Configuration Service uses).

- Click **Get Remote Services!**



- Look carefully at the entries above. You will be establishing a Connected Service Definition to a BSL endpoint that operates with TransportWithMessageCredential security, using a username/password for authentication to the remote services. Any client not supplying this, will be rejected automatically by the StockTrader BSL. The StockTrader Web application is configured to supply a single username/password combination, much like an ADO.NET connection string works for an application.

Note

Later you can explore using ConfigWeb to change the endpoint to be the currently de-activated endpoint using x.509 client certificates for client credentials, instead of username/password. Both are just as secure, as long as going over https. These steps are documented later in this document.

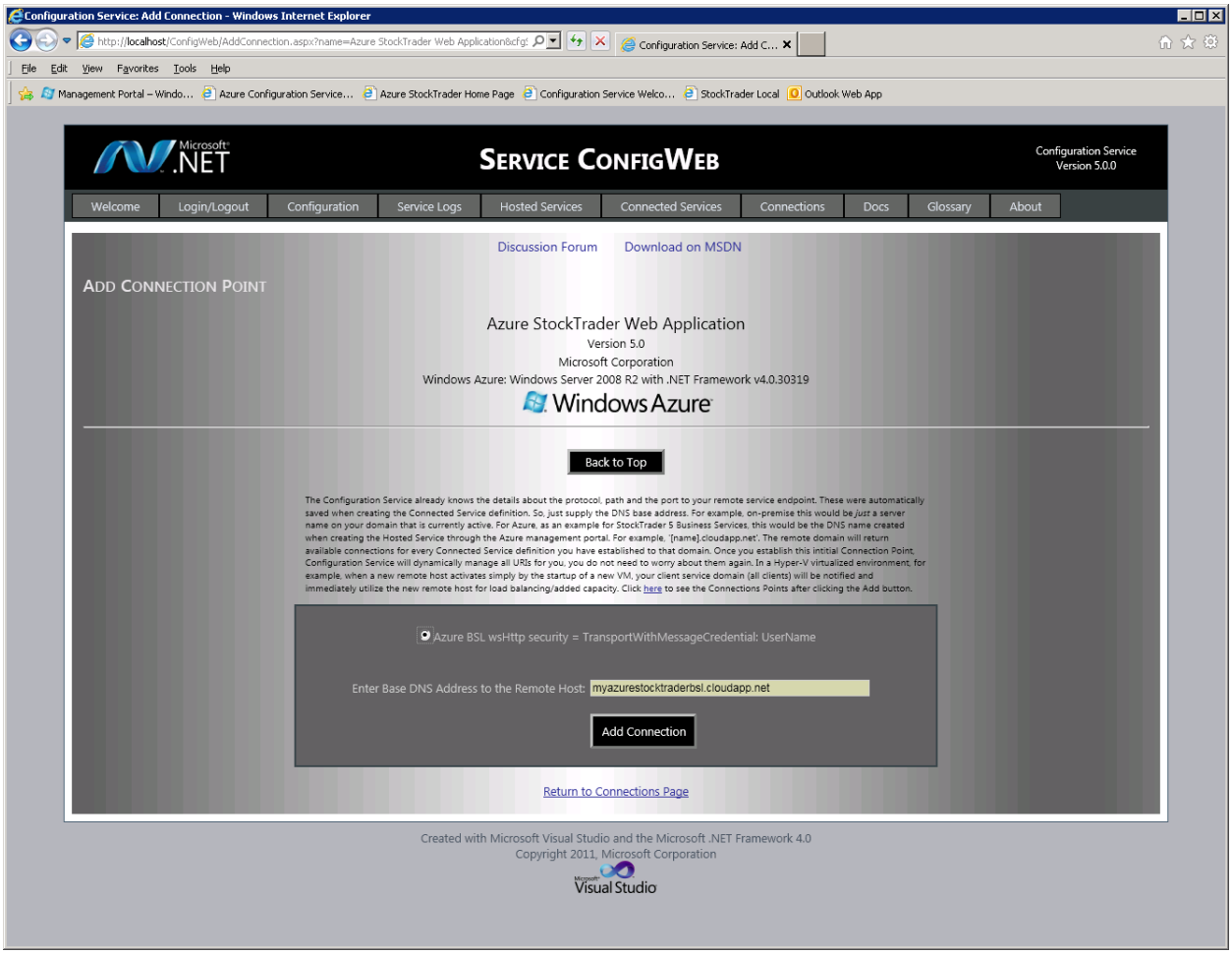
- The configuration repository for the StockTrader Web application is pre-loaded with a User definition for the StockTrader BSL with the correct user id and password. So you just need to check off Apply Default UserName Client Credentials as shown above.

- Next, choose the azurebsloperationuser from the dropdown list.
- Now simply click the **Add** button.

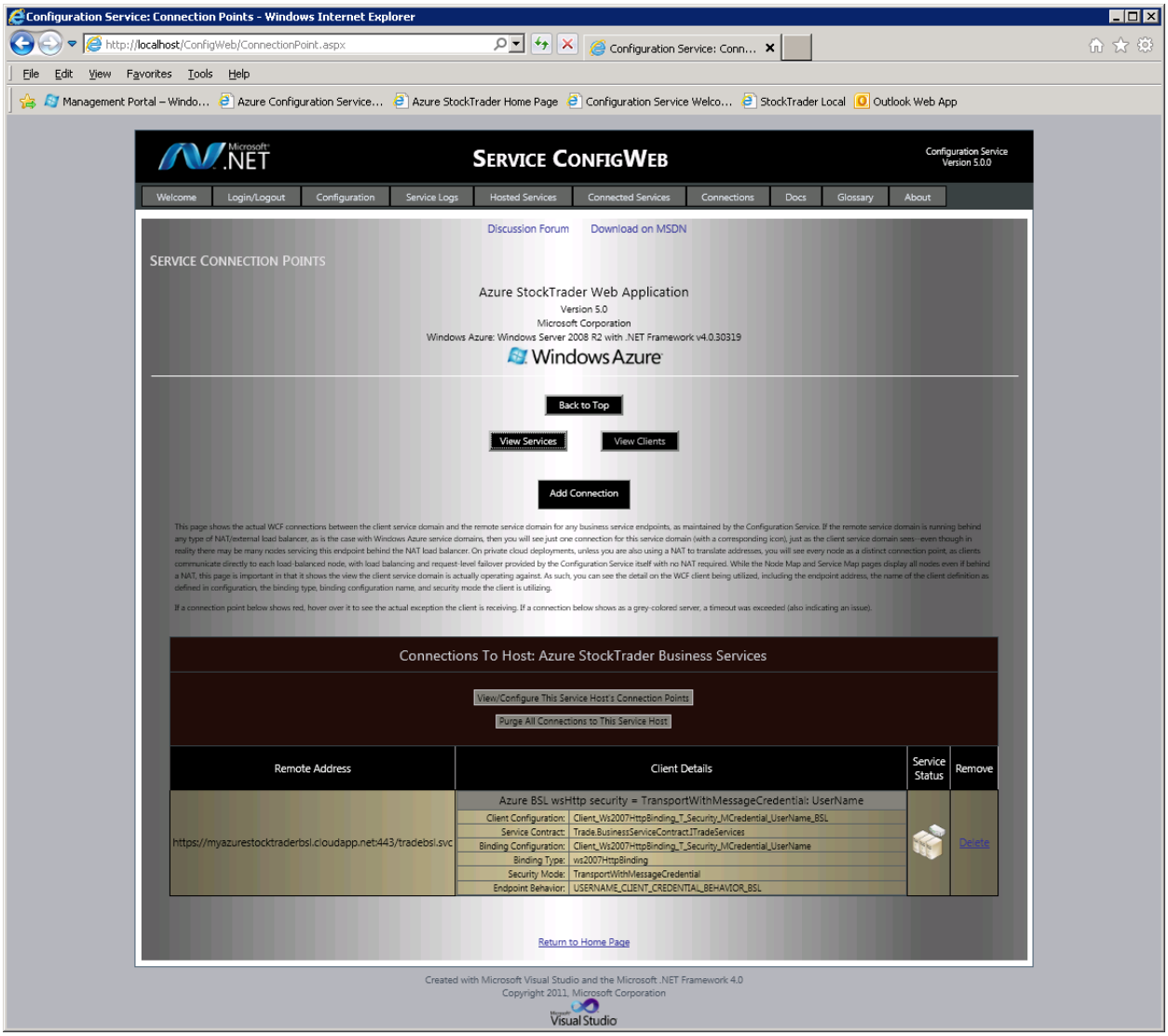
- Next, click on the **Connections** top-level menu item at the top of the current ConfigWeb page.

- From here click the **Add Connection** button.

- You will then see the following:

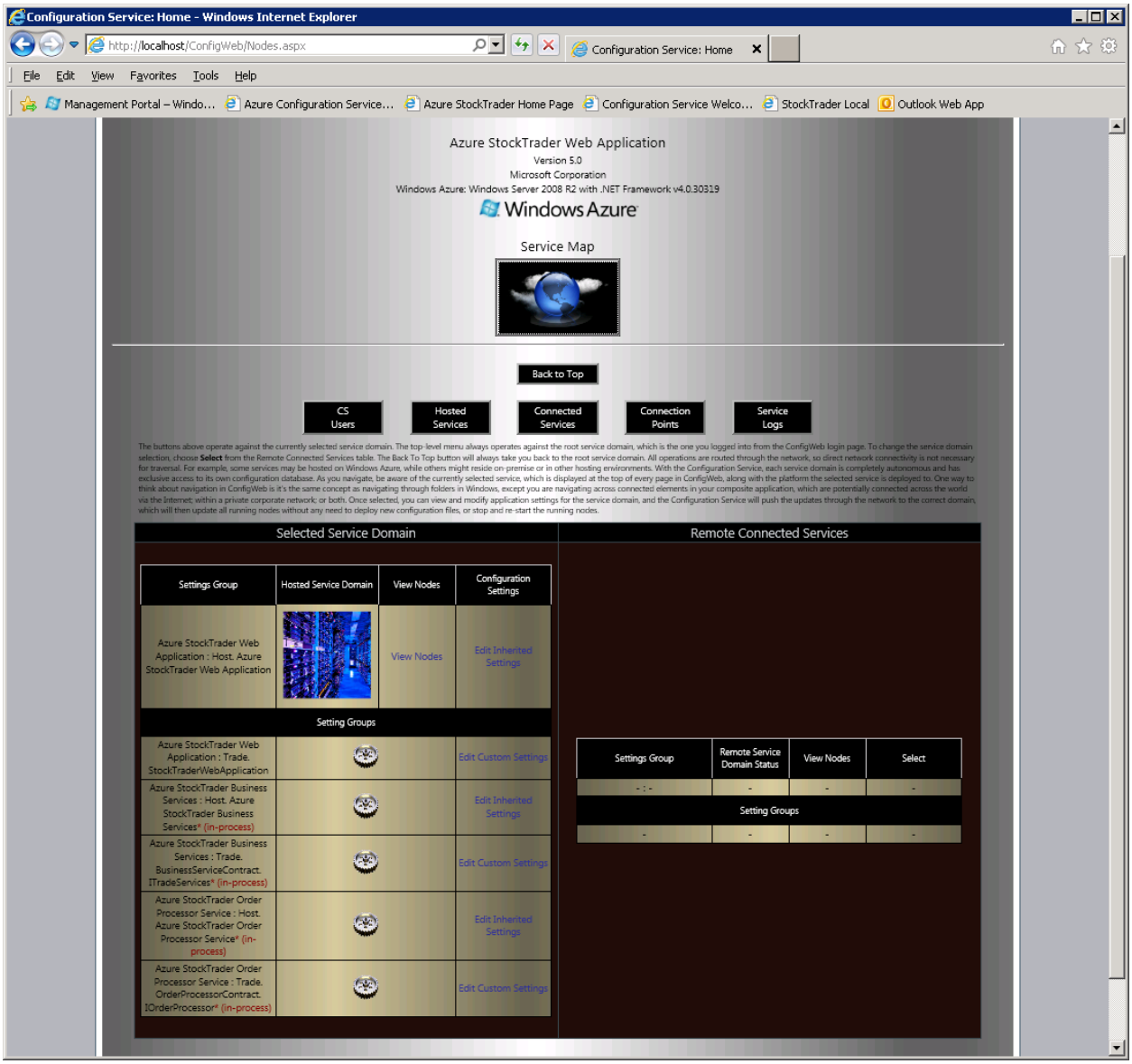


- Note that YOUR DNS address is already supplied, since the Configuration Service now already knows it. So you do not have to do anything except click **Add Connection**; the address in the textbox is already correct.
- After the Connection Point is added, click on the **Connections** top-level menu item to see the following page:

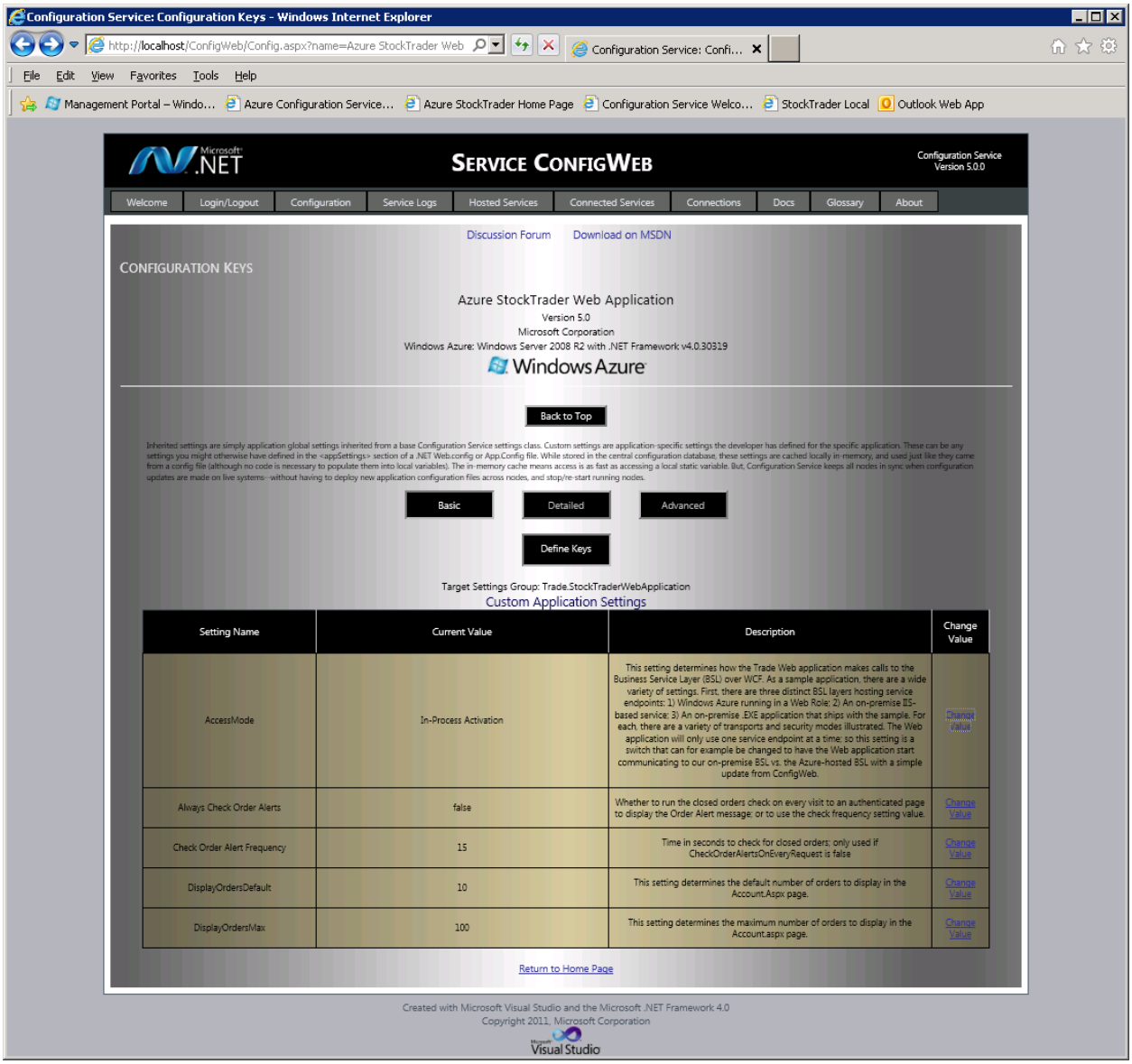


You will note the text with the friendly name of the service is in black. For StockTrader, with multiple alternate selectable service endpoints with different security modes/transports to connect to, a Connection Point currently in use will be green. This is tied to the AccessMode setting for the StockTrader Web application, and the OrderMode setting for the StockTrader Business Services. So, we need to change the AccessMode for the Web app to use this remote service, instead of running in standalone 'collapsed' mode. Now, you have already done this before for the on-premise application in the first part of the walkthrough. The steps are **exactly** the same, as below:

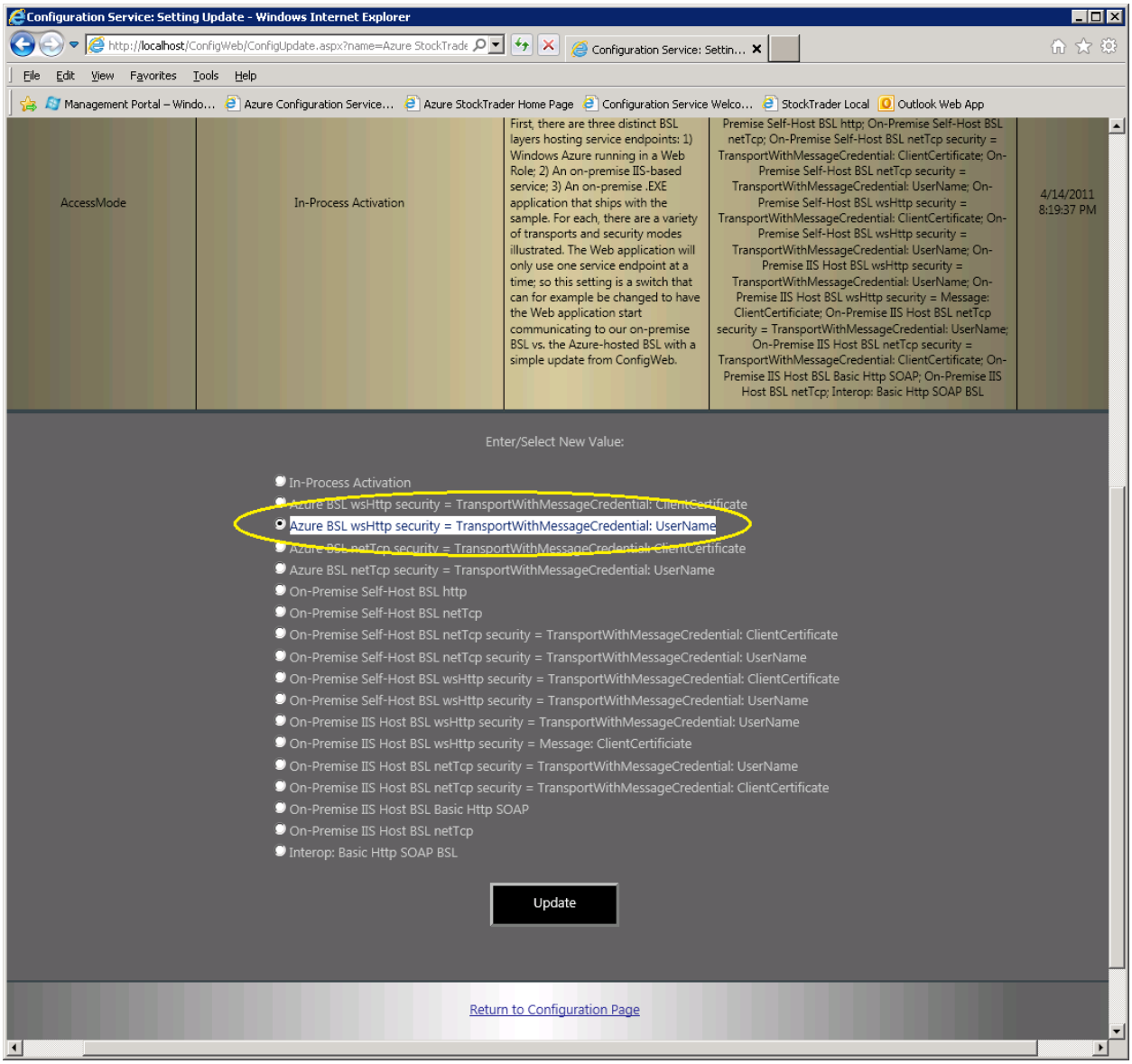
- Click on the **Configuration** menu item to go back to the home page of ConfigWeb.



- Click on **Edit Custom Settings** (the second link for the Azure StockTrader Web Application).



- Select the **Access Mode** Setting by clicking on **Change Value**.



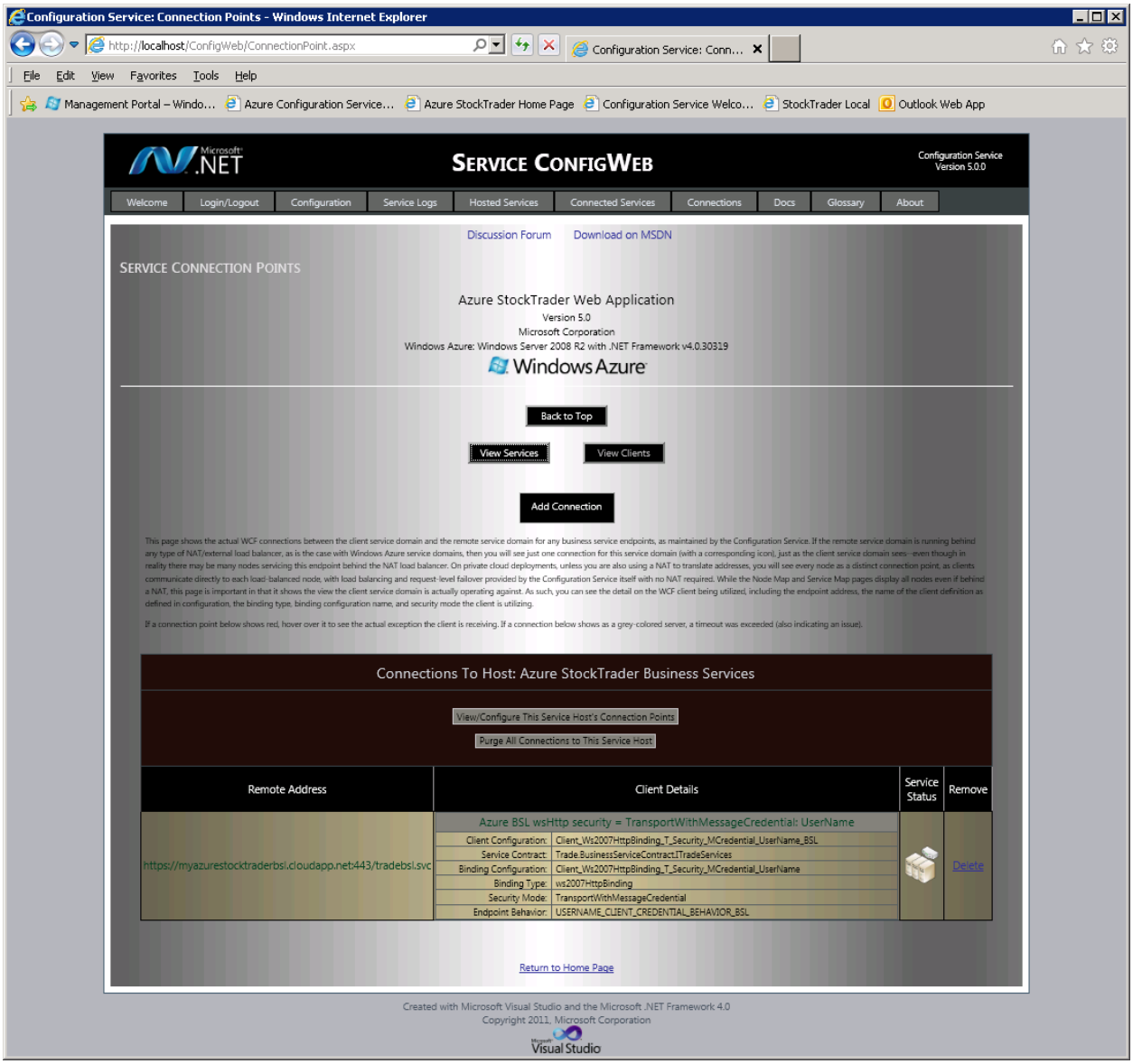
- You will select the **AccessMode** as shown above:

  Azure BSL wsHttp security = TransportWithMessageCredential: UserName

Note

Remember that most of the endpoints shown above are not active by default, but they are loaded into the configuration repositories so they can be activated later optionally, and the settings (AccessModes) pre-configured. And not all endpoints can be hosted simultaneously by the same service host at the same time: for example, if the security settings for the same protocol are different, one endpoint might have to be de-activated first before activating the desired endpoint; and, a different Service Behavior applied to the service host hosting the endpoint. All of this can be accomplished via ConfigWeb without programming, or redeployment, as shown later.

- Click the **Update** button.
- Go back to the **Connections** Page:

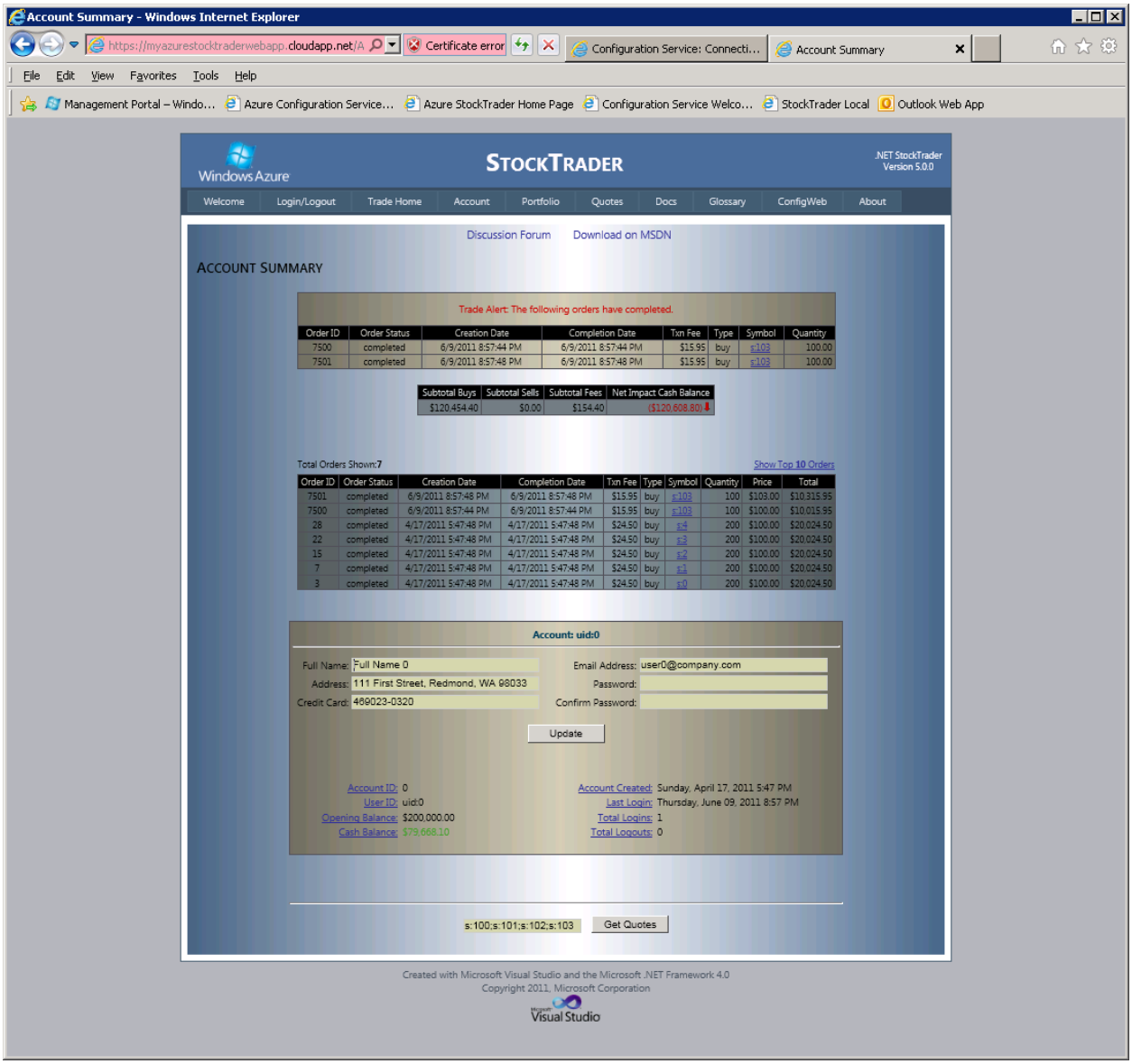


- You will notice the name is now showing in **green** text; meaning this connection is the active connection being used based on the AccessMode setting. Remember, you might have connections to many different hosts, and different service endpoints for each host. ConfigWeb/Configuration Service can handle this, ala StockTrader. There is no limitation to the

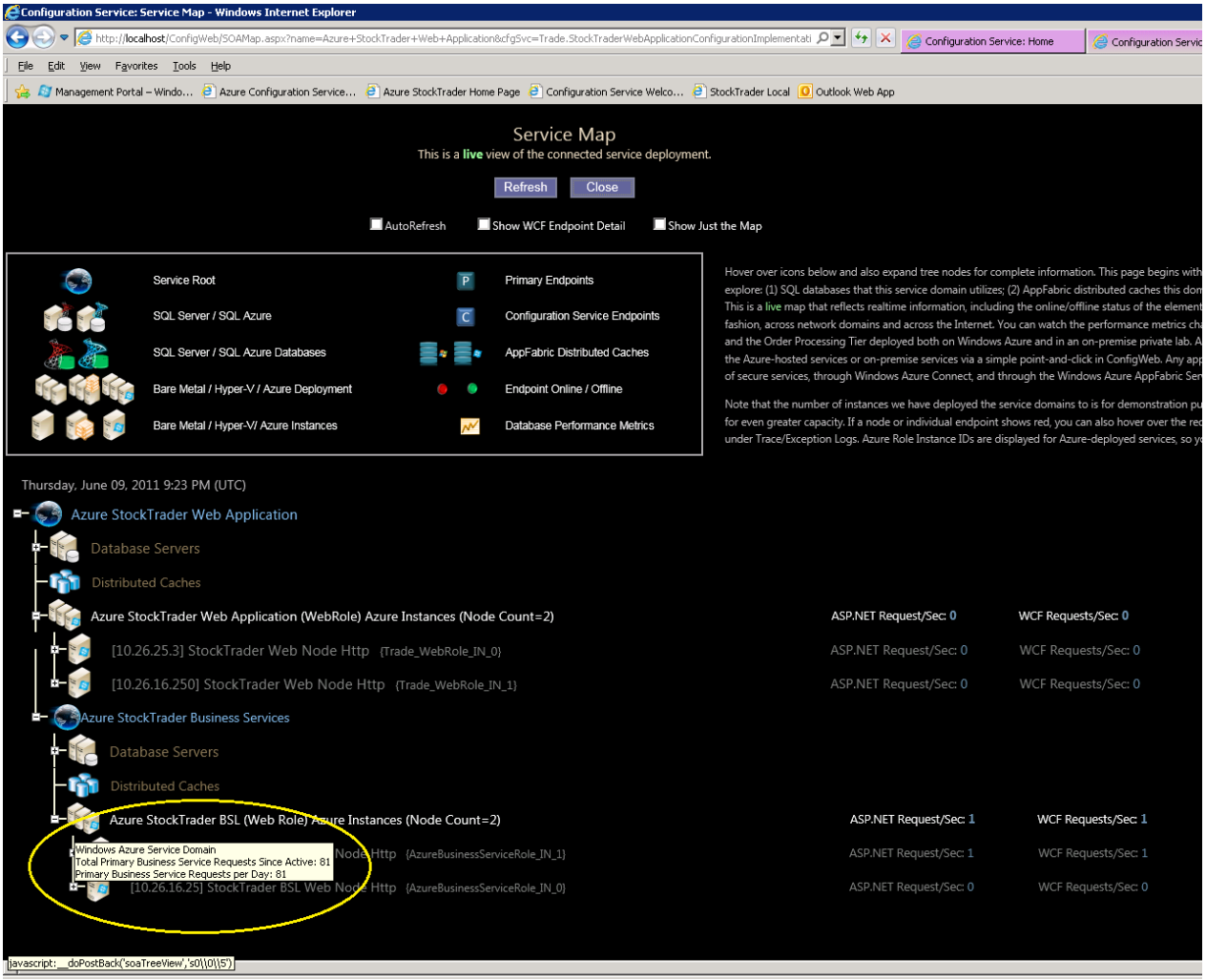
number of services a client can subscribe to, etc.

- If you use the StockTrader Web application as a user now, and login and visit some pages, you will be going remote to the StockTrader Business Services tier (the user never knows).

- Login to the StockTrader application as a user (perhaps as user uid:0), and click around a few pages. Place two orders as buy orders.



- Now, as the Administrator, goto the Service Map for Azure StockTrader Web Application, and hover over the BSL service domain as shown below:

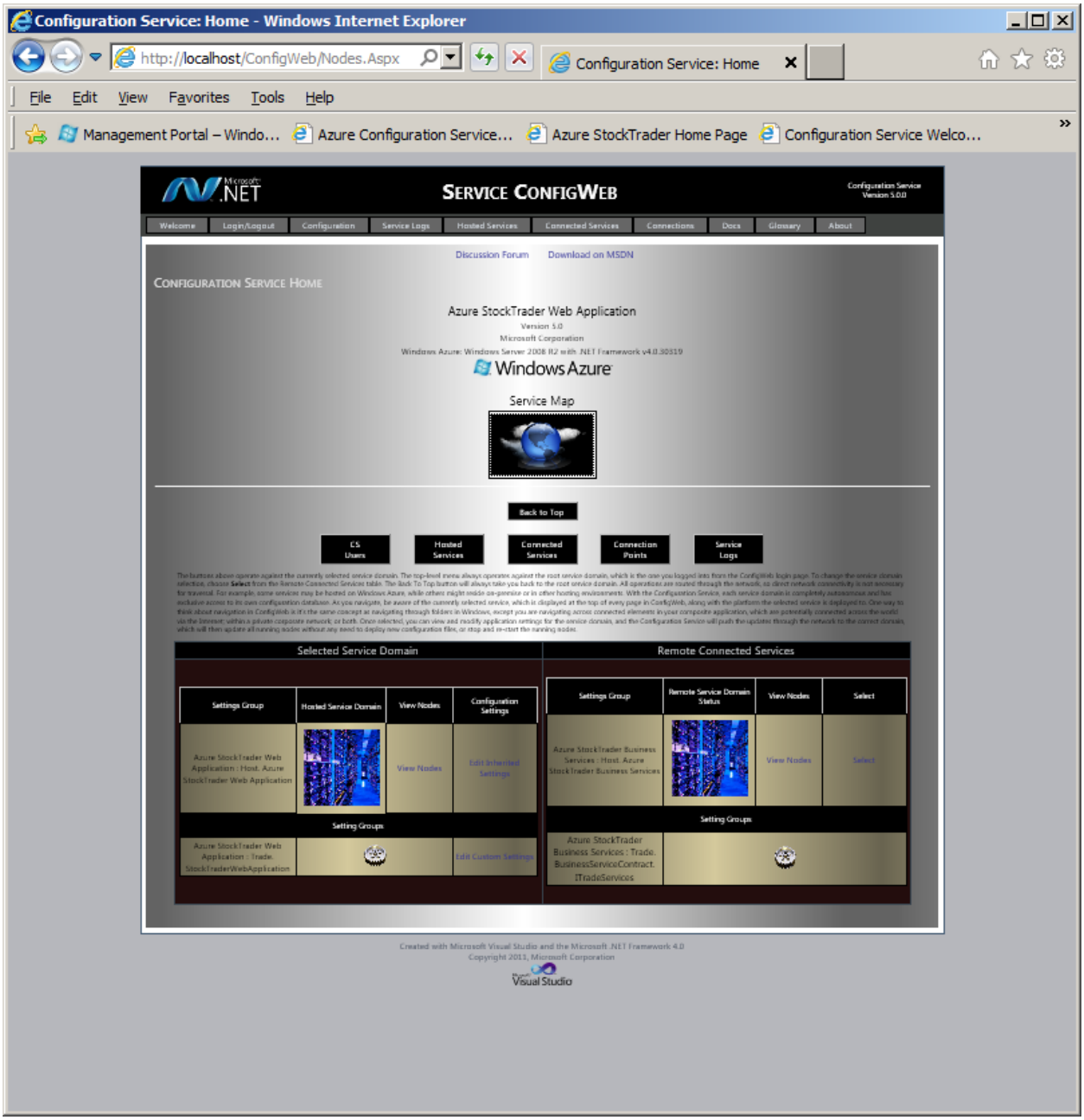


When you hover over the domain/virtual host (or an individual node), you will see how many service requests have been serviced. These are set to persist for each node on 60 second intervals, to keep accurate over time even as nodes are added, subtracted, taken offline, restarted, etc. The 60 second persist interval can be adjusted in ConfigWeb, it's another Configuration Setting managed by the Configuration Service.

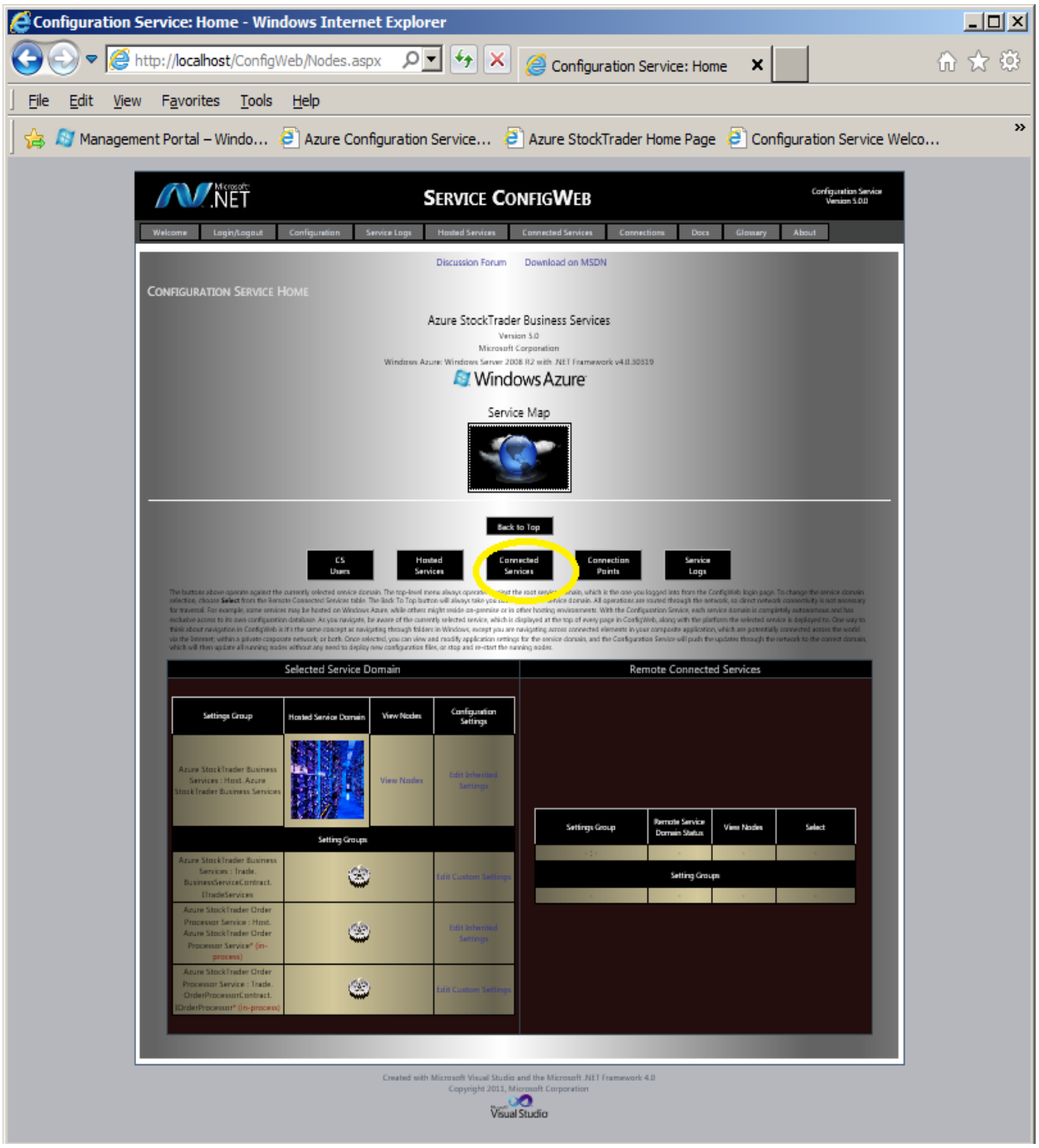
Note by default the Configuration Service is counting all activations of the primary service class; this count is accurate since the service is set to activate per call. Also note that visiting one StockTrader Web page actually involves multiple remote service calls, not just one per page. So as you click around a few pages as a user in the StockTrader web application itself, you will see many more service requests by the BSL service than pages you actually visited. Each individual request is being tracked. For the Web application, we have the request tracker set to just track **valid logins**, not all page visits.

Now that the StockTrader Web application is set for remote mode to (securely) utilize the StockTrader Business Services domain; the next step is to bring the Order Processor Service into the equation. Currently, the Business Service layer is directly placing orders. So, to demonstrate the use of a service for this processing (potentially hosted by another company, or in another data center, on-premise, or in the public cloud), we will now switch the Business Service layer to send buy and sell orders to a remote service, the Order Processor Service (OPS).

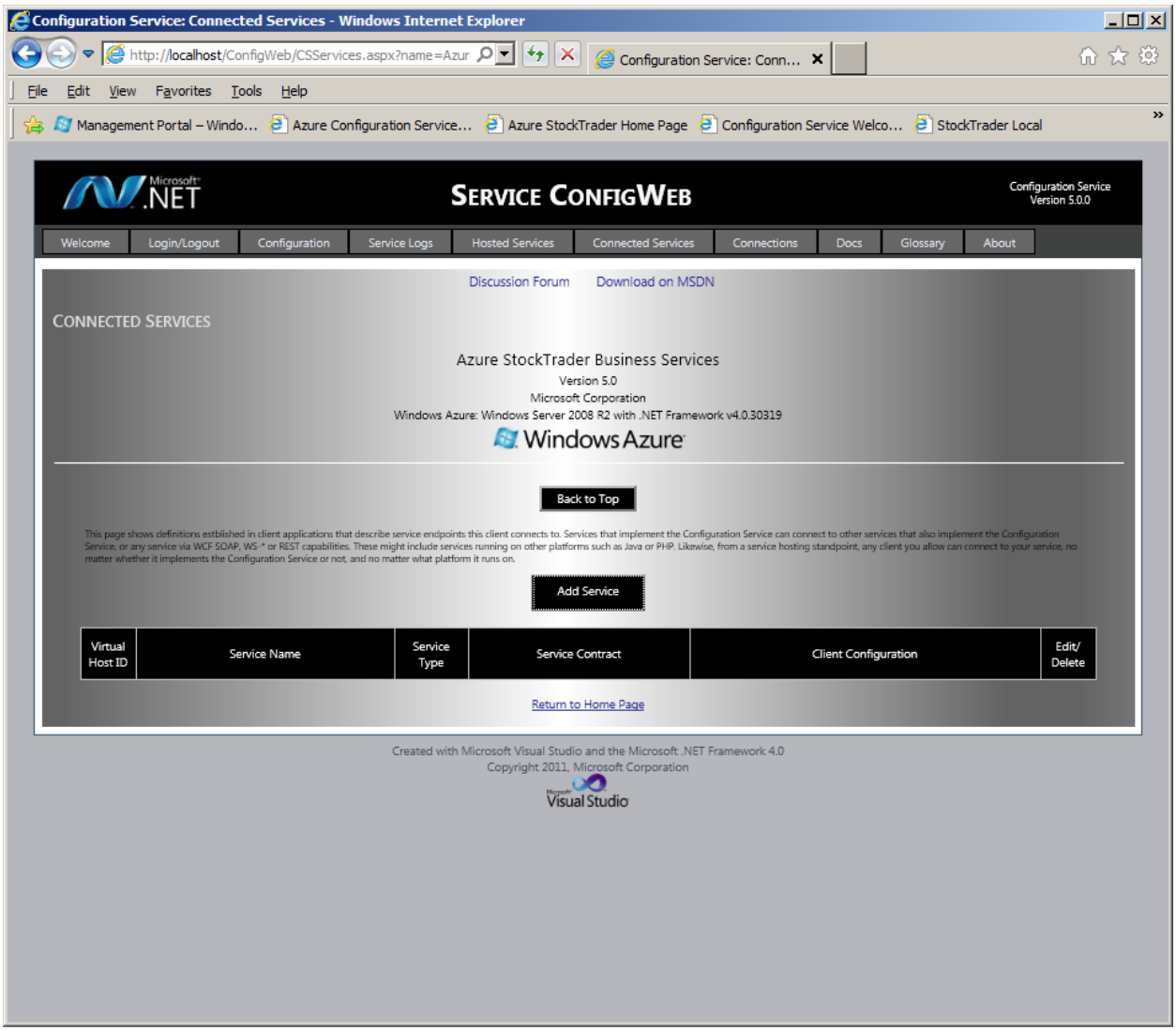
- Close the Service Map. Click on the Configuration menu item to make sure you are at the home page with the root StockTrader Web application selected.



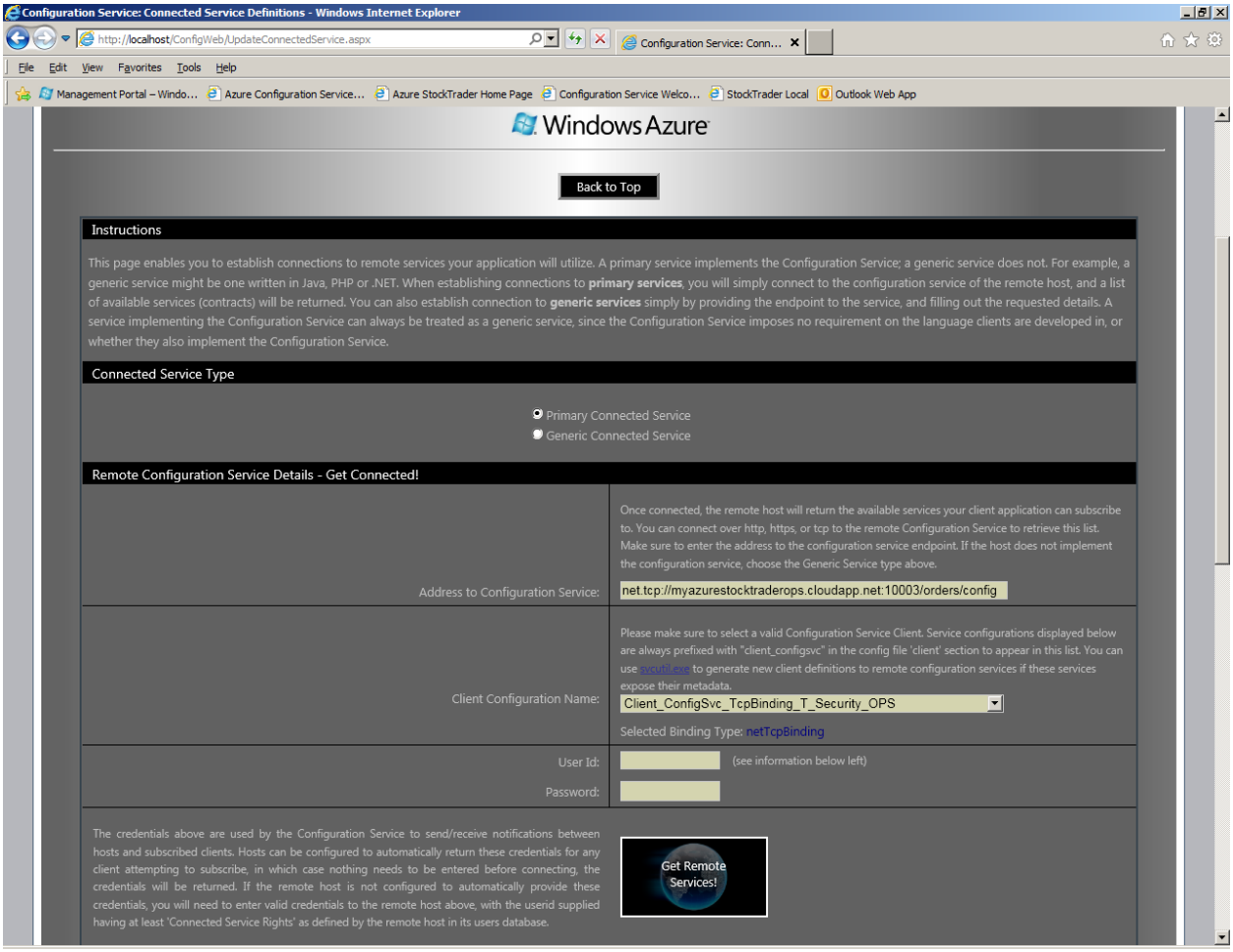
- You will now click on the '**Select'** link for the Azure StockTrader Business Services domain.



- Now, with Azure StockTrader Business Services selected, click on the Connected Services Button (not the top-level menu item).



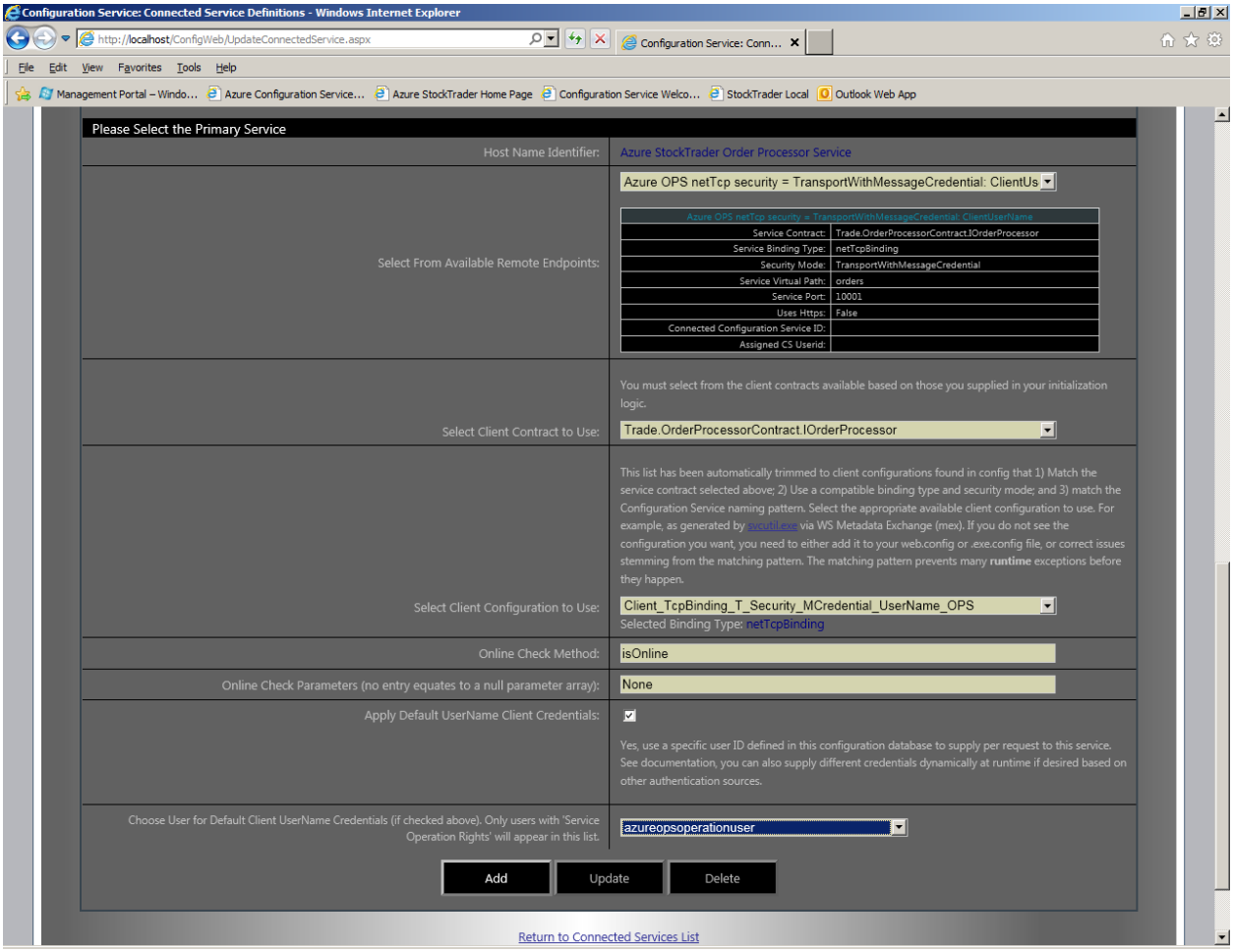
- Now click **Add Service.**



- You will now type in the address of your StockTrader Business Service domain (Azure Hosted Service). Shown above is net.tcp://[myazurestocktraderops.cloudapp.net]:10003/orders/config. You will replace [myazurestocktraderops.cloudapp.net] with the address to your service domain, followed by :10003/orders/config (the net.tcp endpoint for the OPS Azure Worker Role). Make sure you start the address with net.tcp:// since the Order processor exclusively uses net.tcp, and not https, for communication. (The net.tcp endpoint also is secure/encrypted).

- Make sure you choose the **Client_ConfigSvc_TcpBinding_T_Security_OPS** client configuration (which uses net.tcp and an expected identity for the OPS based on its service certificate (StockTraderOPSService.Com).

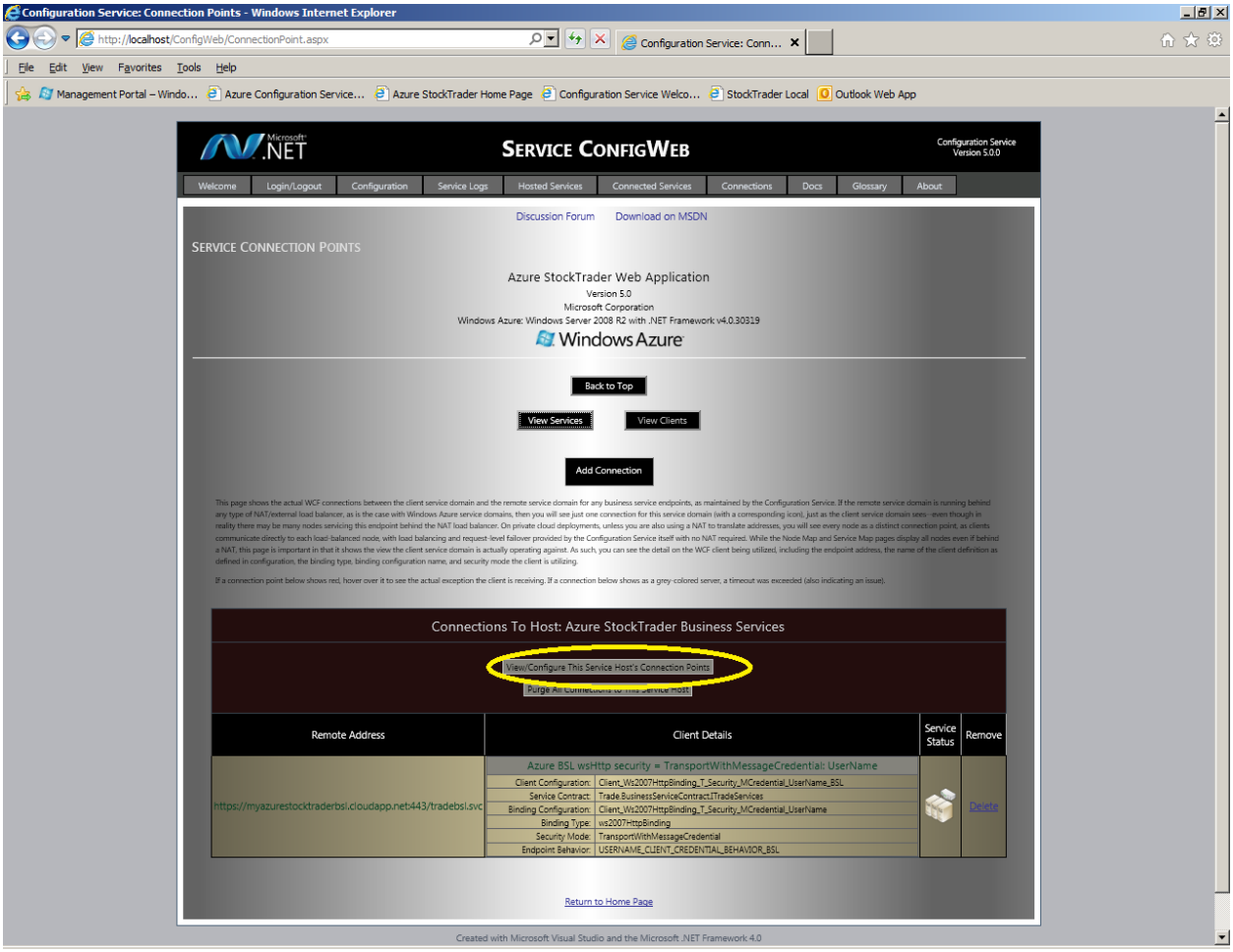
- Click **Get Remote Services**!



- Make sure to select Apply Default UserName Client Credentials, and then use the dropdown to select the **azureopsoperationuser** (this is pre-configured with the correct default password for the OPS service).

- Click **Add**.

- After the Service Definition has been added, click on the **Connections** top-level menu link in ConfigWeb at the top of the current page.

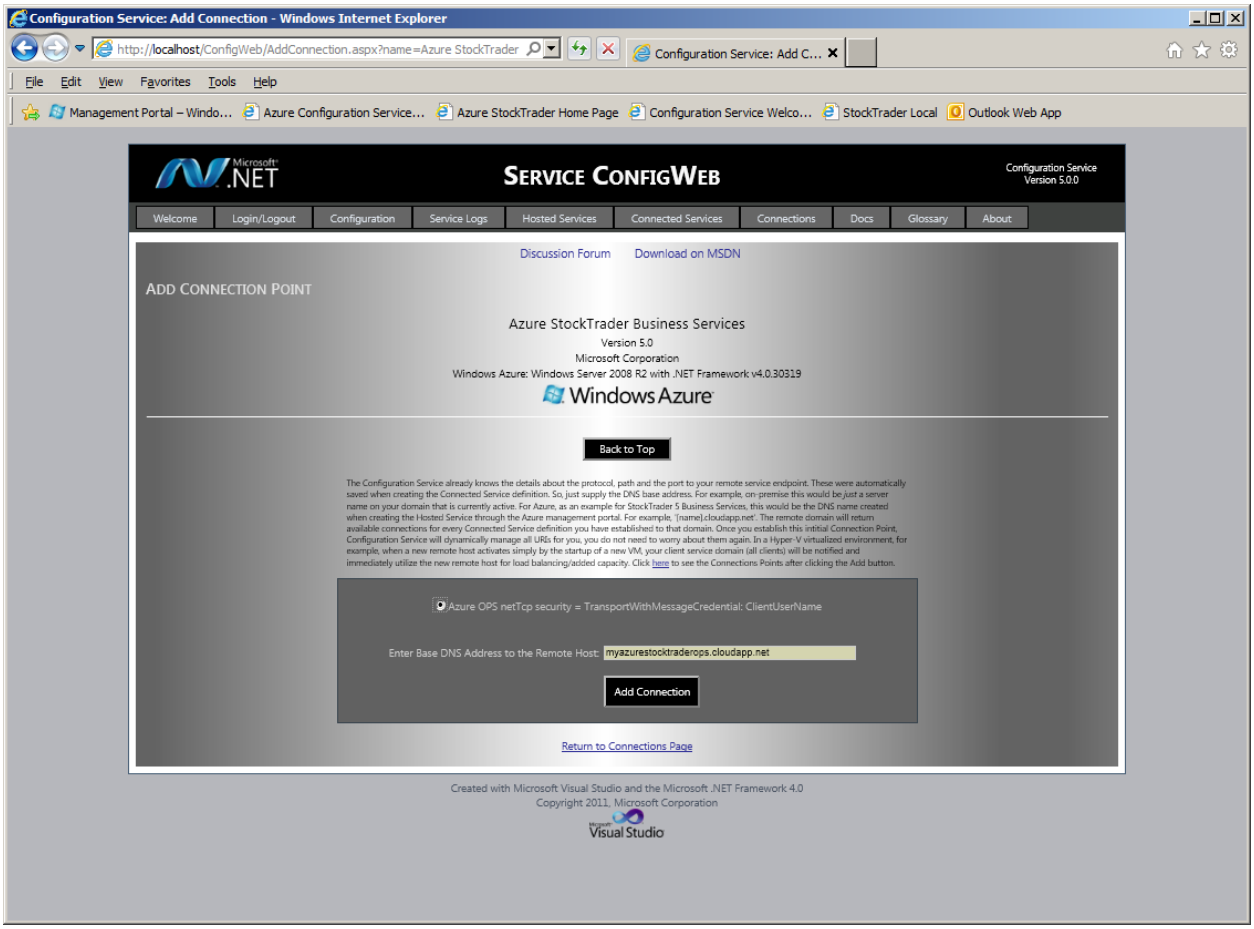


- At this point, you are looking at the StockTrader Web Application's (root node) connections to the Business Service tier. We need to drill down across service domains to view and configure the Business Service tier's Connection Points (to add one to the Order Processor).

- Click on **View/Configure This Service Host's Connection Points** as circled above.

- You can see at the top of the page, we are now looking at the Business Service domain's connection points to other services (and there are none, yet).

- Click **Add Connection**.

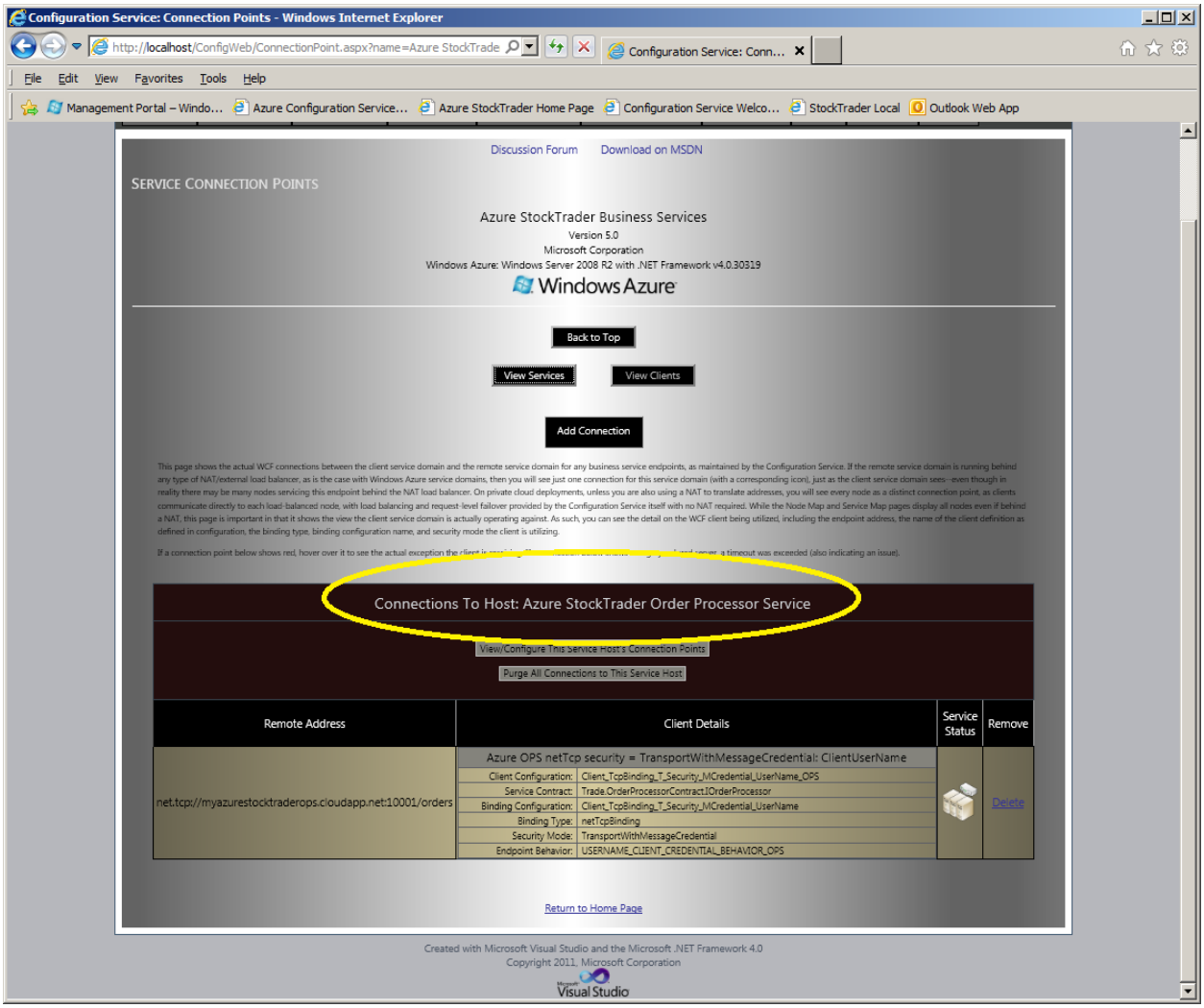


- You do not have to change the address, this is the correct address to your Order Processor Service hosted service domain, which the Configuration Service already knows.

- Simply click **Add Connection**. This might take a few seconds to complete.

- Now, click on the Connections Menu item again at the top-level menu.

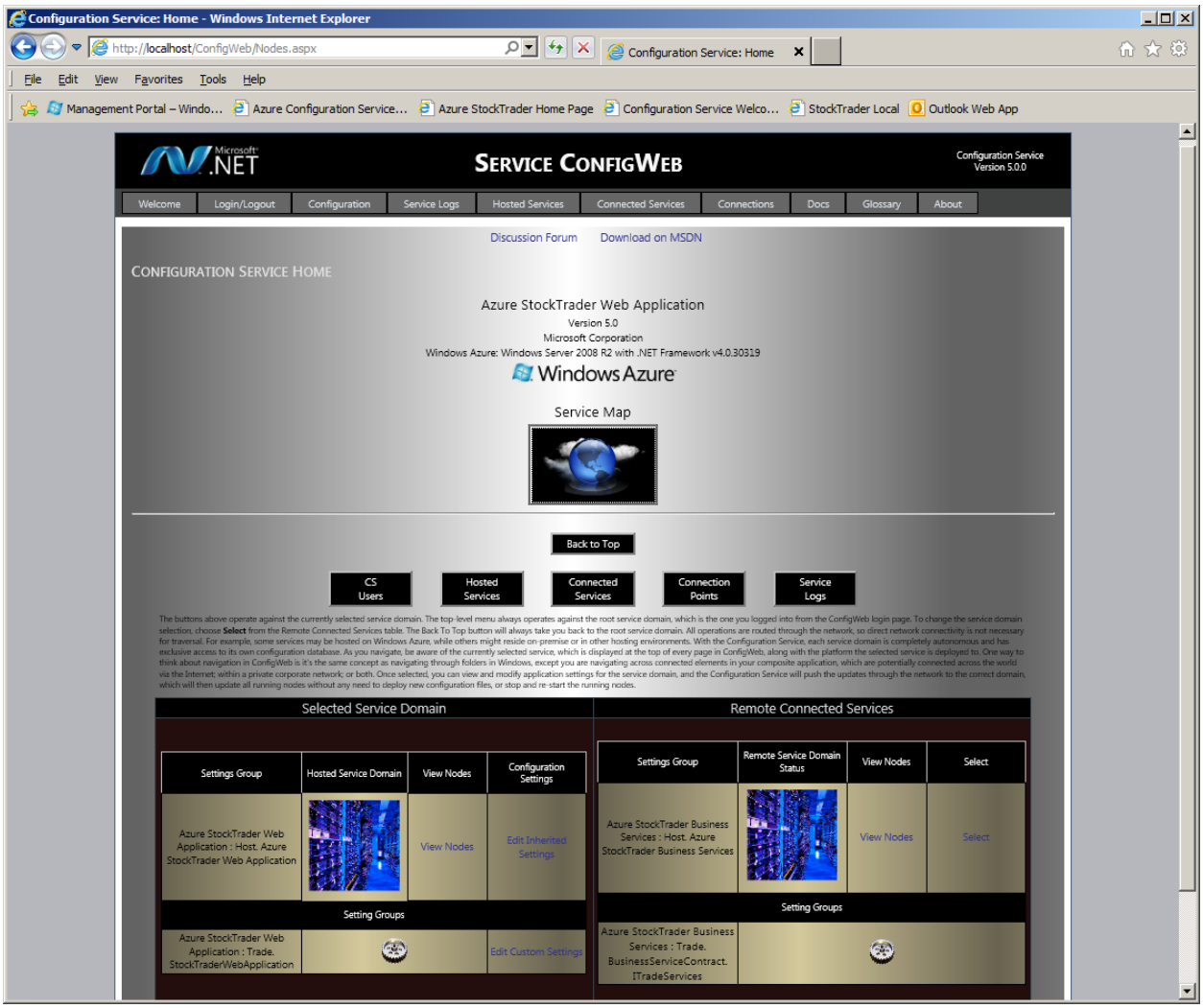
- Click on **View/Configure This Service Host's Connection Points** as you did before, to drill down into the Business Service tier connection points.
- Click on **View/Configure This Service Host's Connection Points** as circled above.



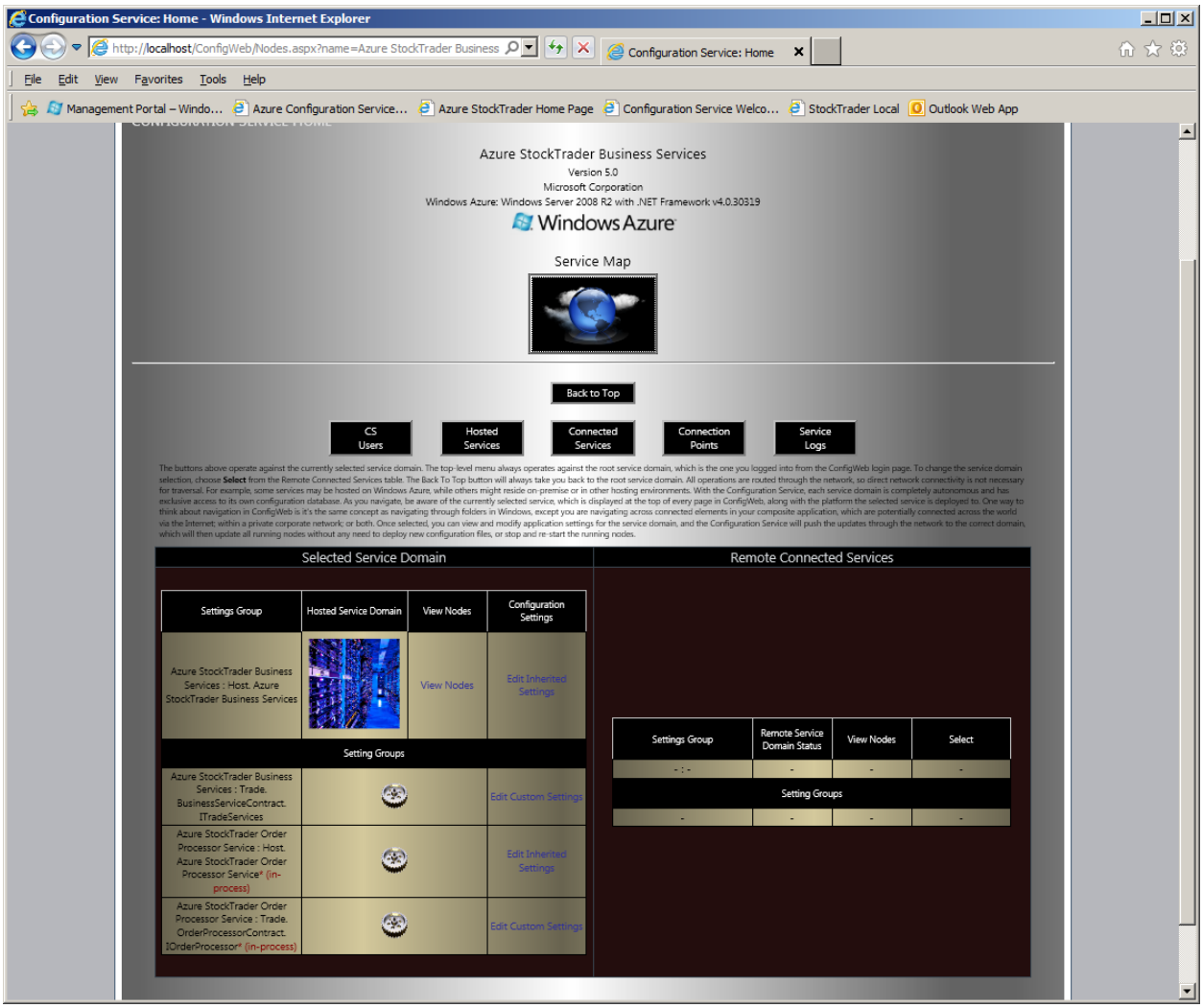
- Make sure you are looking at the right Service (BSL); and its Connection Points (to the Order Processor Service as circled above).

- Again, the Connection Point is added, but it will not be used (orders sent by the BSL to the remote Order Processor Service) until we change the Business Services's **OrderMode** setting.

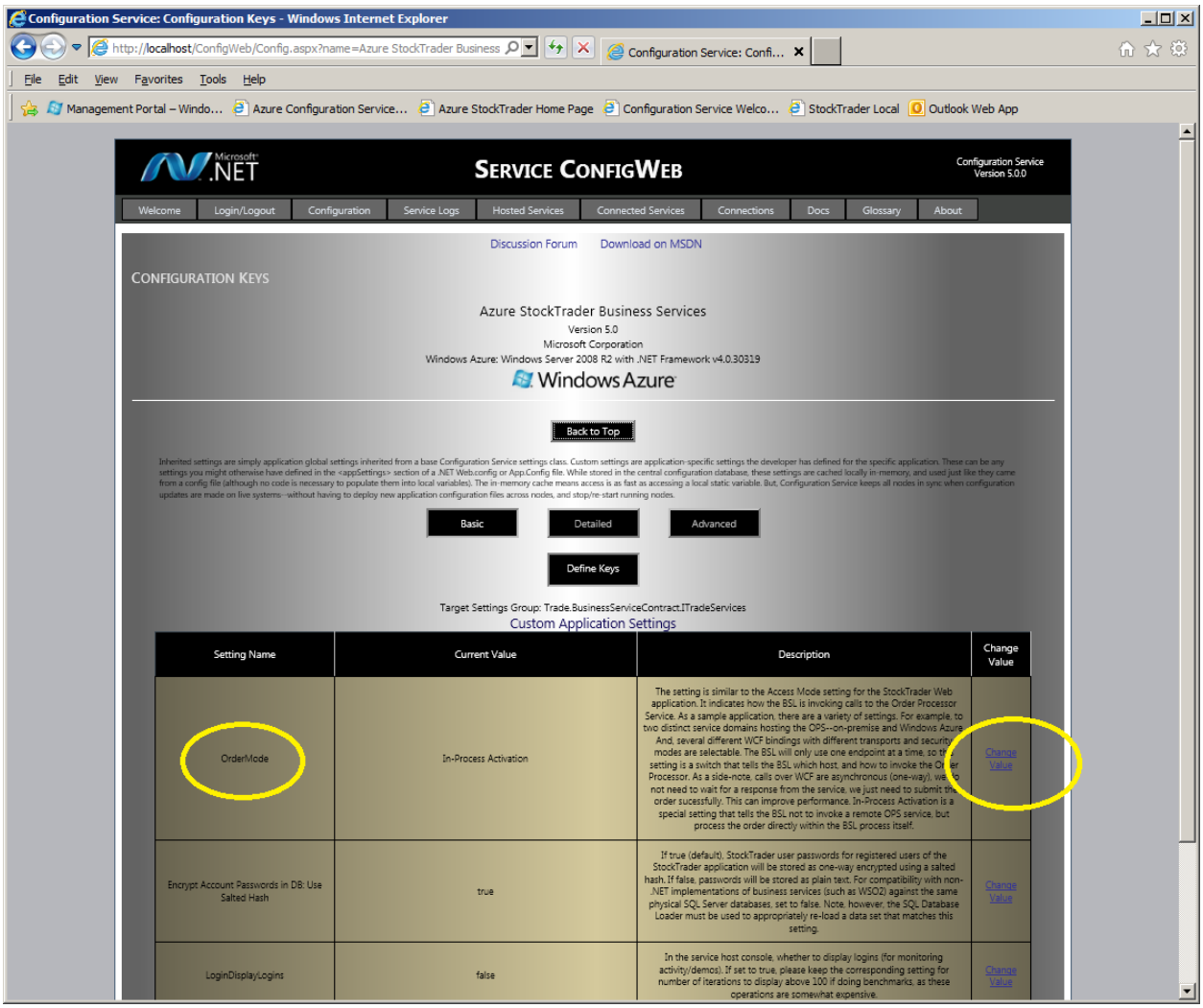
- To do so, click **Back To Top** to go back to the root home page for the StockTrader Web application.



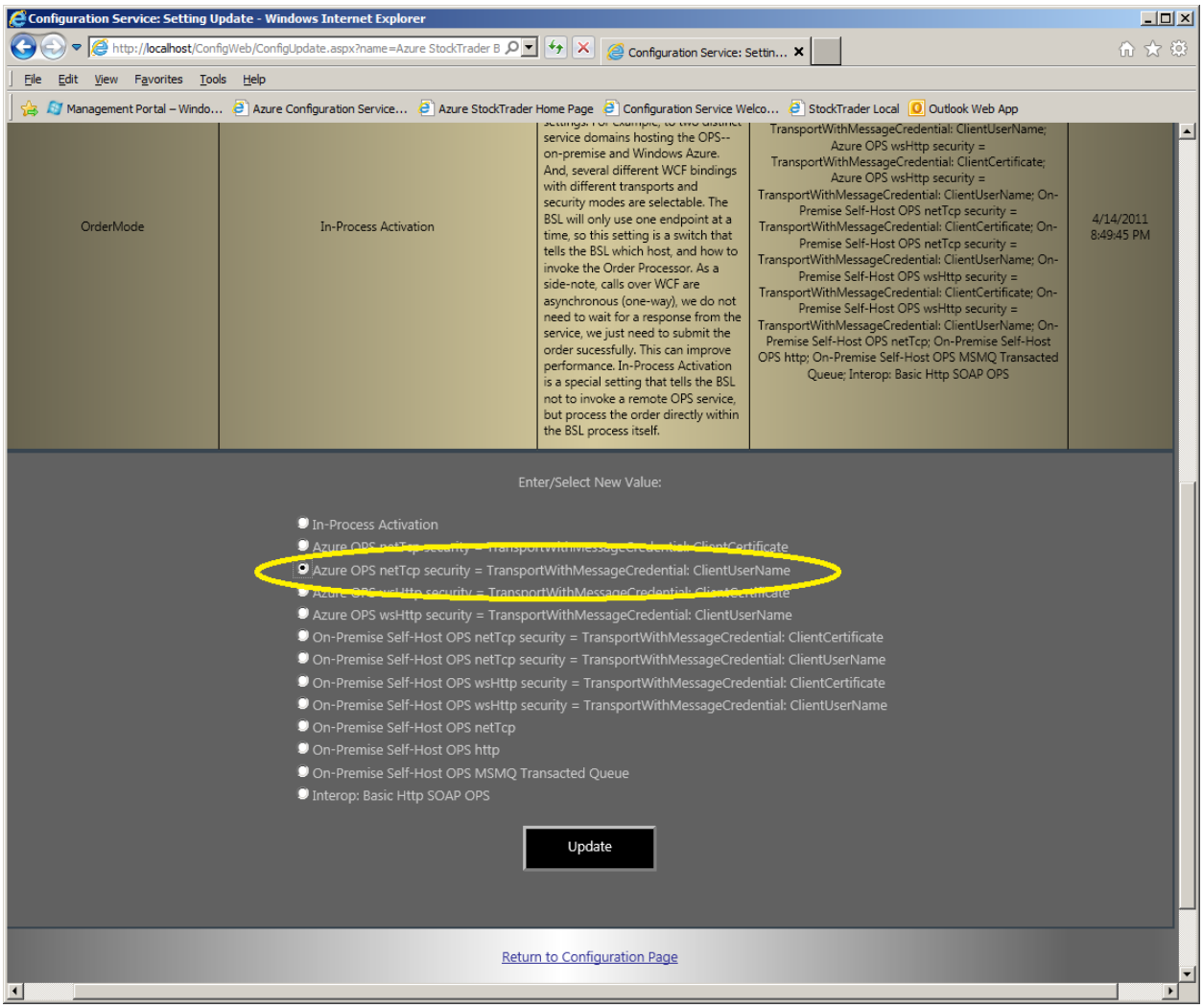
- Now click Select to select the remote Business Services domain.



- Now click **Edit Custom Settings** (the second link down).



- Click **OrderMode** setting's **Change Value** link.

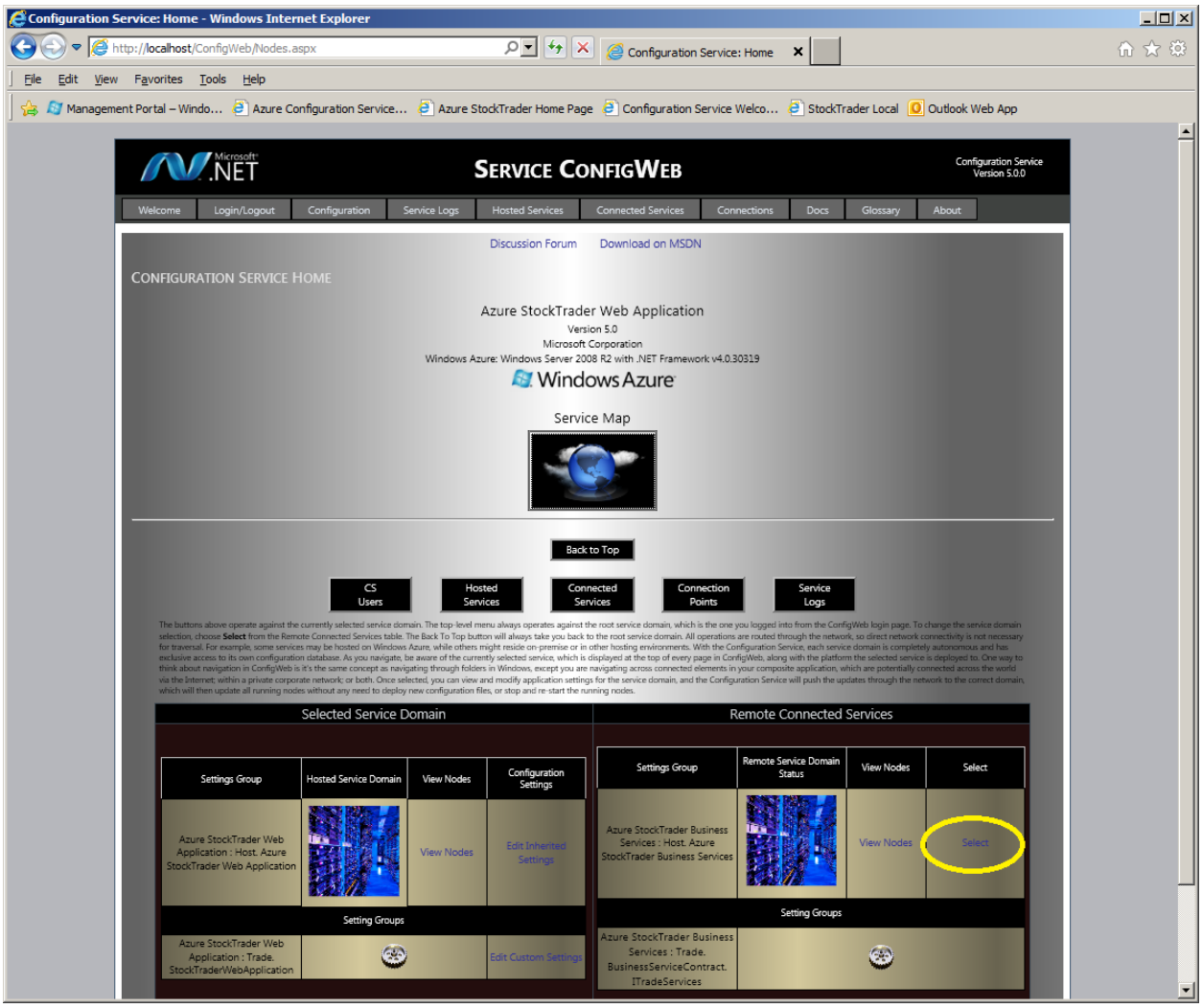


- Select the active endpoint as shown. This is:

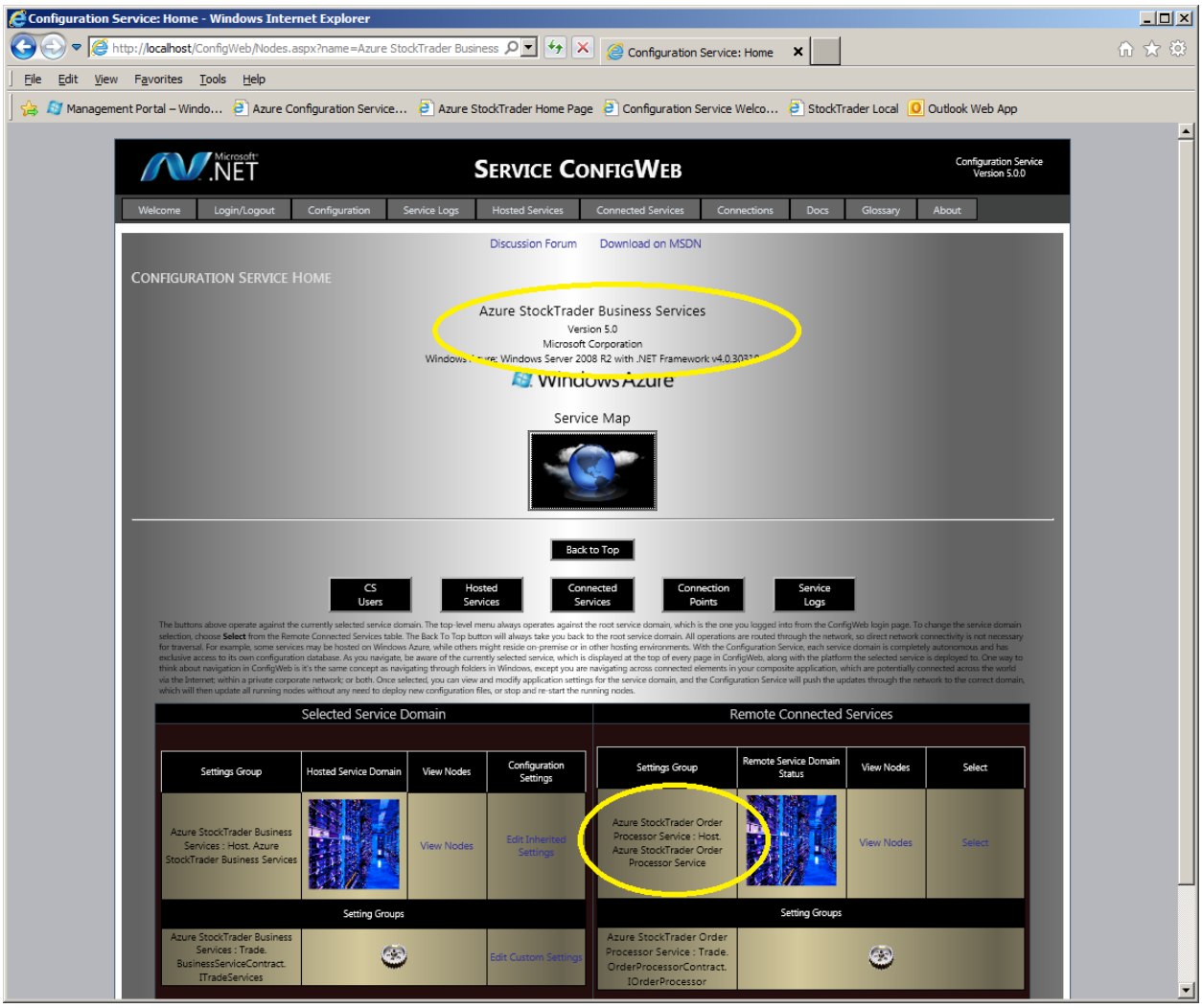
  Azure OPS netTcp security = TransportWithMessageCredential: ClientUserName

- Click the **Update** button.

- Click on the top-level Configuration menu item (or Back To Top).

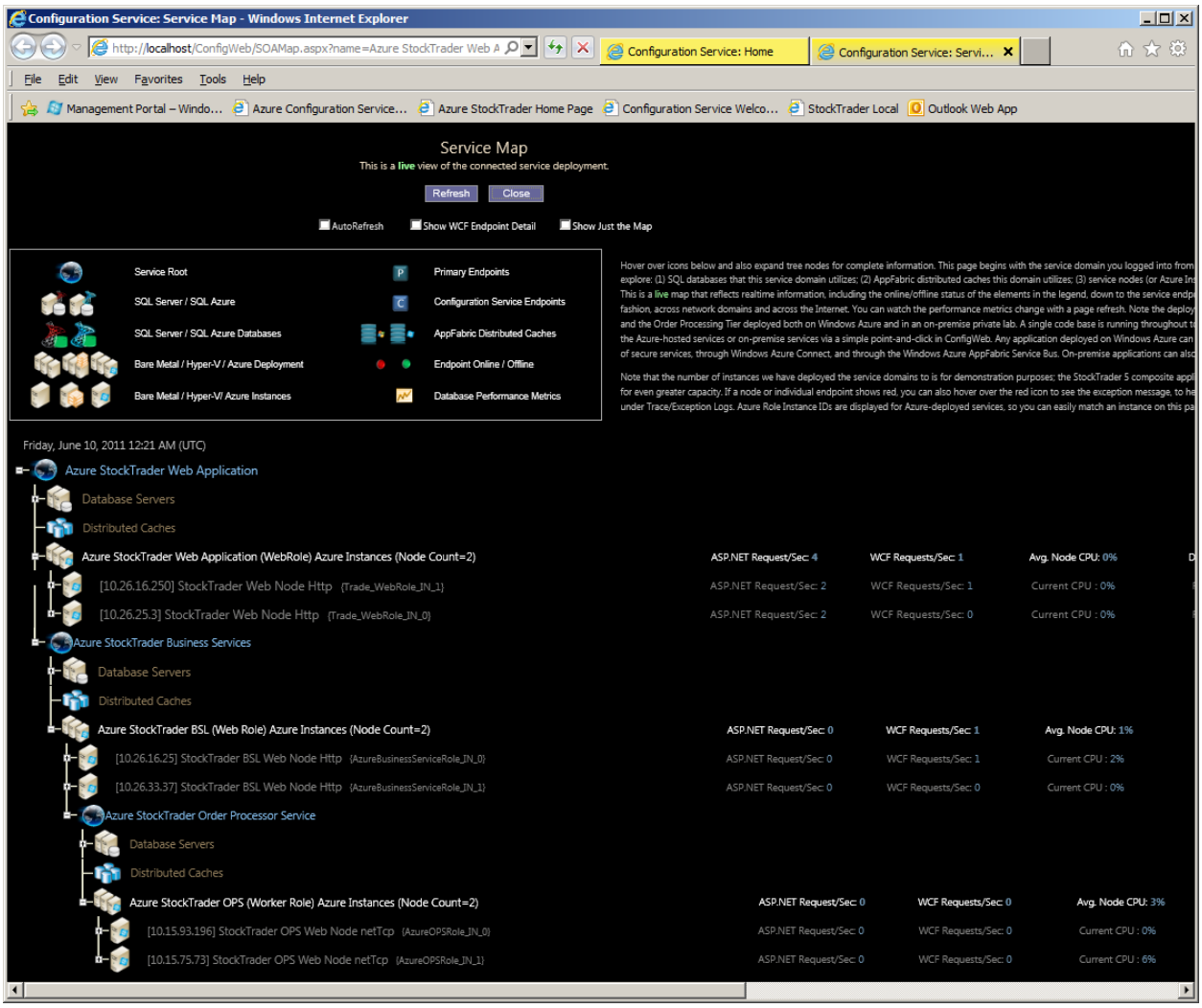


- Now choose Select to select the Business Service tier.



- Now the StockTrader Business Service Tier is selected, and from here you can traverse the network another layer by clicking on Select for the Order Processor Service if you want.

- Click on the Service Map button.



- You can now see the complete composite application deployment using all three tiers (plus SQL Azure database) of the application.

- If you login to the StockTrader Web application now, and place some sell and buy orders; they will be sent to the remote OPS tier for processing. You can do so, then refresh the Service Map page above to see the nodes for the OPS that processed the requests (hover over the nodes for the OPS above).

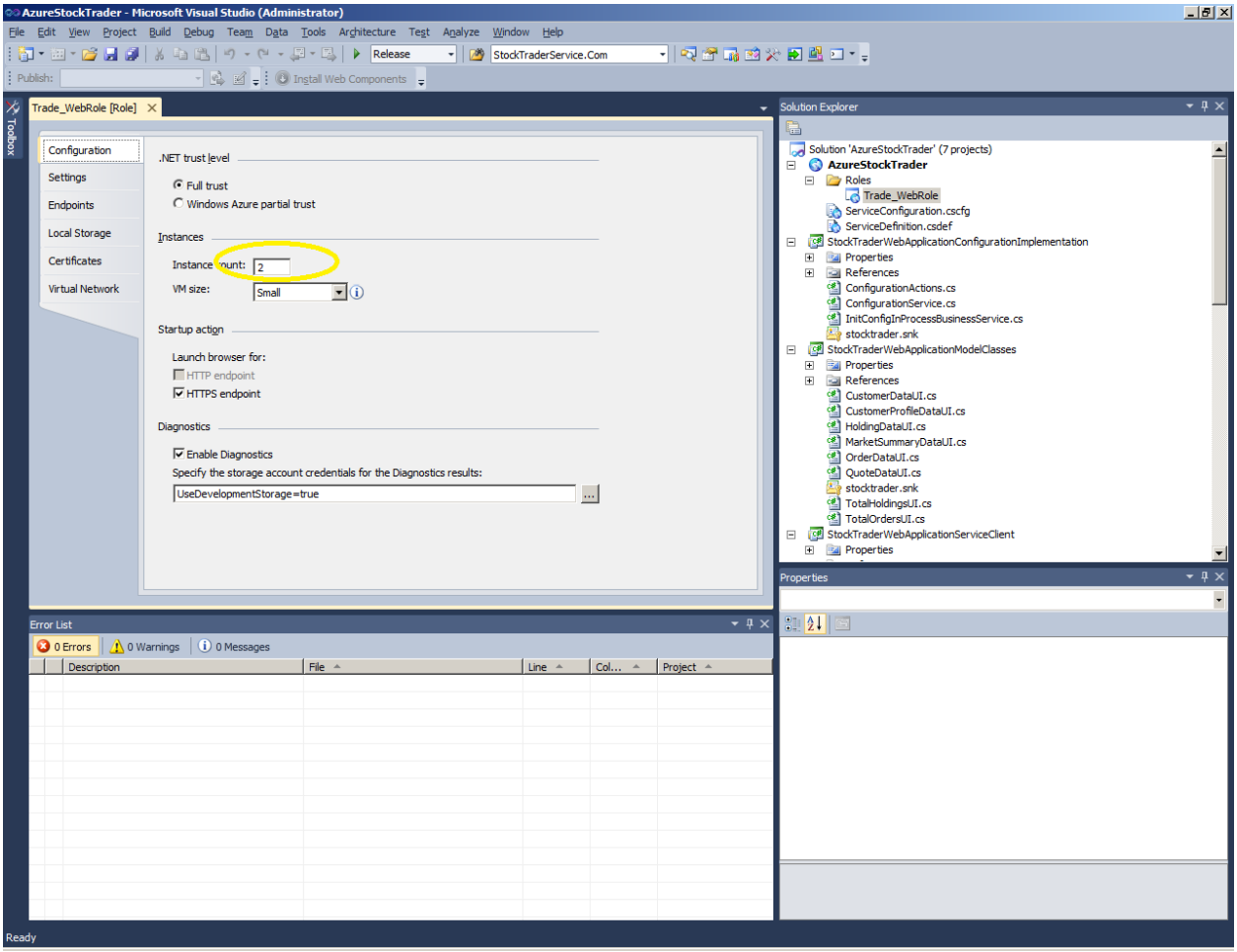
Adding Scale-Out, Load Balanced Nodes

For **on-premise deloyments**, to add a new node, simply install StockTrader on a new computer and indicate during install that the databases have already been created. Type in the address to the database server; the new installation will ensure your services share their single central

configuration repository. Hence, their configurations will be kept in sync. Simply startup the service hosts on the new computers, and the configuration system will add them as nodes; and notify existing clients of the presence of the new nodes. Closing the host program (or stopping the IIS service for IIS-hosted services) will similarly cause the node to remove itself from the system, and notify clients. The Web application and business service clients detect failed service nodes, and make a new request to a different node if more than one node is online. They will stop directing traffic to failed nodes altogether after a configurable number of consecutive failed online check requests (6 by default).

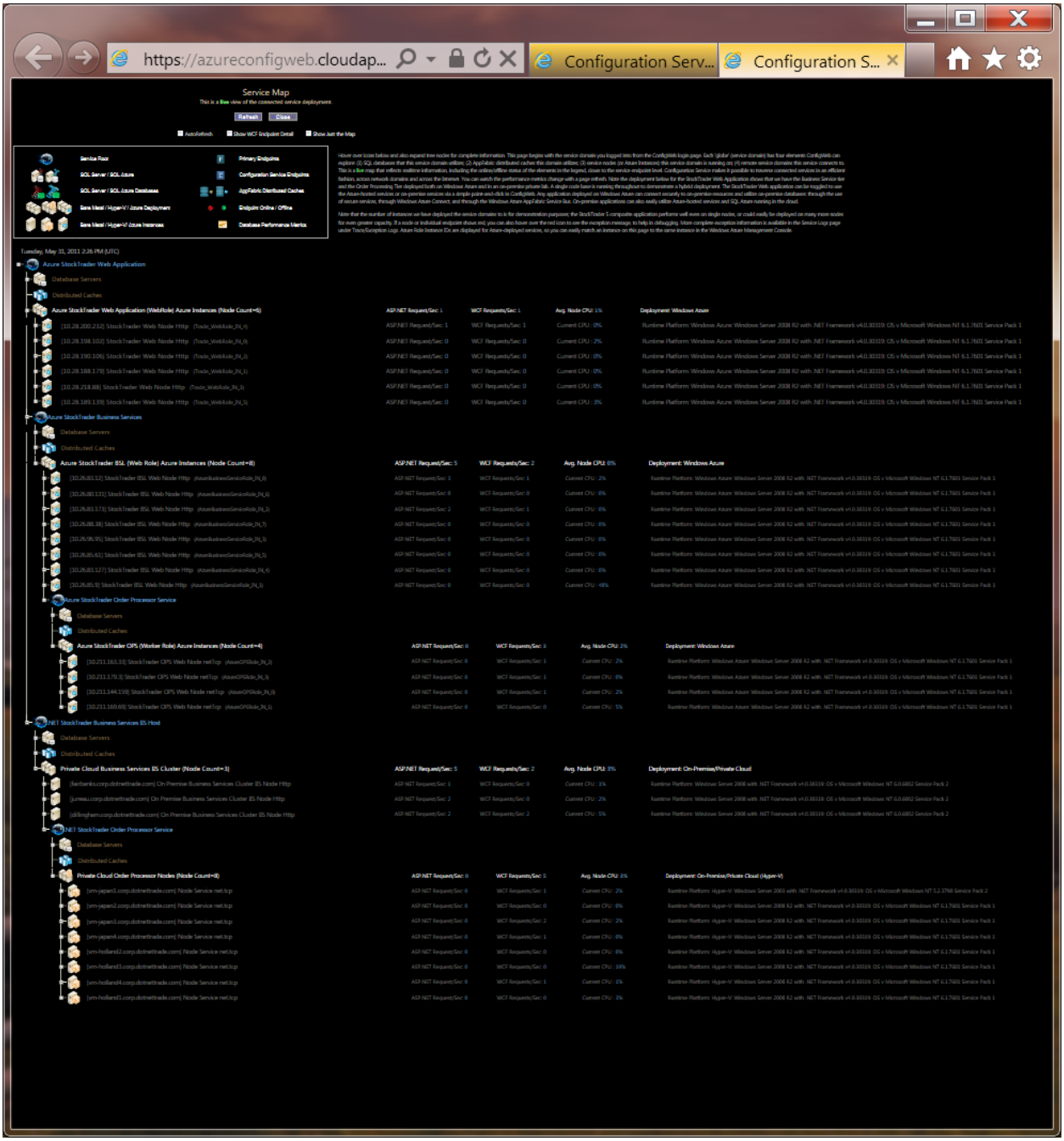
For Azure deployments, you simply use Visual Studio to change the number of nodes (and re-publish/upgrade the application). See the circled item below.

Or, you can also edit the configuration file from the Azure Management Portal to change the nodes the hosted service domain scales out across. Windows Azure automatically load balances requests against the nodes, which live behind the NAT/Azure Load Balancer (Azure Fabric Controller).



Below, you can see in our Azure/Private Cloud hybrid deployment we have scaled each tier to add more capacity, both on Windows Azure and in our private cloud running on Hyper-V/with System

Center VMM 2012. On Azure, Having at least two nodes active is important, as it ensures high availability and makes you eligible for the Windows Azure compute SLA for uptime.



For **on-premise**, you can also load balance/cluster using Windows Network Load Balancing (aka NLB; a feature integrated into Windows Server), or hardware load balancing. This can be accomplished on any tier. For example, the Web application, which is accessed by an unknown number of anonymous users via a browser, would be load balanced using NLB or a hardware load balancer, and its address hence virtualized to the outside world. Also, NLB or hardware load balancers such as F5 BigIP can be used to accomplish load balancing and failover for the Business

Services and the Order Processor Service (or any WCF service) in the same manner. *If you do so, you have to mark you Primary and Configuration Service endpoints with the load-balanced IP address:port or DNS address:port using ConfigWeb in the hosted services pages.

Azure Security Discussion and Changing the Security Configuration for WCF Endpoints

User Name Credentials

On install the StockTrader Web application, Business Services and Order Processor use no security for the on-premise installation. However, if you deploy the Azure projects, security is enabled by default, using **User Name** client credentials. The on-premise elements can be reconfigured to these same secure modes as well, optionally, via ConfigWeb. In this section we will discuss this topic, and show how it works. Later we will walk through some steps for the Azure hosted version to change the security from User Name credentials to **Client Certificate** credentials.

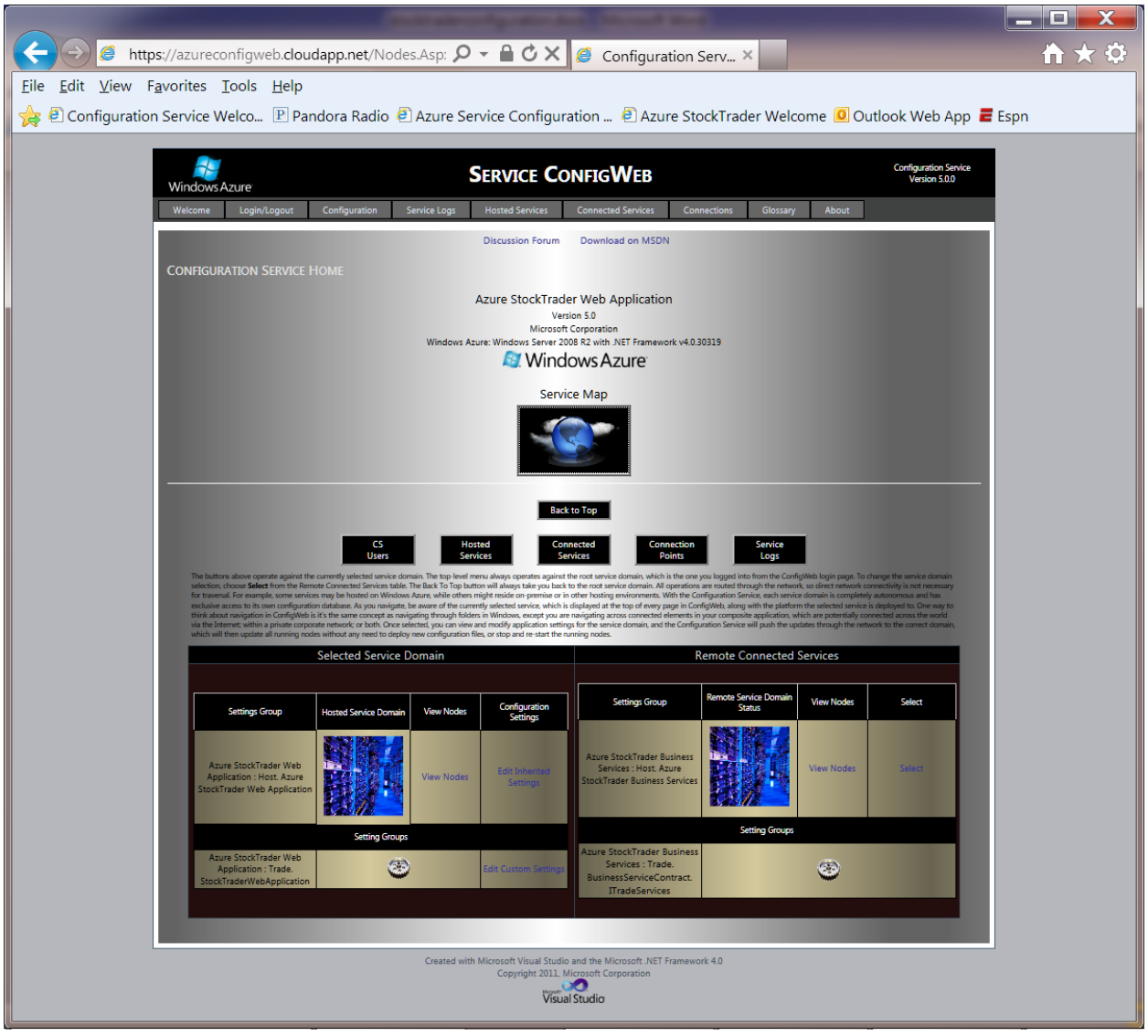
WCF supports a wide variety of security modes. Azure StockTrader uses https/ssl for the Web application to protect http transmissions, since these over-the-wire transmissions include the users' personal stock data, passwords for sign on, etc. While the StockTrader Web application authenticates the user logins via a custom Account table in the StockTrader database, the StockTrader Businesses services and Order Processor Service do not implement their own independent authentication (although they could be changed to authenticate every request with custom logic). So, to lock-down these internet-exposed service endpoints, we are using WCF security. This ensures that only the StockTrader Web application itself can access the BSL tier, and only the BSL tier can access the OPS tier to submit orders. We are doing so with WCF security set to TransportWithMessageCredentials. So, all communication is over https/ssl to protect the content of the messages over-the-wire; and additionally the client must send client credentials that the WCF infrastructure automatically examines before it allows a service operation to be invoked. A Service Certificate (X.509) is used (and configured on the WCF services) in this configuration, both for the Business Service tier which operates over https/ssl; and for the Order Processor which operates over net.tcp.

By default, the client credentials are configured as User Name client credentials, which means on each service request to the BSL, for example, from the Web application, the Web application has to send a userid/password that WCF then authenticates on the service side. This is **independent** of how the StockTrader Web application uses ASP.NET Forms authentication to authenticate individual registered users of the application via the login page. Rather, the StockTrader Web application uses a global username/password much like applications use a username password (in the ADO.NET connection string) to connect to a database.

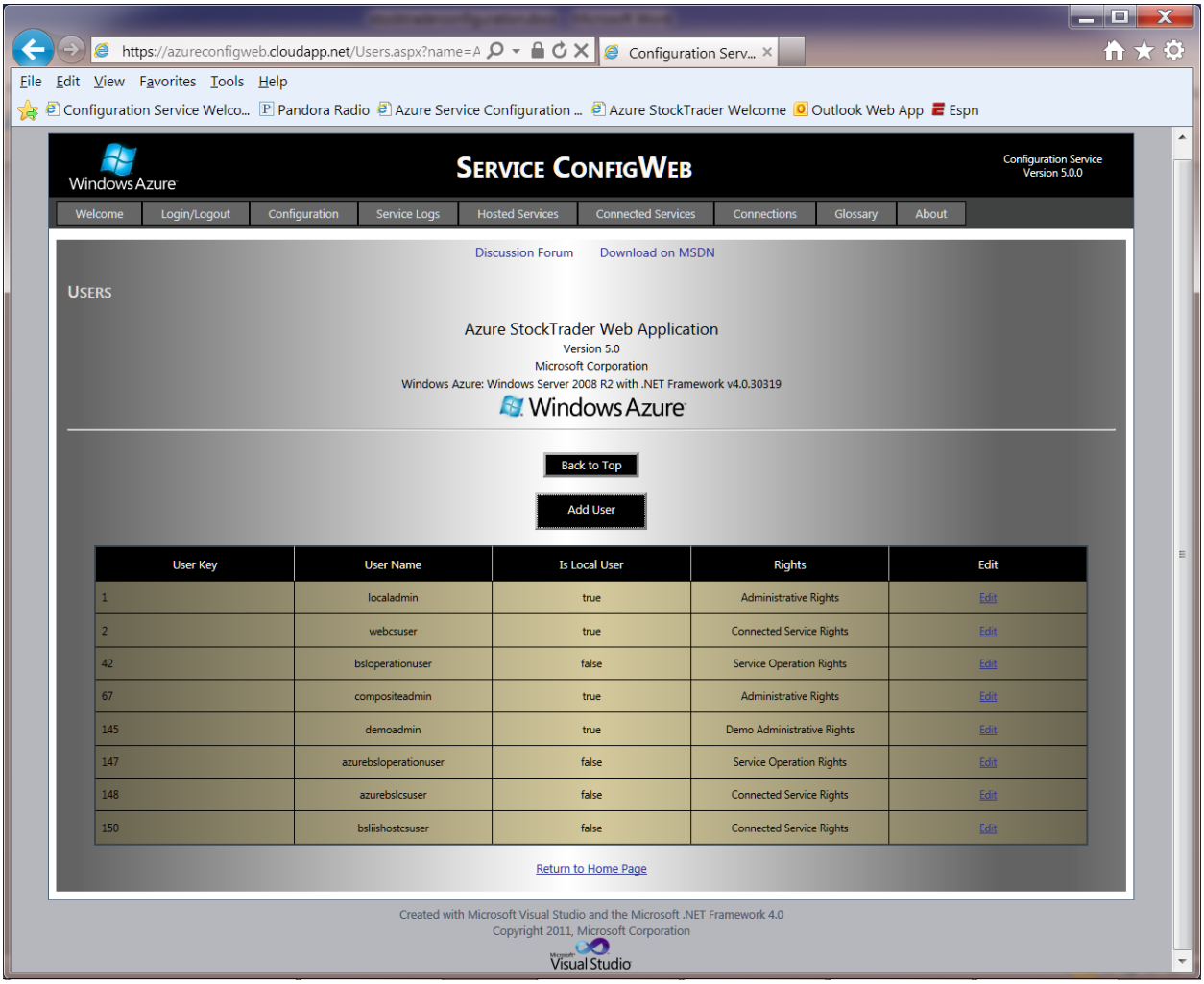
Where Does the User Name Come from and How is it Authenticated?

The username/password the StockTrader Web application supplies on each request to the Business Services tier is stored in the Configuration database, and can be configured via ConfigWeb. When establishing Connected Service Definitions, checking off "Supply Default Username Credentials" allows you to specify which username/password is used. You can create new user ids, or change the passwords on the ones already loaded using the Users menu item in ConfigWeb. If you have everything working on Azure in remote modes, you can test the security by using the Users page to change the password for the '**azurebusinessserviceopsuser'** user (here ops means service operations: 'ops')**.** If you change the password for this user, which is defined in both the StockTrader configuration database and the Business Service database, on either tier so they do not match, all requests will be rejected by the BSL that come from the StockTrader Web application (until you make them match again). So you can test this out now using ConfigWeb as follows:

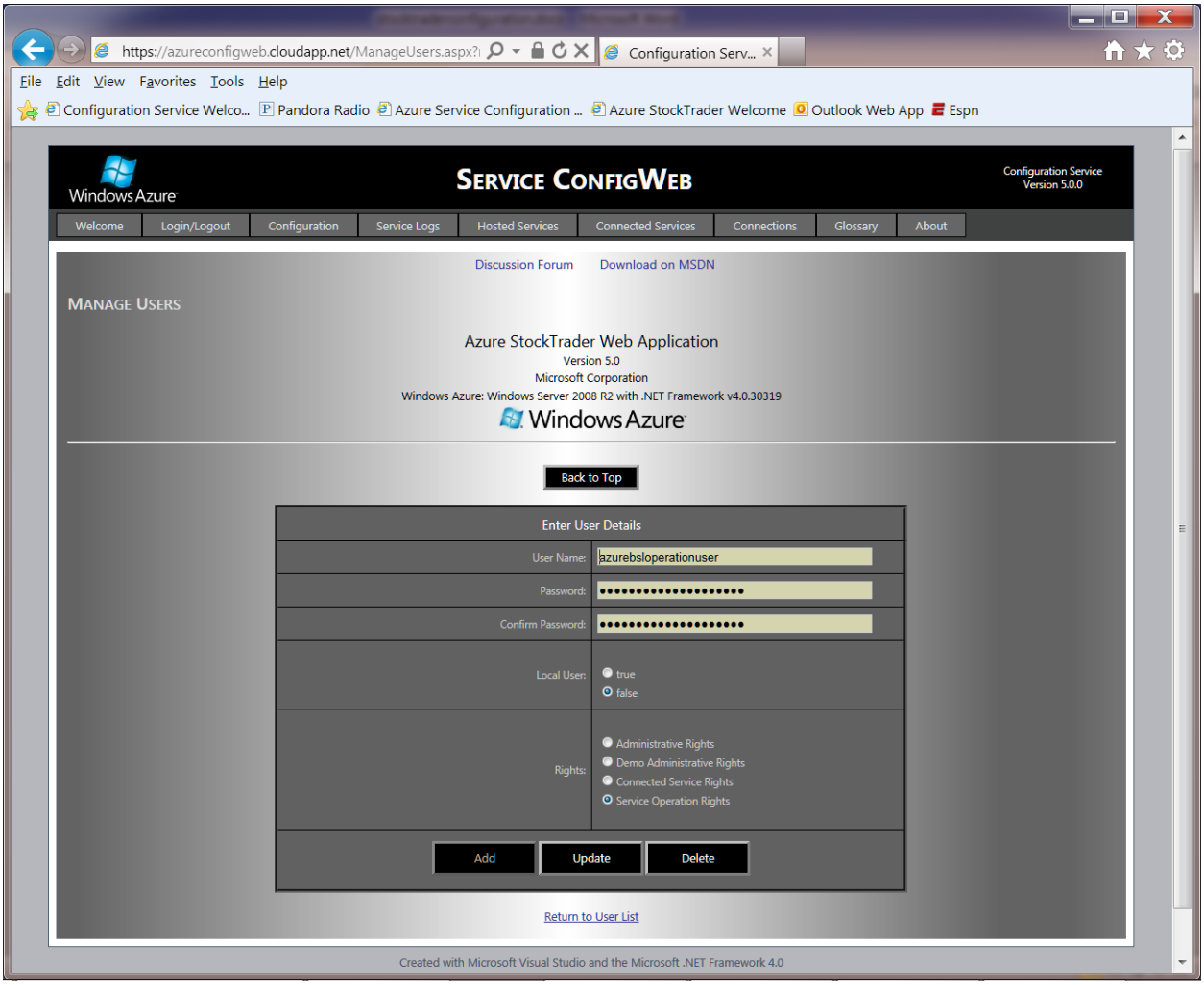
- Login to the StockTrader Web Application configuration service via ConfigWeb.



- Click on the **CS Users** button.

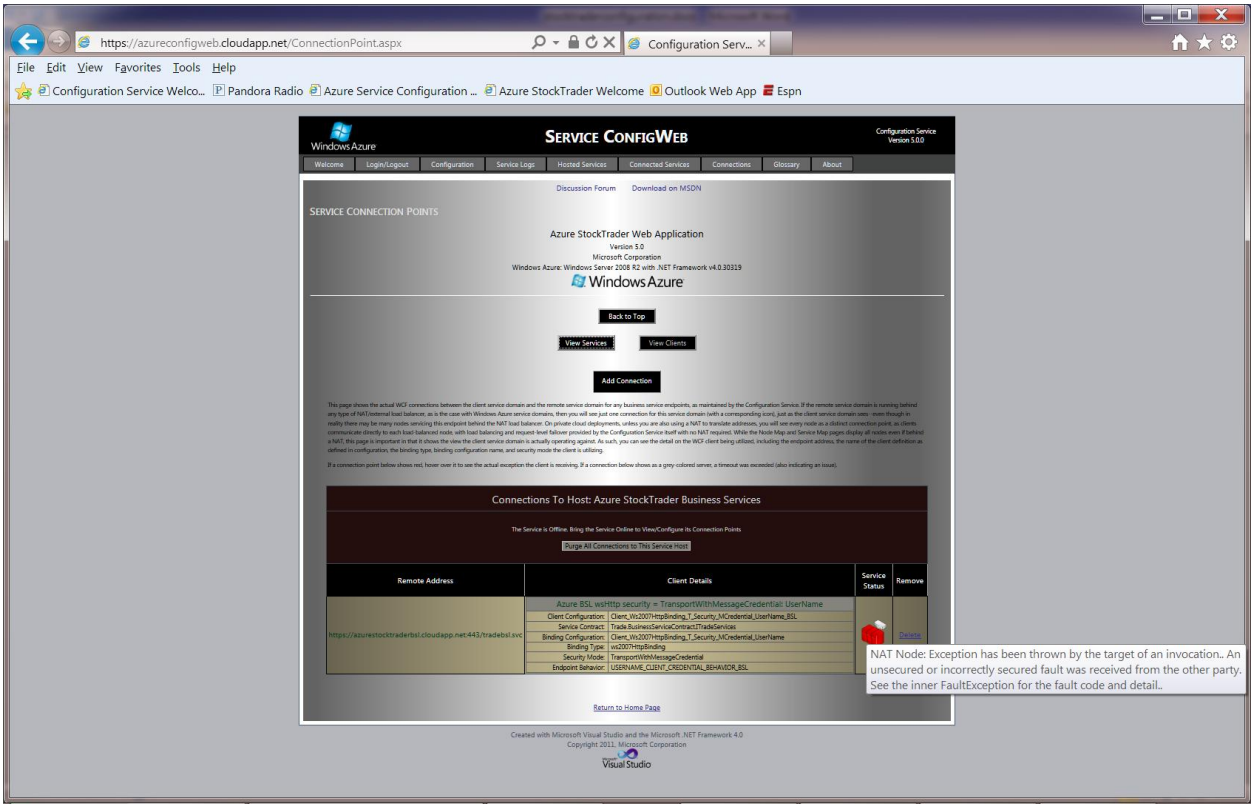


- Select the Edit link for the '`azurebsloperationuser`' user.



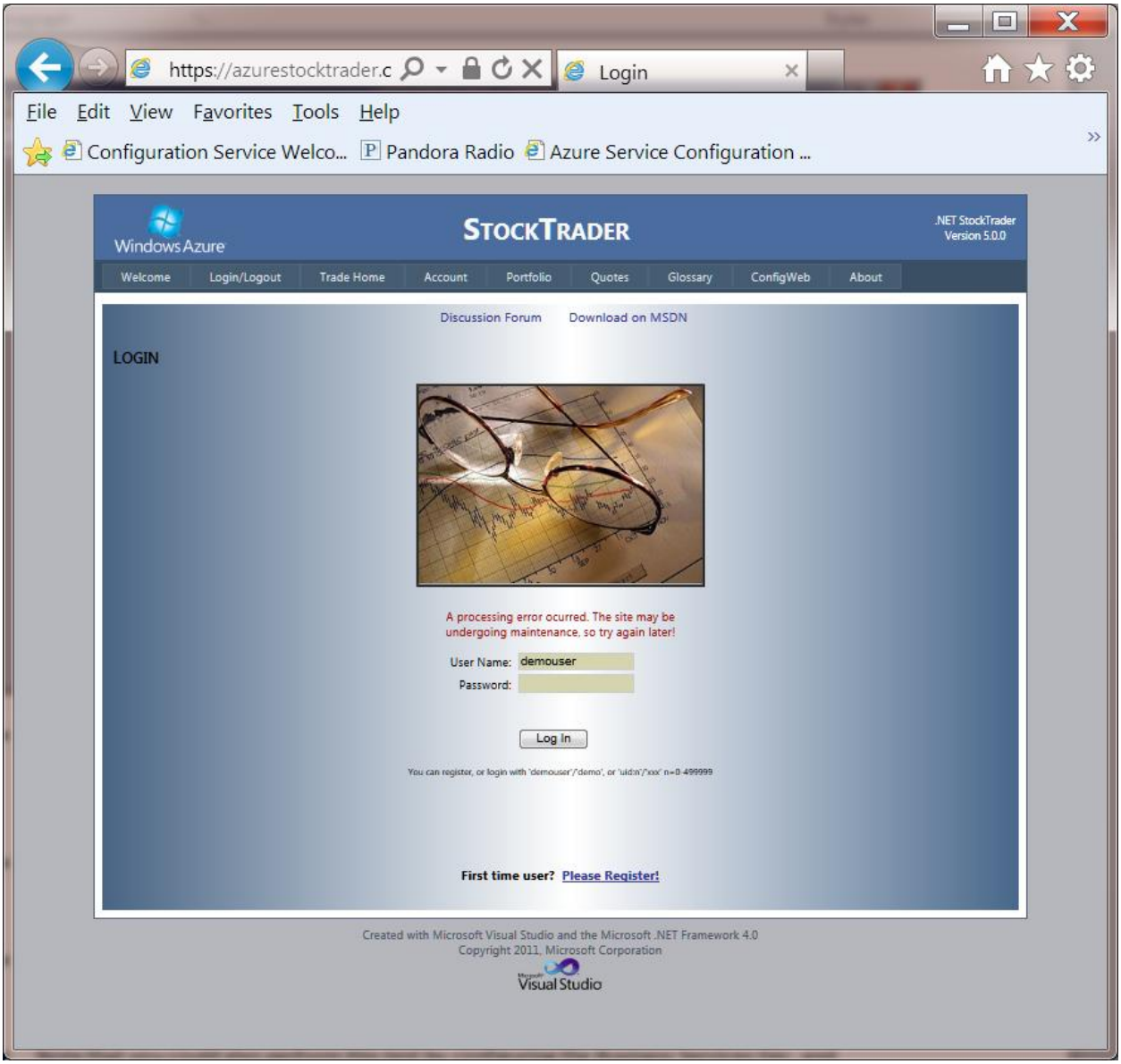
- Change the password to 'test'. The BSL service host will still expect '`azurebsluser#1`' as the password. So this will break the Azure deployment by creating a security violation (on purpose!) after changing the password above and clicking Update.

- Click on the **Connections** menu item.



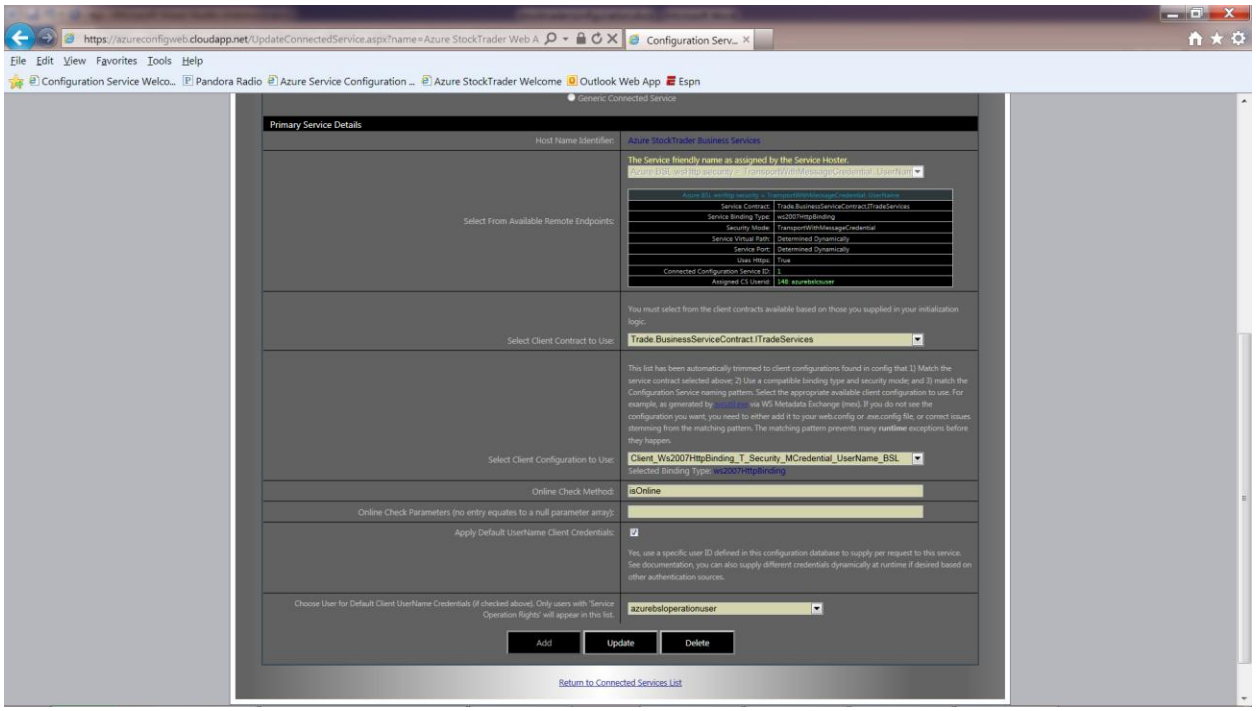
- Immediately you will notice the connection shows up as **red**. You can hover over it to see the security exception message. More detailed exception information is always logged and viewable in the **Service Logs** page.

- Try to login as an end-user to the StockTrader Web application (make sure you are running in remote mode as configured in the previous section). You will not be able to.

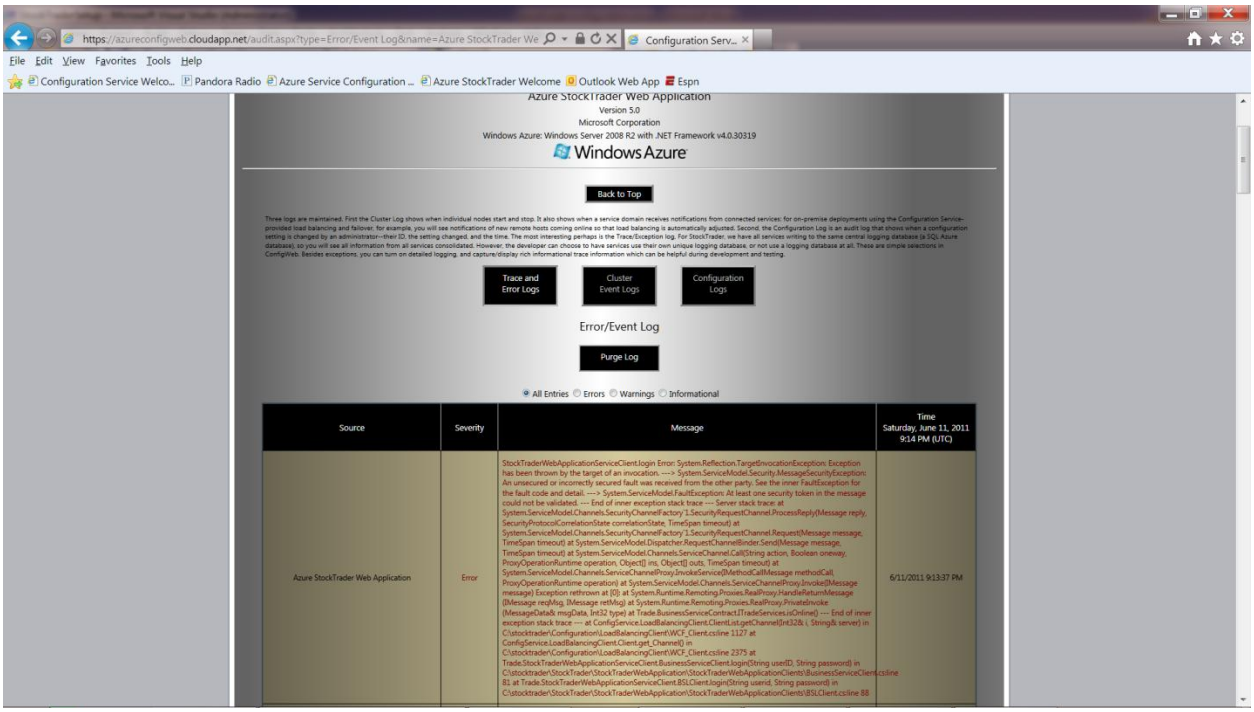


- You can repeat this test for the Business Service tier with respect to its client credentials supplied to the Order Processor Service. If you change these so they do not match what is expected by the OPS (which uses the same mechanism for authentication), you will be able to log into the StockTrader Web application as a registered user, but not place trade orders.

- Also if you were to change the Connected Service Definition (for either the StockTrader Web application-to-BSL definition or the Business Service tier-to Order Processor Service) to uncheck the checkbox '**Apply Default UserName Client Credentials'**, the StockTrader Web application will be unable to make remote service requests to the BSL (or BSL to OPS, depending on which one you change for the respective service domain).



If, in ConfigWeb, you now go back to the Service Logs page (Trace and Error Logs), you will see the following exception after the login attempt above:



- Now in ConfigWeb, change the password back to the default of 'azurebsluser#1'. Refresh the connections page, and try the Login for the StockTrader Web application again. All will be well again.

- Note that you could also perform this test by configuring the Business Services tier, and changing the password for this same user id via ConfigWeb for this tier, as opposed to the client Web application tier. The client credentials must match BSL-authenticated credentials.

How the User Name Gets Passed and Authenticated on Requests

The base client class for Configuration Service will automatically supply the client username credentials selected when you first create the Service Definition: *but only if the client configuration name selected includes 'T_Security' indicating transport security is applied on the binding, see the Configuration Service Technical Guide for details. You would never want to pass username/passwords over unencrypted channels.

The Business Services tier has a special WCF behavior applied to the Primary Service Host, named 'BSL_M_Security_UserName_Behavior". This service behavior tells the service to require username credentials on each service request. In turn, the behavior definition, which is contained in the Web.config file, tells WCF to use a custom username validator class to authenticate the usernames on each request. In the behavior definition within Web.config, the configuration tells WCF the name of the assembly, and the class to use within this assembly as follows:

```
<userNameAuthentication userNamePasswordValidationMode="Custom" cacheLogonTokens="true"
customUserNamePasswordValidatorType="Trade.BusinessServiceImplementation.CustomUserNameValidator,
Trade.BusinessServiceImplementation"/>
```

This class is the CustomUserNameValidator class in the **BSLCustomCertValidator.cs** file within the **BusinessServiceImplementation** project (it's used whether running on-premise or on Azure when configured for this mode). If you open this class and look at this class (which also contains the custom certificate validator logic, discussed later), you will find the following class definition:

```
public class CustomUserNameValidator : ConfigService.CustomValidators.CustomUserNameValidator
    {
        public override void Validate(string userName, string password)
        {
            base.Validate(userName, password);
        }
    }
```

It's very simple, because it inherits from a base class the Configuration Service 5.0 provides. This base class uses the standard .NET framework System.Security assembly to perform the validation on the service-side. The logic provided in the base class by Configuration Service 5 uses the **Users** table within the service's configuration database, looking for a valid matching user that has at least 'Service Operation Rights'. This method is automatically invoked on service requests as they come in from clients. The Configuration Service can perform the validation very quickly, since it maintains an in-

memory cache of its username/passwords from the Users table, and this is always kept synchronized with the database as long as updates flow through the Configuration Service itself (which ConfigWeb always uses). The base class looks like this, and is contained in the \configuration\ CustomValidators project within the Configuration Service Visual Studio solution:

```
/// <summary>
    /// Note how this class is tied in via a ServiceBehavior, defined in config, to override default
Windows auth validation.
    /// </summary>
    public abstract class CustomUserNameValidator : UserNamePasswordValidator
    {
        /// <summary>
        /// Overrides to instead validate the username/password against the Configuration DB Users table.
        /// </summary>
        /// <param name="userName">User id coming in as UserName credentials from client.</param>
        /// <param name="password">Password coming in as UserName credentials from client.</param>
        public override void Validate(string userName, string password)
        {
            ServiceConfigHelper configHelper = new
ServiceConfigHelper(ServiceConfigHelper.MasterServiceSelfHost._settingsInstance);
            ServiceUsers csUser = new ServiceUsers();
            csUser.UserId = userName;
            csUser.Password = password;
            //All we need to do is make this call, which validates the user for operation rights.
            if (userName.ToLower().Equals("demoadmin") || !(configHelper.setServiceRights(ref csUser,
ConfigUtility.CONFIG_SERVICE_OPERATION_RIGHTS) >= ConfigUtility.CONFIG_SERVICE_OPERATION_RIGHTS))
                throw new SecurityTokenValidationException("The operation failed authentication with the
username/password provided.");

        }
    }
```

Note we have a special user '**demoadmin'** setup that lets people browse ConfigWeb for our Azure deployment, but not change any configuration for the deployment. We explicitly disallow this user from using this ID to invoke service operations on the BSL and OPS tiers.

By throwing a security exception if the username/password supplied by the client does not match the required rights for the service, the service operation is automatically rejected by WCF. You could replace this base class implementation (or use the override method in the Business Service implementation) to do your own validation: for example against an Active Directory or other directory service, or against your own users database. Just remember that using the Configuration Service will be fast, since no database or network call needs to take place on validation of each service operation from the client (which, under load, might be coming in at thousands of times per second).

Changing Passwords so others Cannot Access Our Services

Locking down your specific deployment of StockTrader is pretty straightforward. You simply need to use the ConfigWeb Users menu to change **all** the passwords to the various accounts (users) defined there. Just remember the following rules:

Linked services will break if the root login credentials for ConfigWeb do not match the service you are trying to "Select" in the right hand connected services table in ConfigWeb home page. The link will simply show "Not Selectable." So, for example, if you change the 'localadmin' password to the Web

application, and then login to ConfigWeb when running in remote mode (**AccessMode** set to anything but 'In-Process Activation), you will not be able to "Select" and configure the Business Services from that login session. Instead, you would have to login to the Business Services tier directly as a different ConfigWeb login, using its old 'localadmin' password, to configure it. The 'localadmin' password is the same **on install** for all StockTrader service domains; so linked services just work; but really each service domain should have its own unique password.

Think of the 'localadmin' user akin to an 'sa' user on SQL Server. It always has to be there and cannot be deleted, as it's the master administrator for that service domain. So you might want to make unique passwords for each domain. And if you do, there is another administrator ID setup named '**Compositeadmin'**, and this one is set up to use for linked services. If you make each service domain's 'localadmin' password unique, but keep the same password for 'Compositeadmin' across service domains, then when you login to ConfigWeb as 'Compositeadmin', you will see and be able to select and configure remote services; but when logging in as 'localadmin', you will only be able to configure the root service you logged into.

Also, while you cannot 'Select' remote connected services in ConfigWeb if the credentials do not match, you should note that every service operation is authenticated by the remote service always, so even if you could 'select' it (you could modify the browser URL, for example, to make a request, or do so programmatically as a web service call); you will not get anything back from the remote service, since your request will fail authentication.

So in summary, to lock down your Azure-hosted (or on-premise) install of StockTrader and all its service domains, simply change all the passwords for each user, keeping the following in mind:

- csUser credentials must match. Whenever you see a user with 'cs' in the username, this is either a local Connected Service (hence 'cs') user for the hosted service, which it doles out to subscribing client to subscribe to notifications; or it's a non-local user that a client application is using to subscribe to host notifications. So you should make sure that username has the same password everywhere it appears; otherwise, like ADO.NET, if you change the password on the service (akin to changing a login password on SQL Server); then client apps using these credentials will fail authentication because they are using the old password (like an ADO.NET connection string using an old password—the application can no longer connect).

- The Service Operation user credentials also must match, as shown in the previous section, between client and the remote service it is passing these credentials to for authentication.

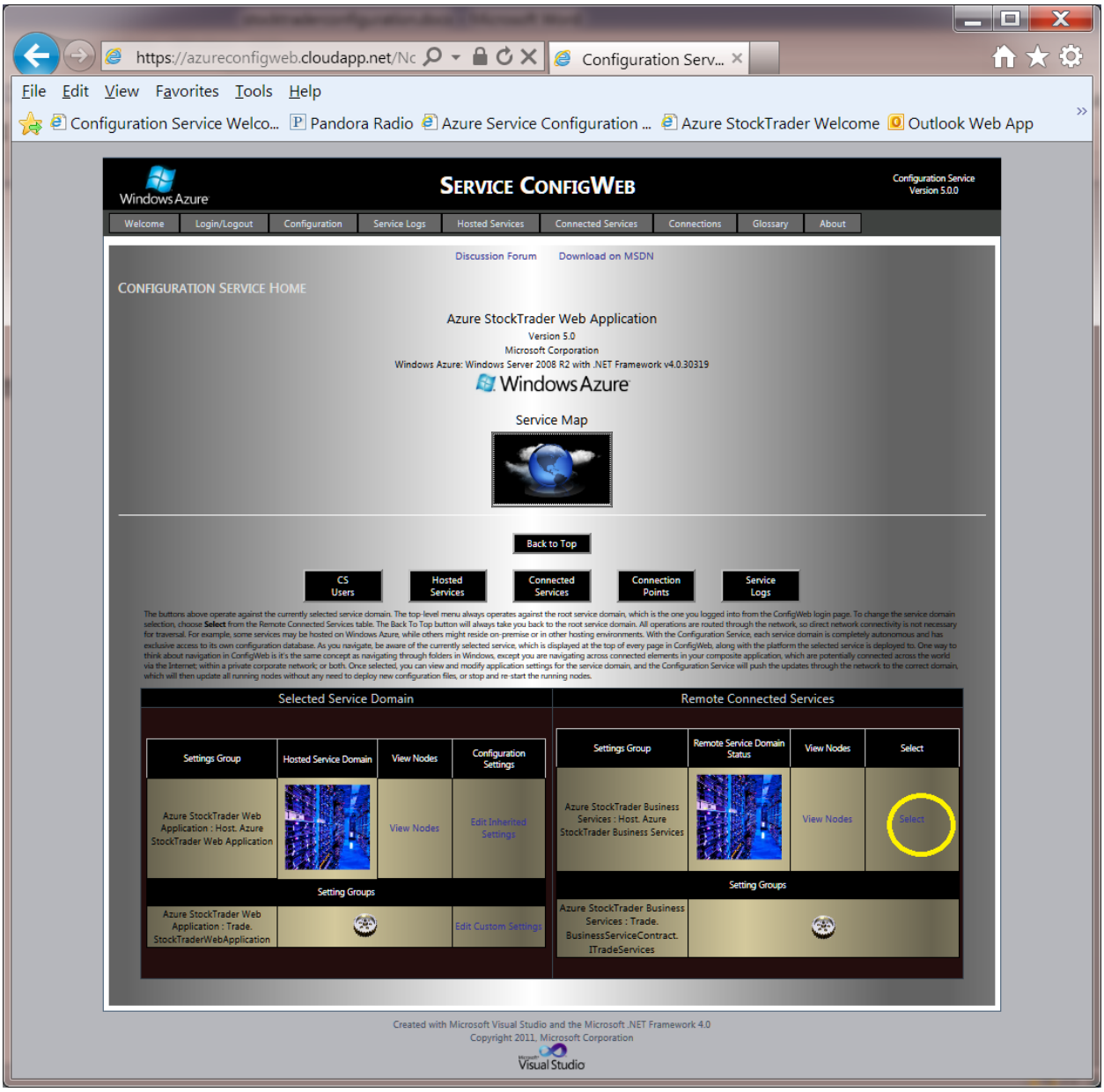
- For linked services to work, you must create some administrator credentials that are shared between client and remote connected service. You are free to create other admin usernames; but 'CompositeAdmin' is setup for this purpose.

Client Certificate Credentials

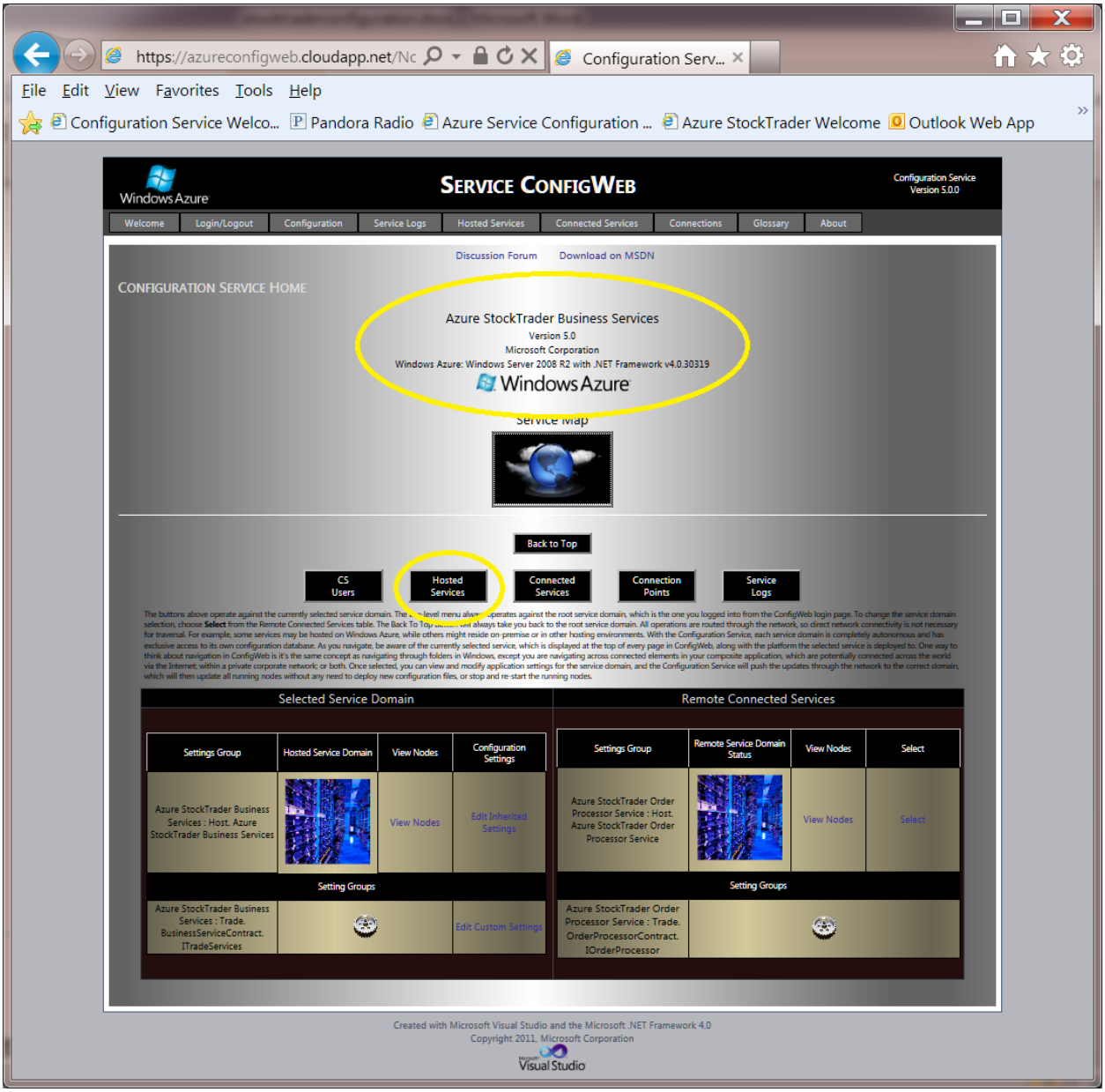
The StockTrader sample (both for Azure and .NET-on premise deployments) also has support for using **Client Certificate Credentials** instead of the default Client User Name Credentials discussed above. In this case, we have the configuration service repositories already loaded with the appropriate endpoint definitions, defined using the Hosted Services pages in ConfigWeb. However, the Client Certificate credentials endpoint is marked as non-activated by default, since we are using User Name credentials. In this section we will show how to change the Business Service tier to require a Client (X.509) Certificate as client credentials from the Web application. The processing logic is the same pattern as for the User Name credentials above: a custom certificate validator class in the BSL (and the OPS) is configured via a WCF Service Behavior in web.config (or app.config for the self-hosts). This class inherits from a base class supplied by the Configuration Service. The base class has the .NET System.Security logic to perform certificate authentication via the **`System.Security.Cryptography.X509Certificates`** .NET Framework assembly.

The Business Service tier can only select one mechanism for authentication at a time over a given network scheme (in this case https). So we will need to **deactivate** the endpoint using User Name credentials first, before we **activate** the defined endpoint using Client Certificate credentials.

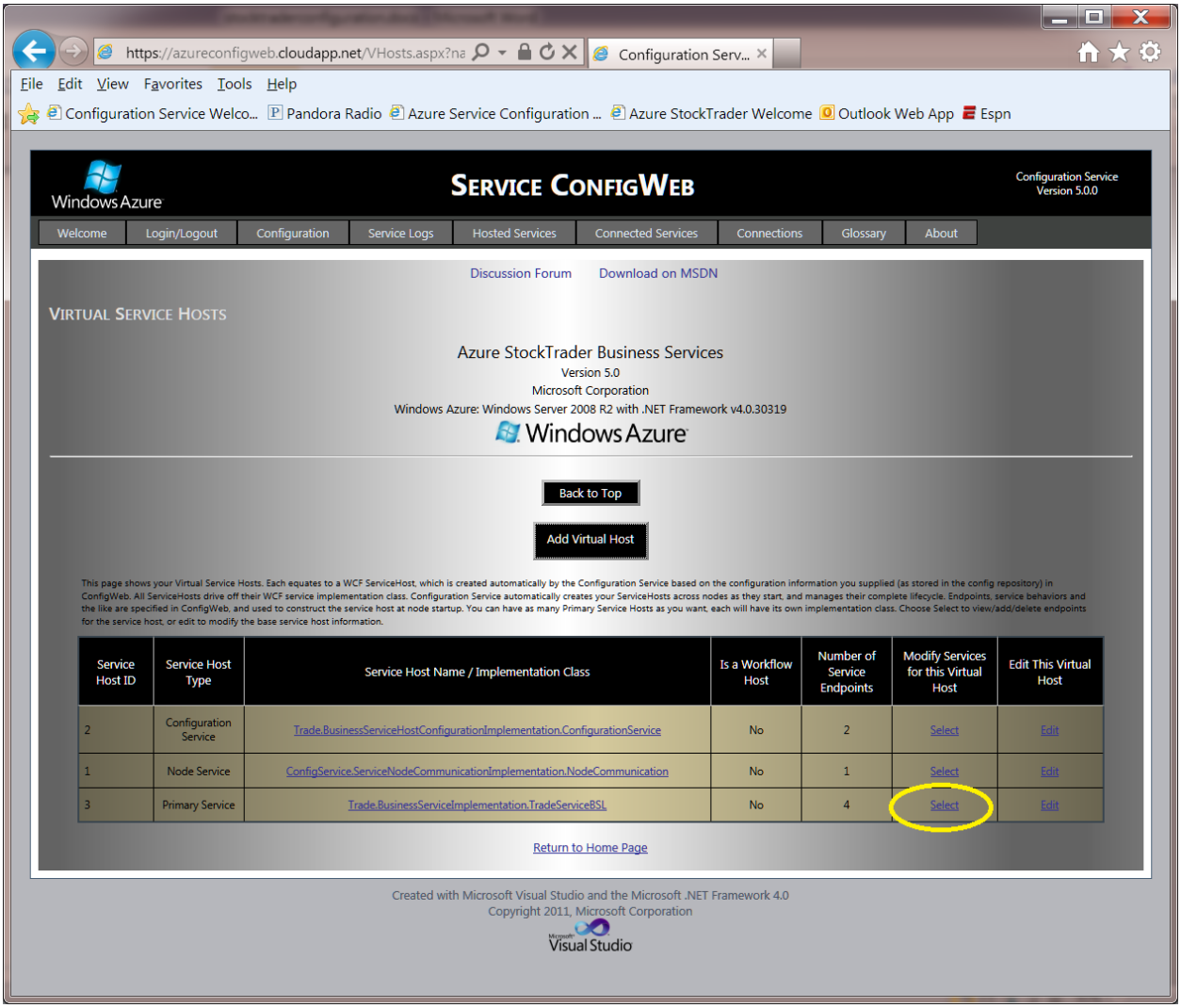
- Make sure you are logged into ConfigWeb against the **StockTrader Web** application Azure Hosted Service domain, as in the section above.



- From this home page, click on the **Select** link for the Business Services tier.

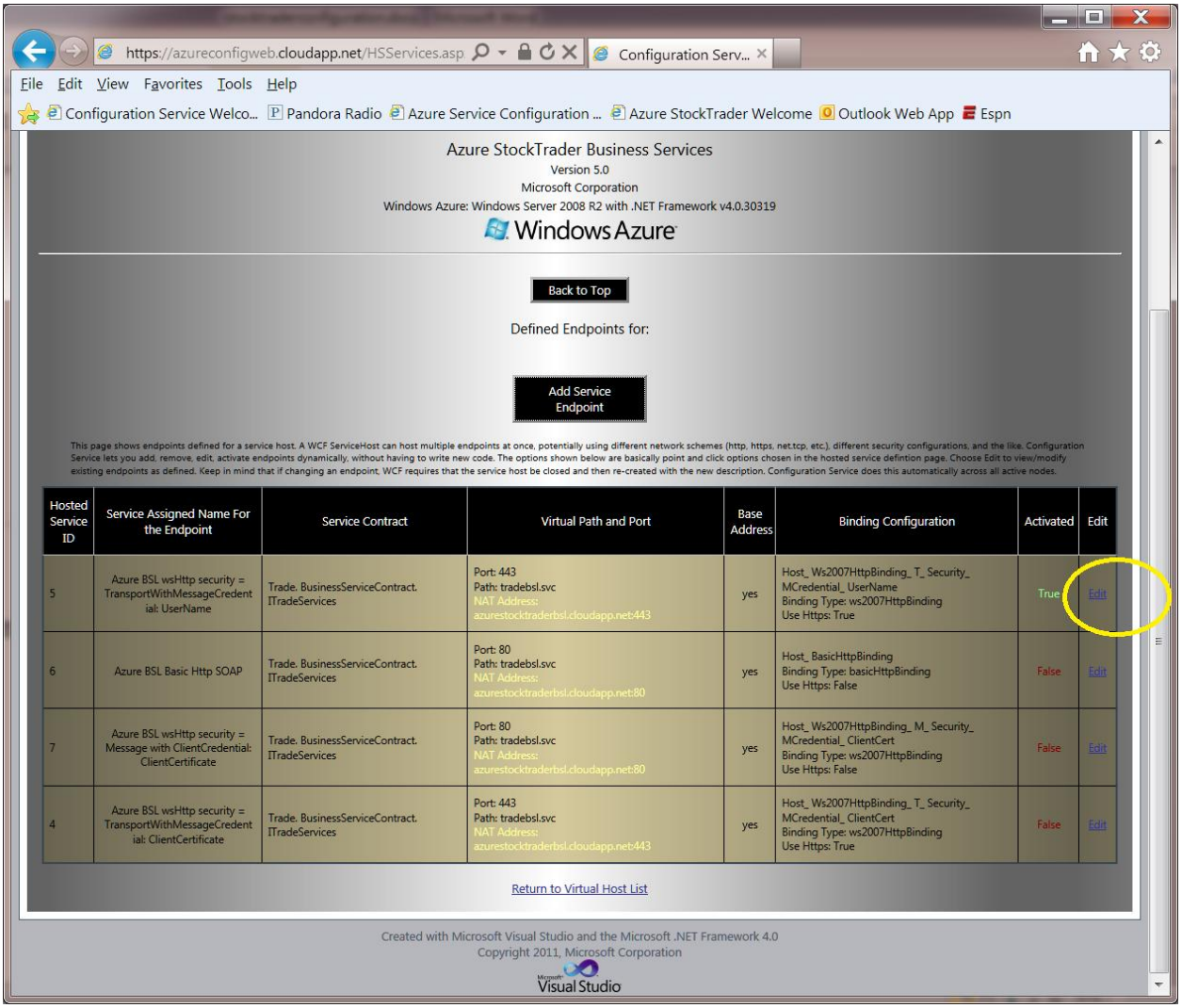


- With the Business Services service domain selected, click the **Hosted Services** button.



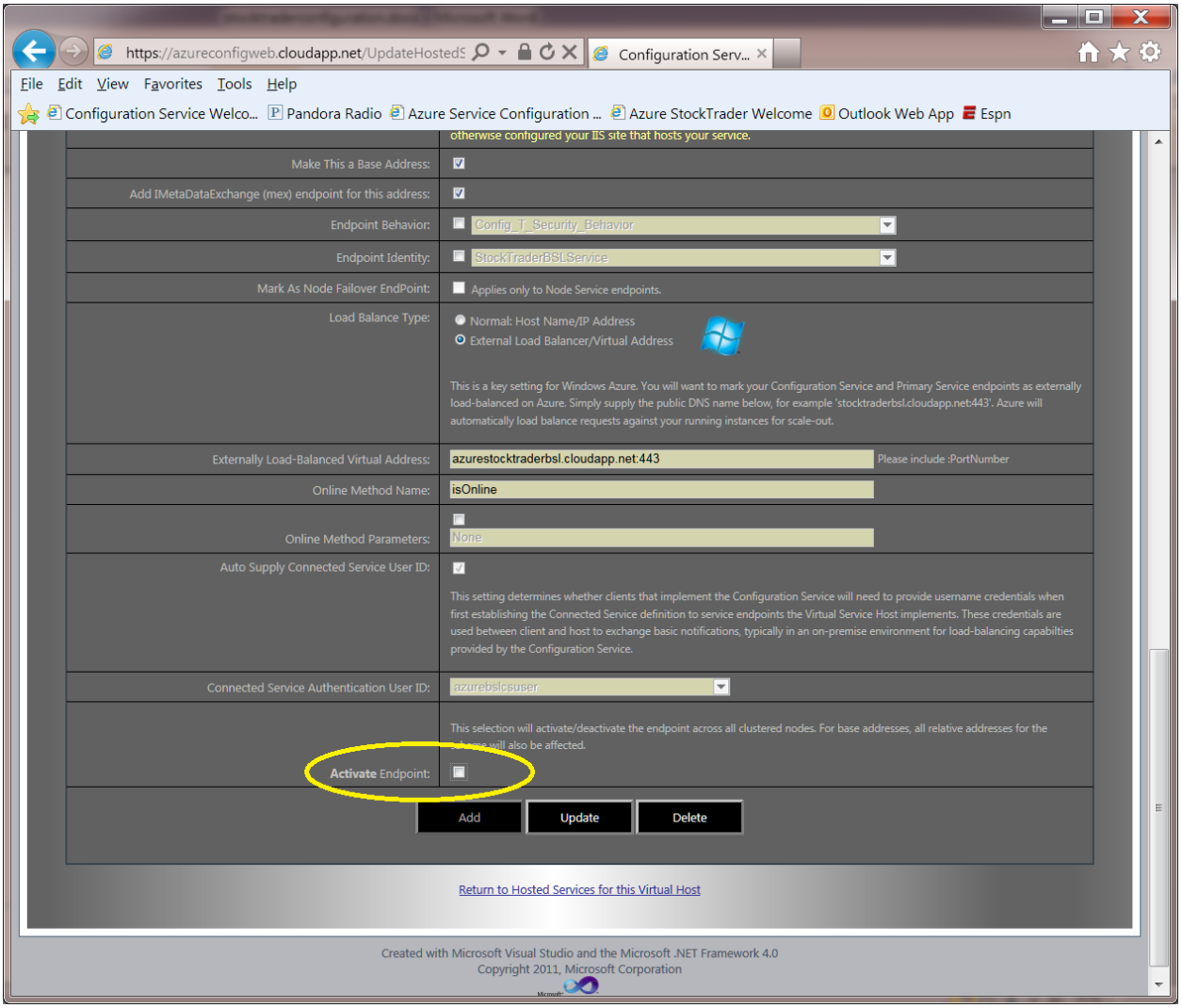
- Click the Select link for the **Primary Service** as shown above.

> **Note**
> Did you know you can create as many Primary Service hosts as you want with Configuration Service for a given service domain? The StockTrader Business Services and Order Processing Service only define one, but in custom implementations/other applications, you can create many, if desired, to host several different business logic implementation classes each with their own endpoints/security, etc. Service hosts can either be based on Web Services with Ws-* bindings, or be RESTful service hosts.

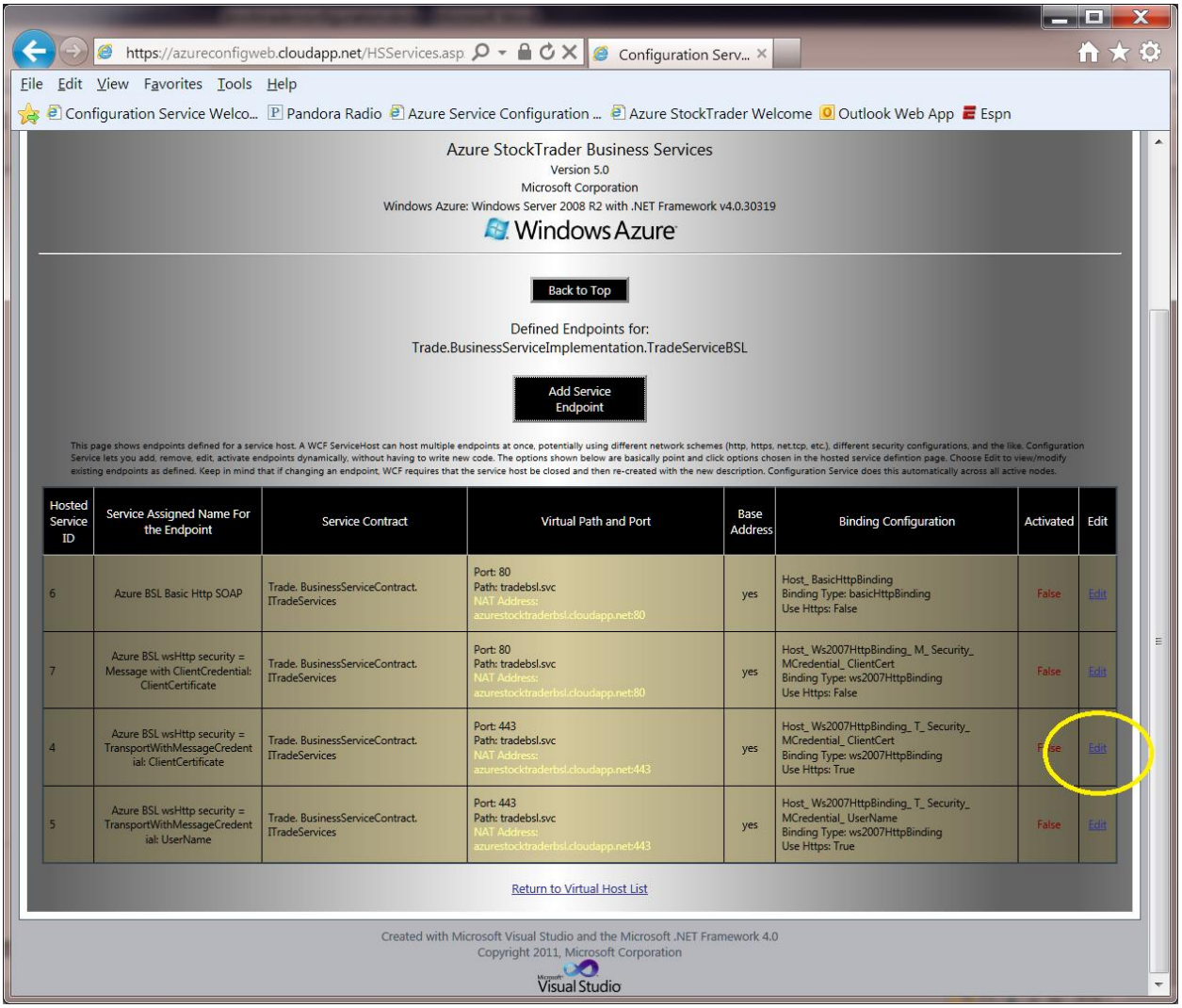


- You will notice the only endpoint currently active is the one using User Name client credentials, although several other endpoints are already defined in the configuration repository, they are in-active.

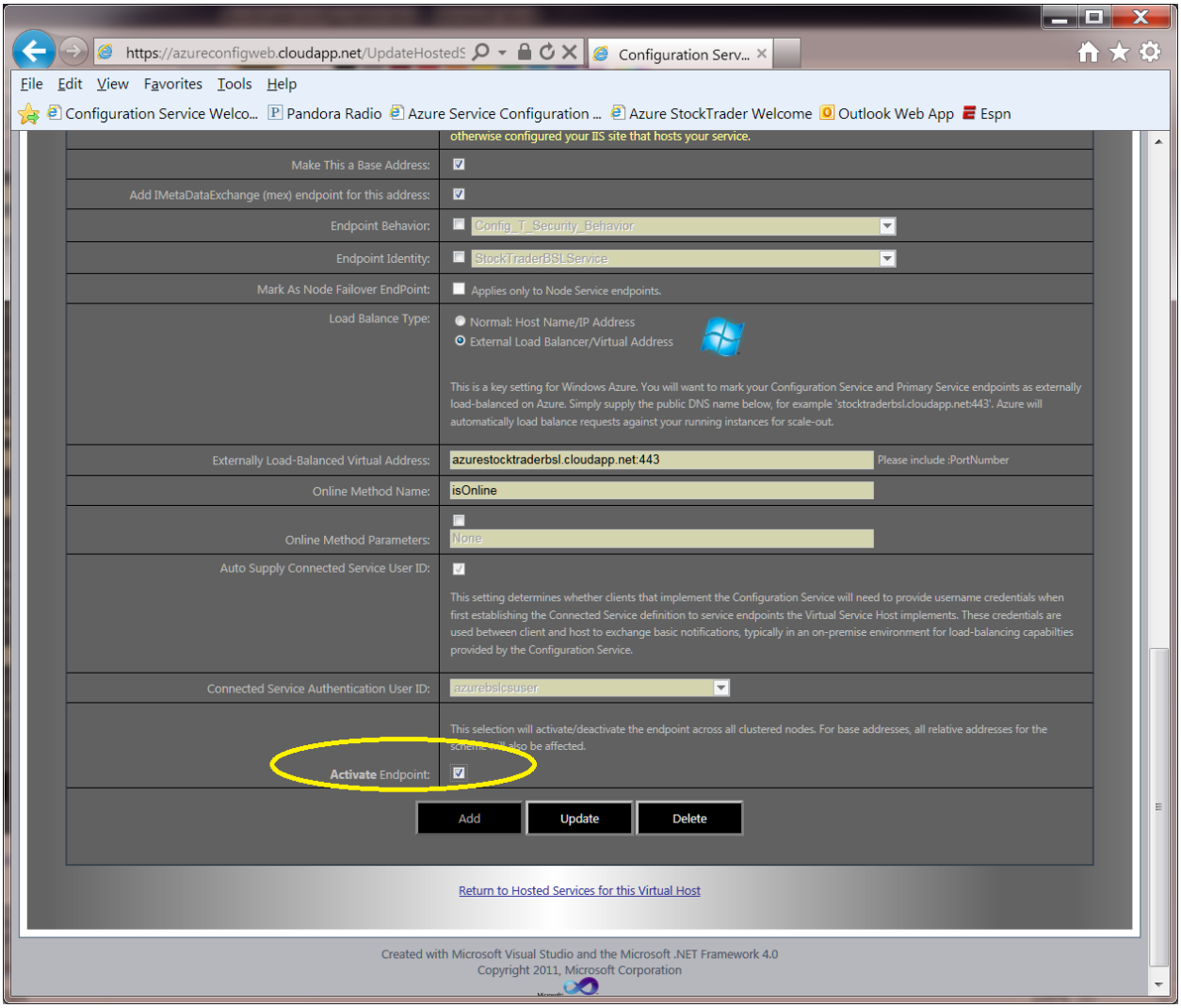
- Click on the **Edit** link for the only activated endpoint (with User Name credentials over https with a ws2007HttpBinding being used).



- We will deactivate this endpoint by de-selecting the **Activate Endpoint** checkbox.
- Click the **Update** button to process the change.
- Click on the **Return to Hosted Services for this Virtual Host link** at the bottom of the page.



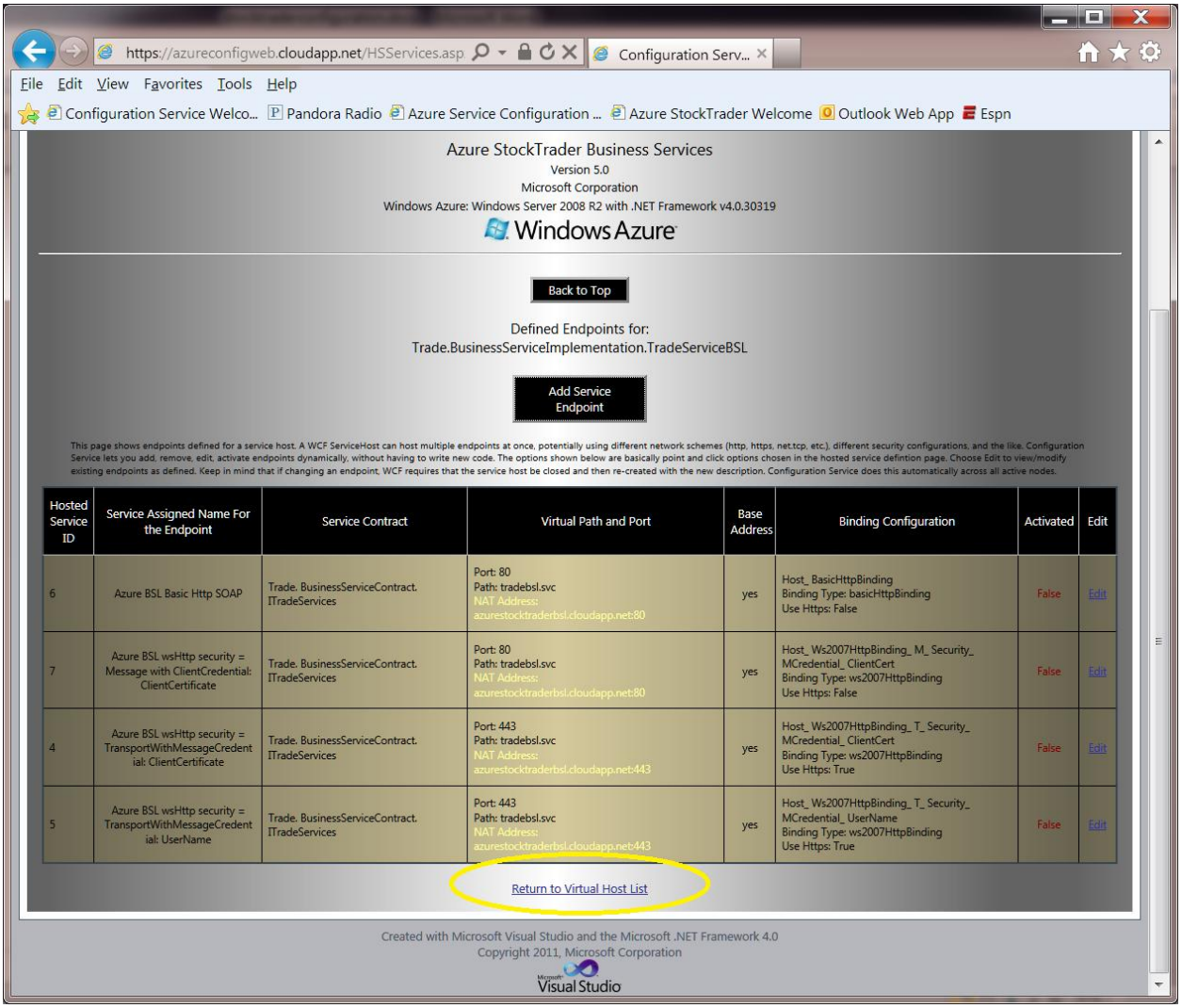
- Select the Edit link for the hosted service endpoint named: **Azure BSL wsHttp security = TransportWithMessageCredential: ClientCertificate.**



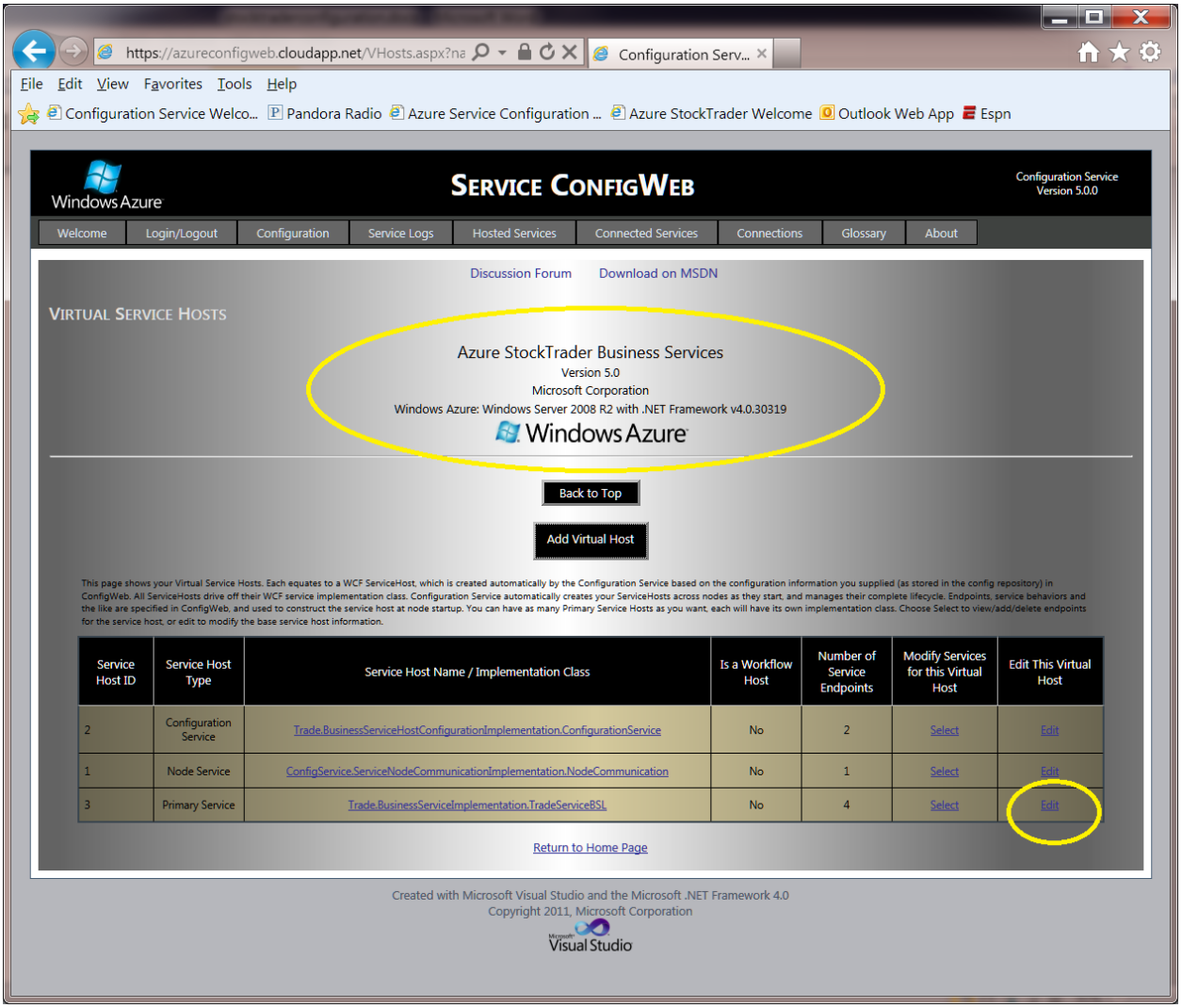
- Check the **Activate Endpoint** checkbox. Click the **Update** button.

For the StockTrader Business Services, we are not quite done yet. Right now, the new endpoint is activated, which requires client certificate credentials as we want. However, we need to configure the Service Host hosting this Primary Service endpoint to use the correct Service Behavior, which tells the application a client certificate will now be required, and what validator to use to validate the client certificate supplied on requests.

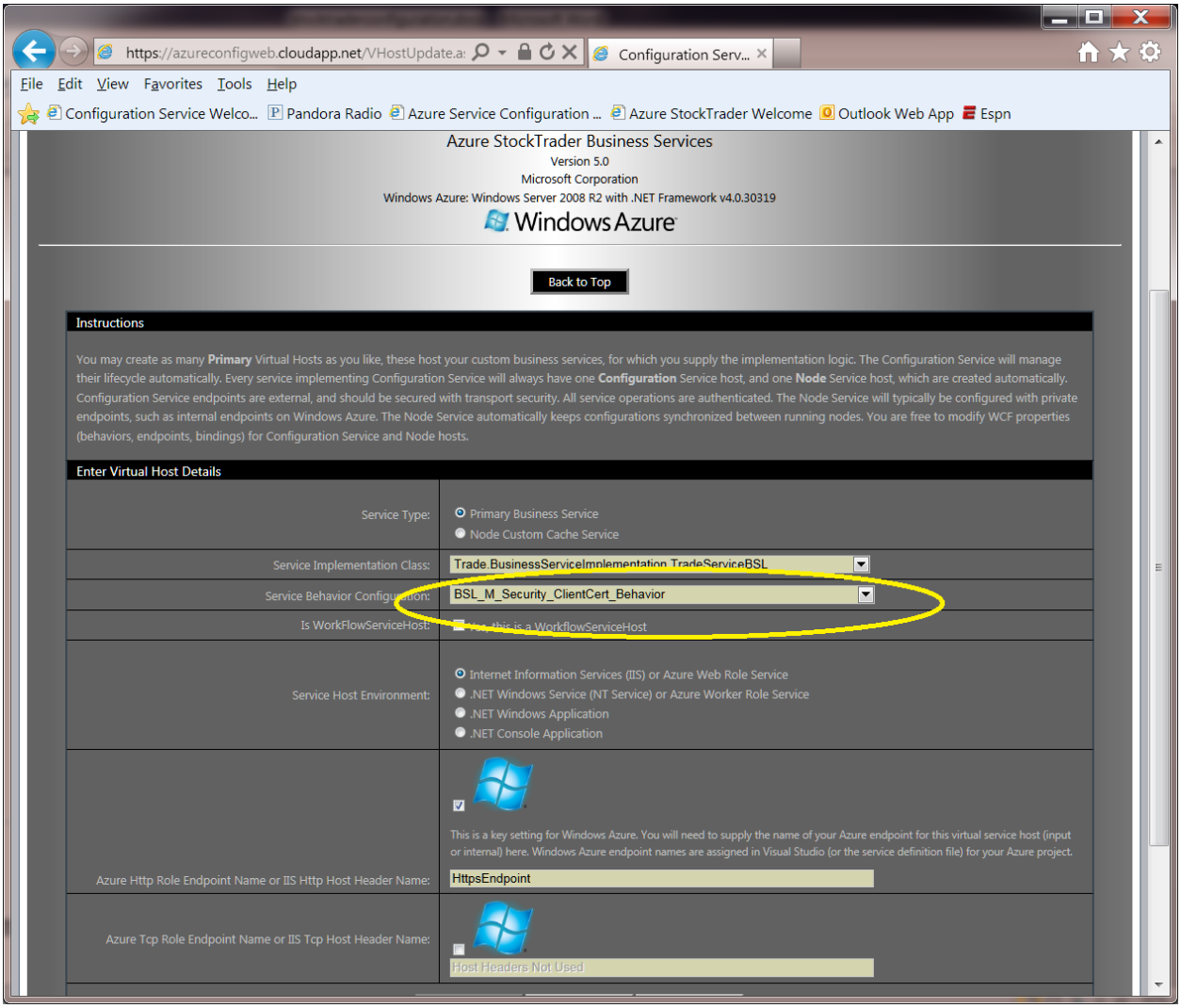
- Click on the **Return to Hosted Services** link at the bottom of the page above.



- Now click on the **Return to Virtual Host** list link as shown above.



- Now click on the Edit link as shown, for the Primary Service Host itself.



- Now, in the second dropdown (Service Behavior Configuration), select the **BSL_M_Security_ClientCert_Behavior**, as shown above.

- Click the **Update** button.

How the Client Certificate Gets Validated, and What Certificate is Valid as a Client Credential for the StockTrader Business Services?

Like the User Name validator discussed in the previous sections, Configuration Service has a base class that is a Custom Certificate Validator. This is tied to the ServiceBehavior in the Web.config for the StockTraderBSL Azure project (and also the on-premise project): it's in the \configuration\CustomValidators project within the Configuration Service Visual Studio Solution.

This base class is configured to validate based on several mechanisms, including the thumbprint of the certificate supplied by the client. We override a method in the StockTrader Business Service

implementation that inherits from the base class to provide the list of valid certificate thumbprints we will allow to use our BSL service operations. The class that does this is in the BSLCustomCertValidator.cs file in the BusinessServiceImplementation project. The code looks as follows:

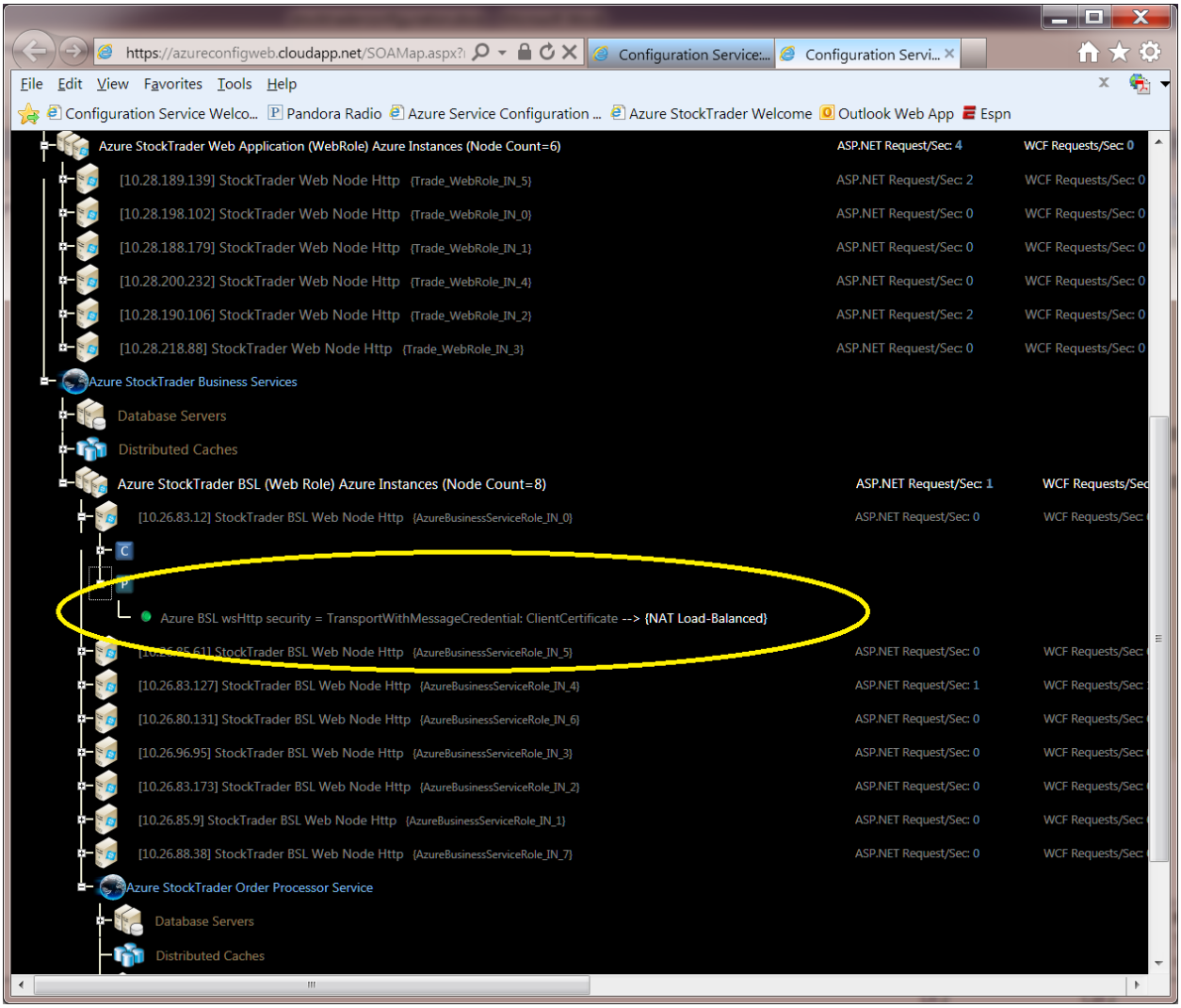
```
    /// <summary>
    /// The Business Services uses this custom X.509 certificate validator, that uses the base class
    /// provided with Configuration Service.  This class is referenced in the config file, with the
    /// BSL_M_Security_Behavior behavior configuration for the host exe.
    /// </summary>
    public class CustomCertValidator : CustomCertificateValidator
    {
        /// <summary>
        /// Override to provide our list of valid cert thumbprints for the service.
        /// </summary>
        /// <returns></returns>
        protected override string[] getAllowedThumbprints()
        {
            List<string> thumbprints = new List<string>();

            //This is the thumbprint for the Test Certificate CN=StockTraderBSLClient.Com
(StockTraderBSLClient.pfx).
            //This Cert is used by the Business Service Layer to validate the StockTrader Client. When
using this binding in
            //conjucntion with the service behavior 'BSL_M_Security_ClientCert_Behavior' (which also
assigns the service certificate),
            //only clients presentin the EXACT client certificate will be allowed in.  The thumbprint is
unique, meaning,
            //no one can create another cert with this thumbprint.  Hence, only an app with this exact
client-side
            //cert can ever access the BSL when using this WCF message security behavior.
            thumbprints.Add("EB0C8C302C4F5E22E4492006F1D16D01008E7826");

            return thumbprints.ToArray();
        }
    }
```

The thumbprint is for the client certificate for the BSL that we already uploaded to the **StockTrader Web application** when installing Azure StockTrader. It's the **StockTraderBSLClient.Com** certificate, contained in the \certs\ StockTraderBSLClient.pfx certificate file.

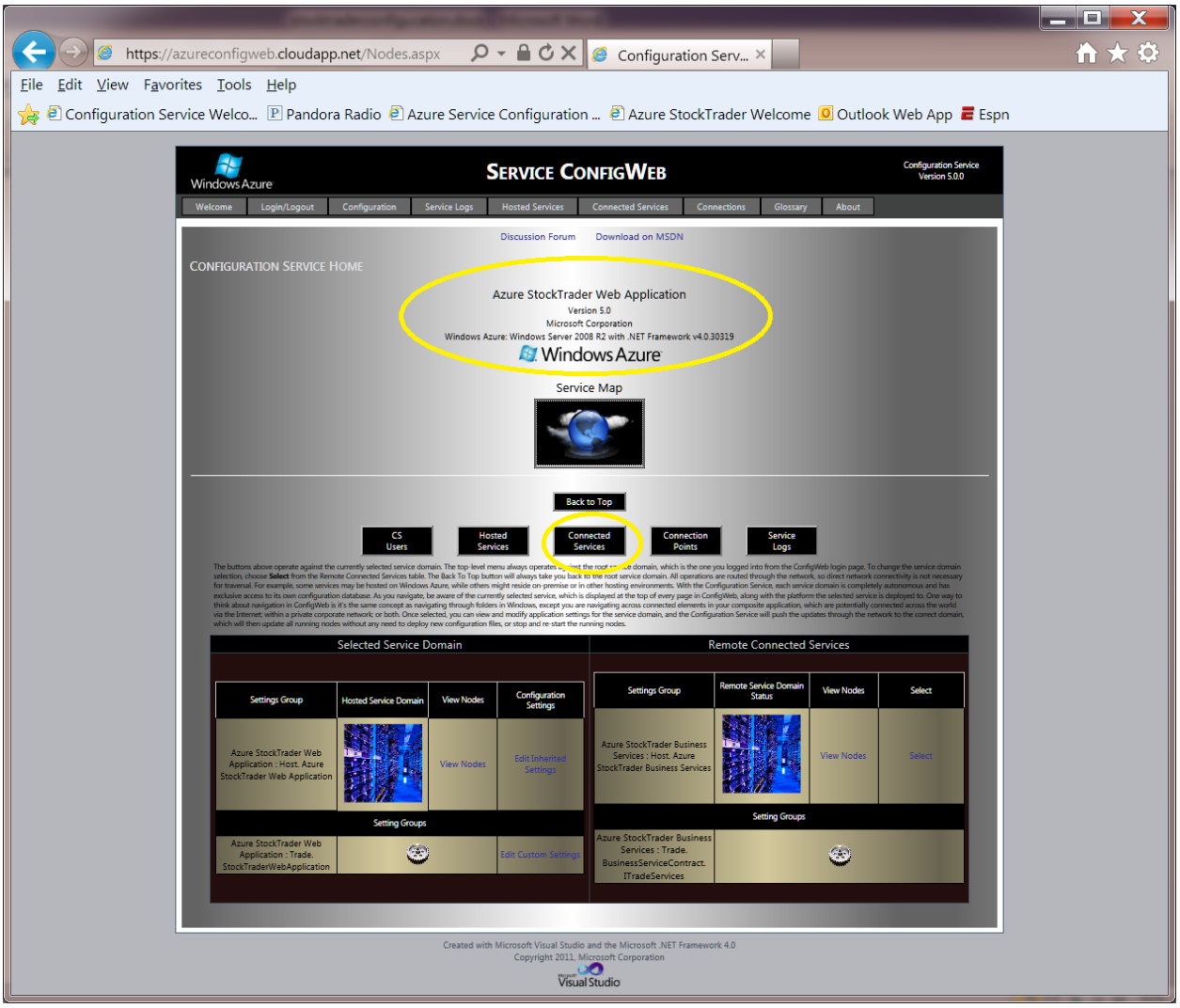
If you go back to the home page of ConfigWeb, and click the Service Map button, you will see the new endpoint is now active and online (green), and the old one using User Name is not active anymore.



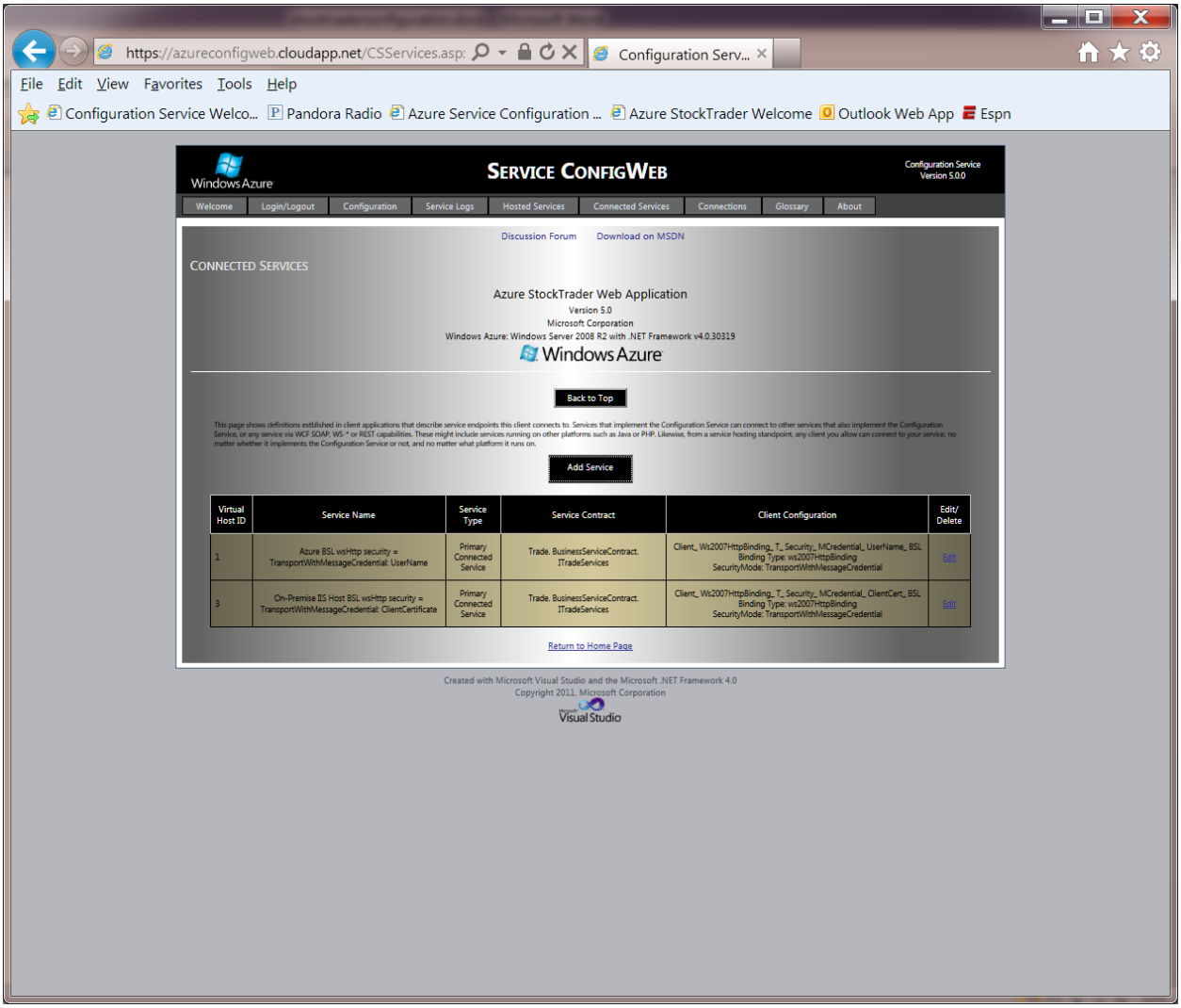
Note that the Hosted Service endpoint for the BSL is already configured with an internal client configuration that allows the BSL nodes to check their own endpoints, in this case using the correct client certificate which we uploaded to the Business Service domain via the Azure Management portal when we first created the hosted service during StockTrader setup. The Service Map page works by requesting remote services to validate their own endpoints for each node (in this case Azure role instance) running. BUT: the StockTrader Web application is not yet configured to use a WCF endpoint requiring a client certificate. So we next need to create a new Connected Service Definition for the **StockTrader Web** application that will use this service endpoint, instead of the User Name authenticated endpoint as installed by default.

Changing the StockTrader Web Application to Supply a Client Certificate to the StockTrader Business Services.

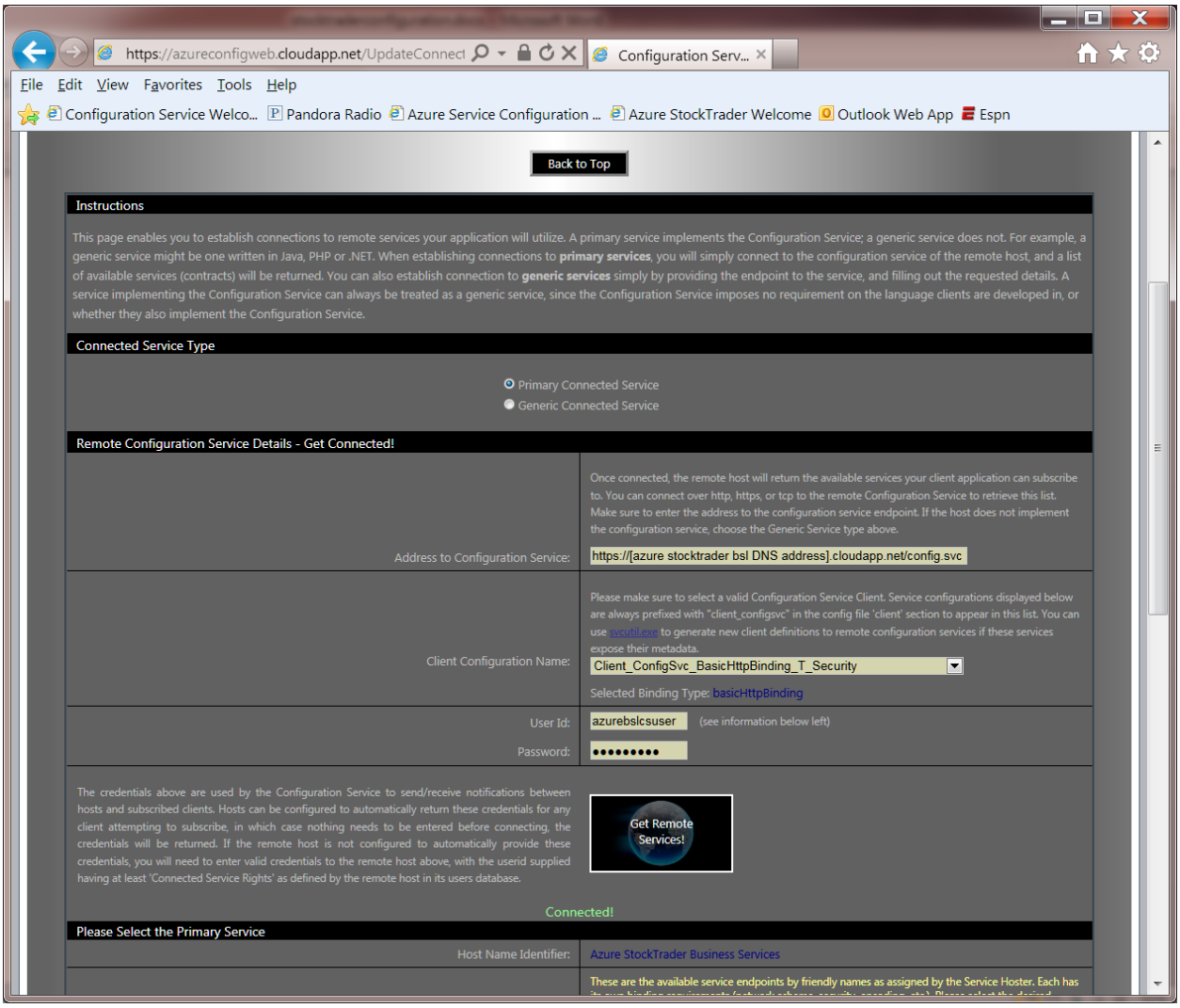
- Click on the Configuration menu top-level menu item to return to the root StockTrader Web application home page for ConfigWeb.



- Click on the **Connected Services** button.



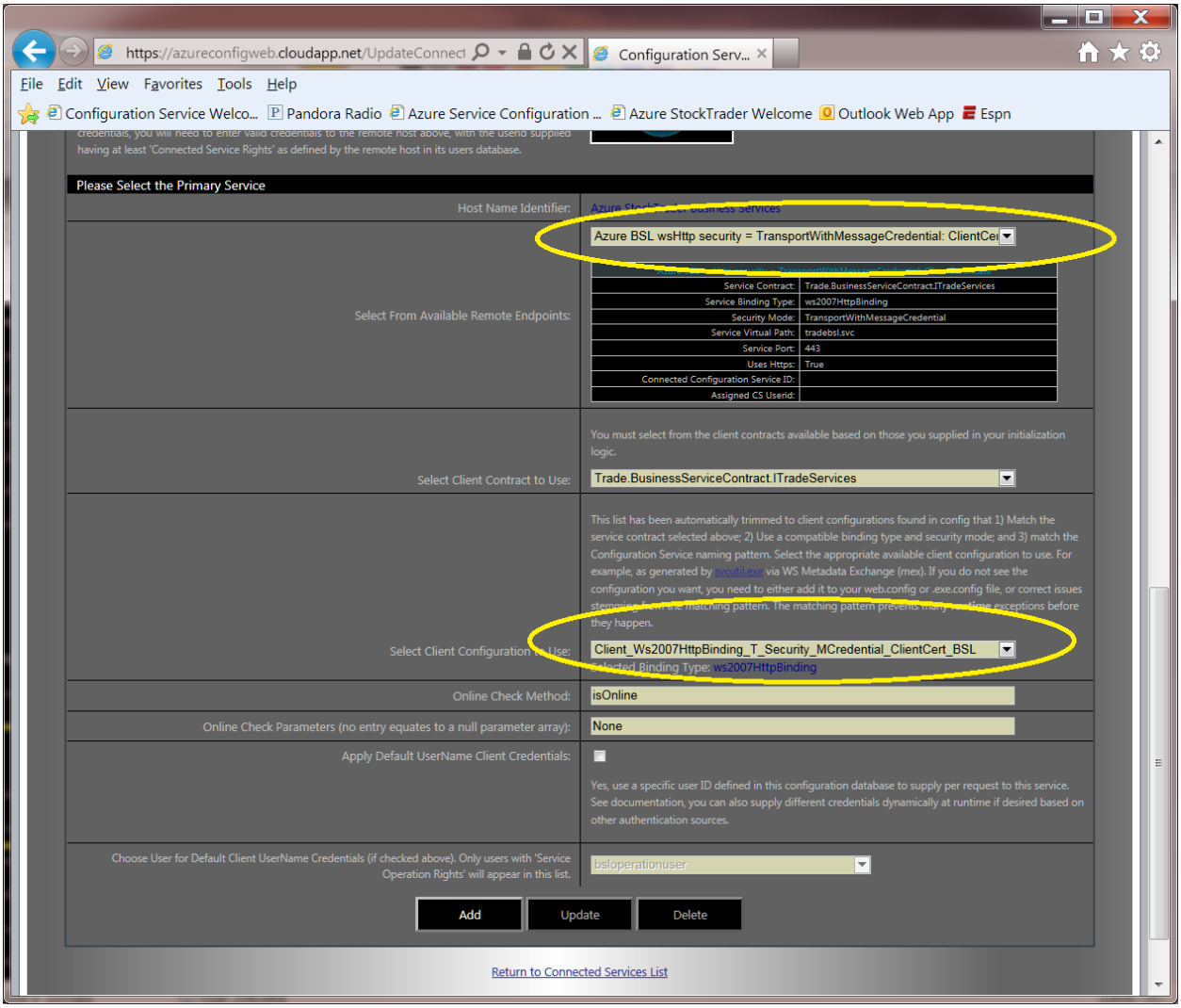
- Click on the **Add Service** button.



We will now connect up to the BSL's Configuration service to get the new endpoint information and create our new Connected Service definition.

- Type in the address to your Azure hosted service domain for the Business Services project you deployed.

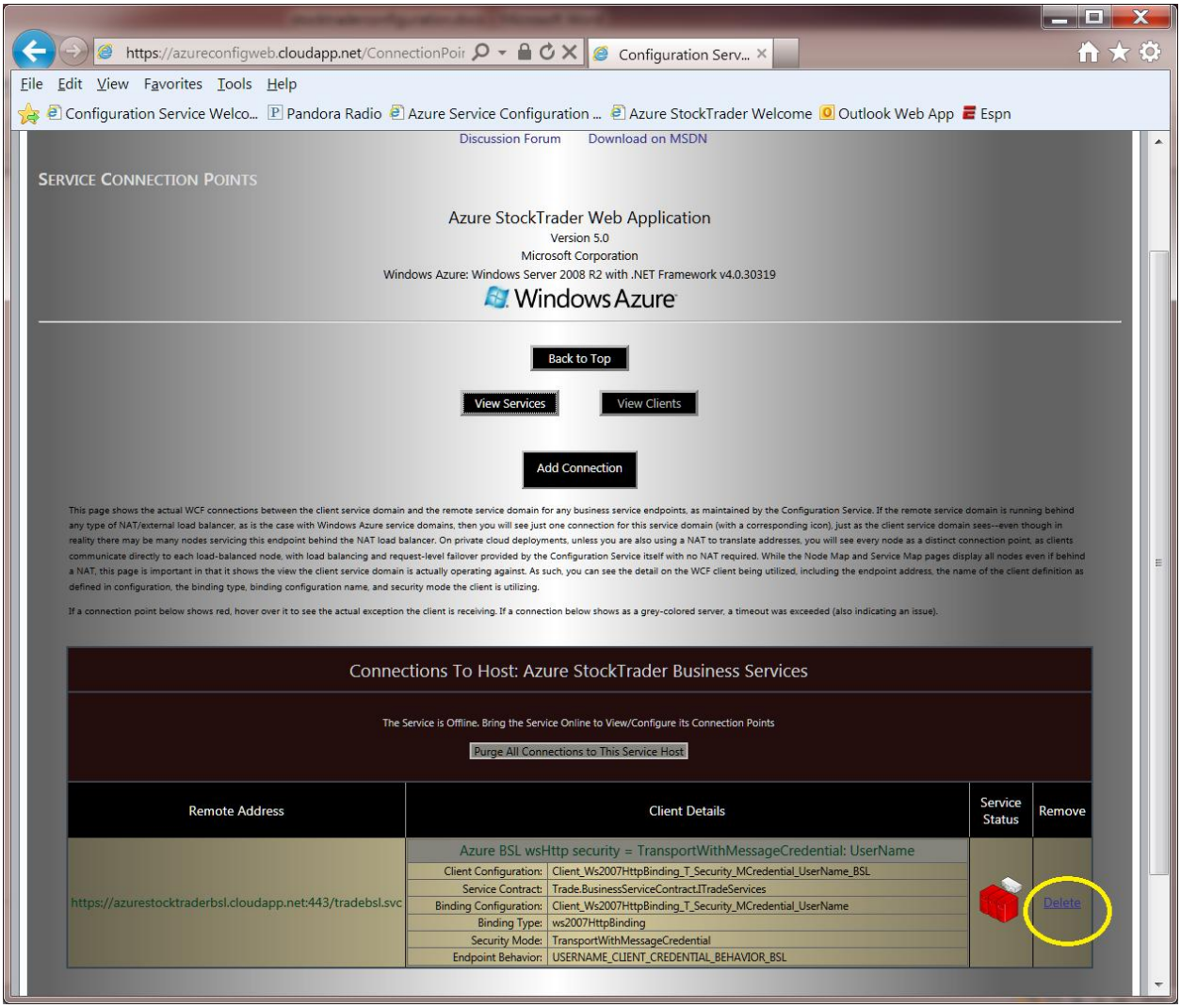
- Choose the **Client_ConfigSvc_BasicHttpBinding_T_Security** client configuration, and click on the **Get Remote Services!** button.



- The BSL has only one active endpoint, but you need to make sure to select the correct client configuration: **Client_Ws2007Binding_T_Security_MCredential_ClientCert_BSL**.

- Click the **Add** button.

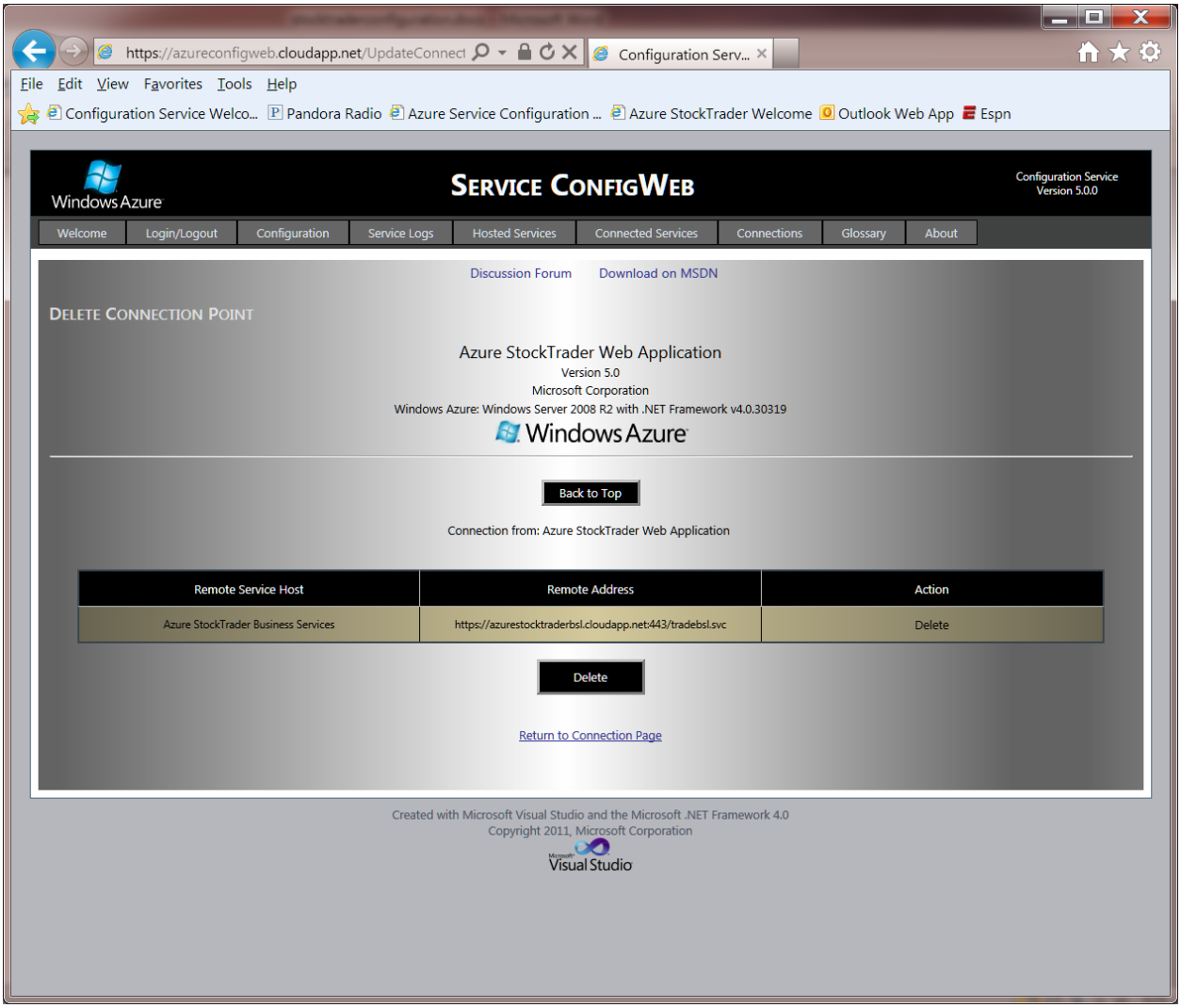
- Next, click on the Connections top-level menu item.



You will notice we still have a Connection Point defined to the old, inactive User Name endpoint. Since the BSL is not hosting this anymore (its deactivated), it shows as red from the client connections page.

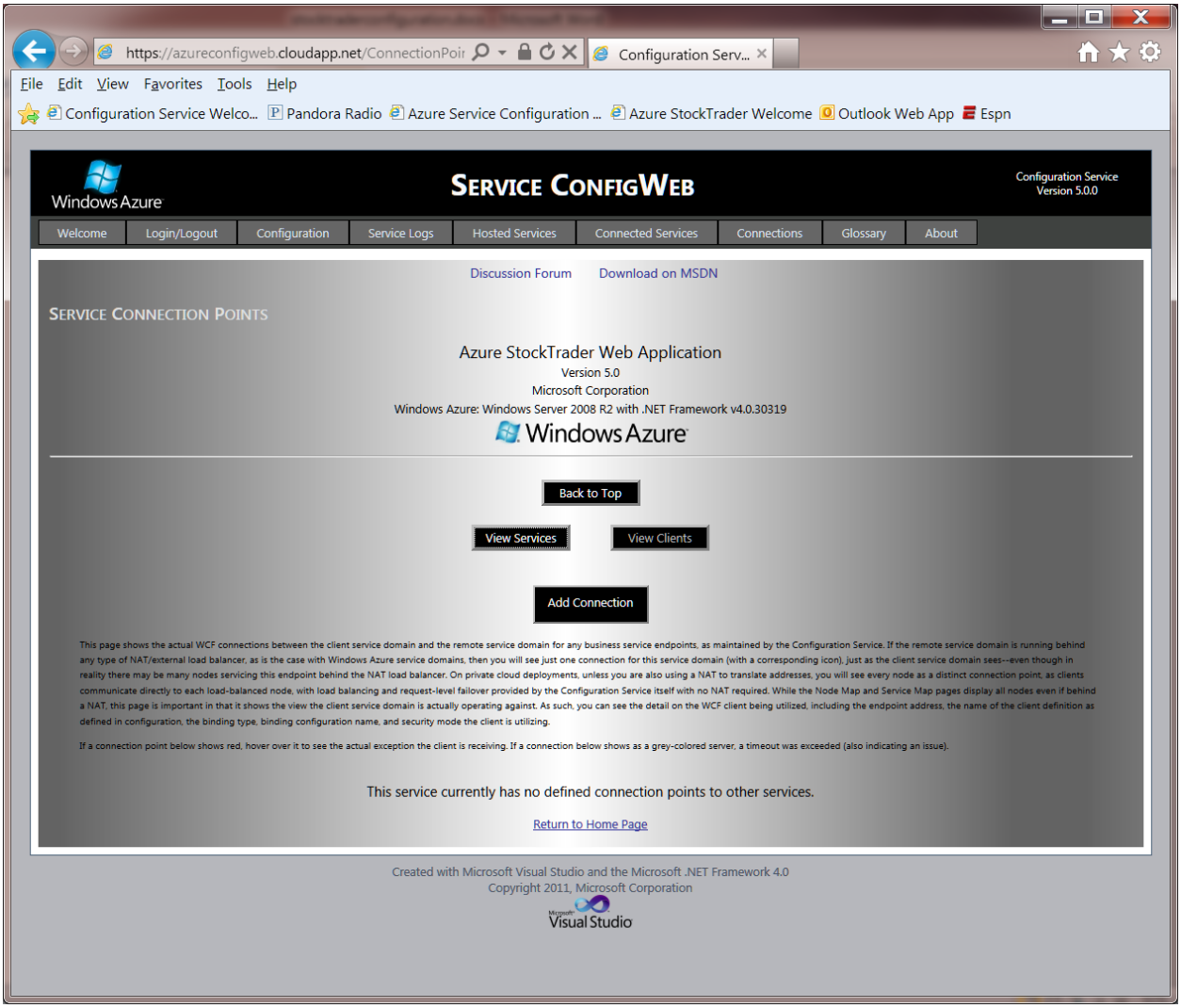
We can delete this non-active connection (it does not hurt anything to leave it be, since we will not be using it, but to keep things clean, we will get rid of it).

- Click on the Delete link, and go through the next page to delete the old connection point.

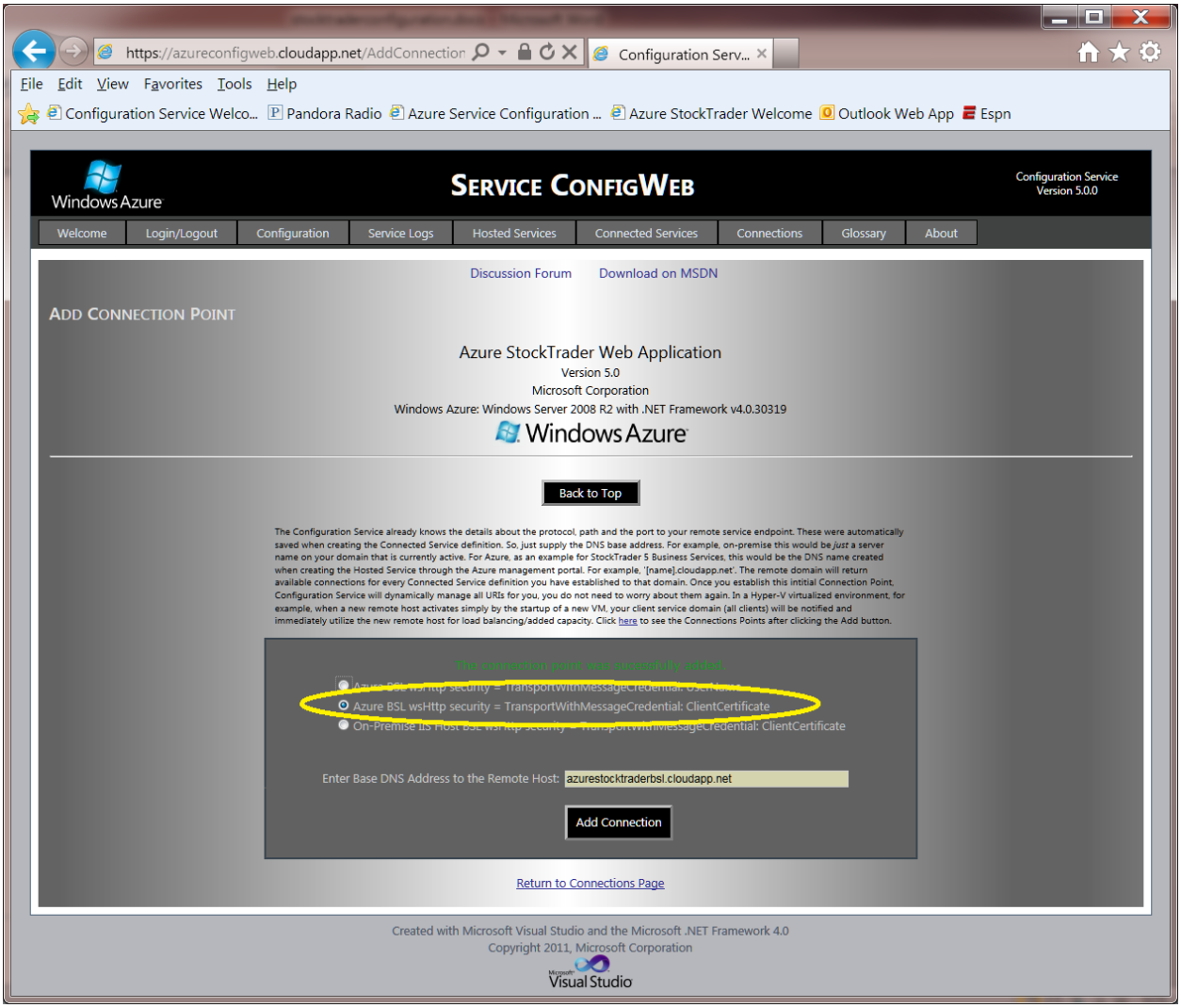


- Click on the Delete button.

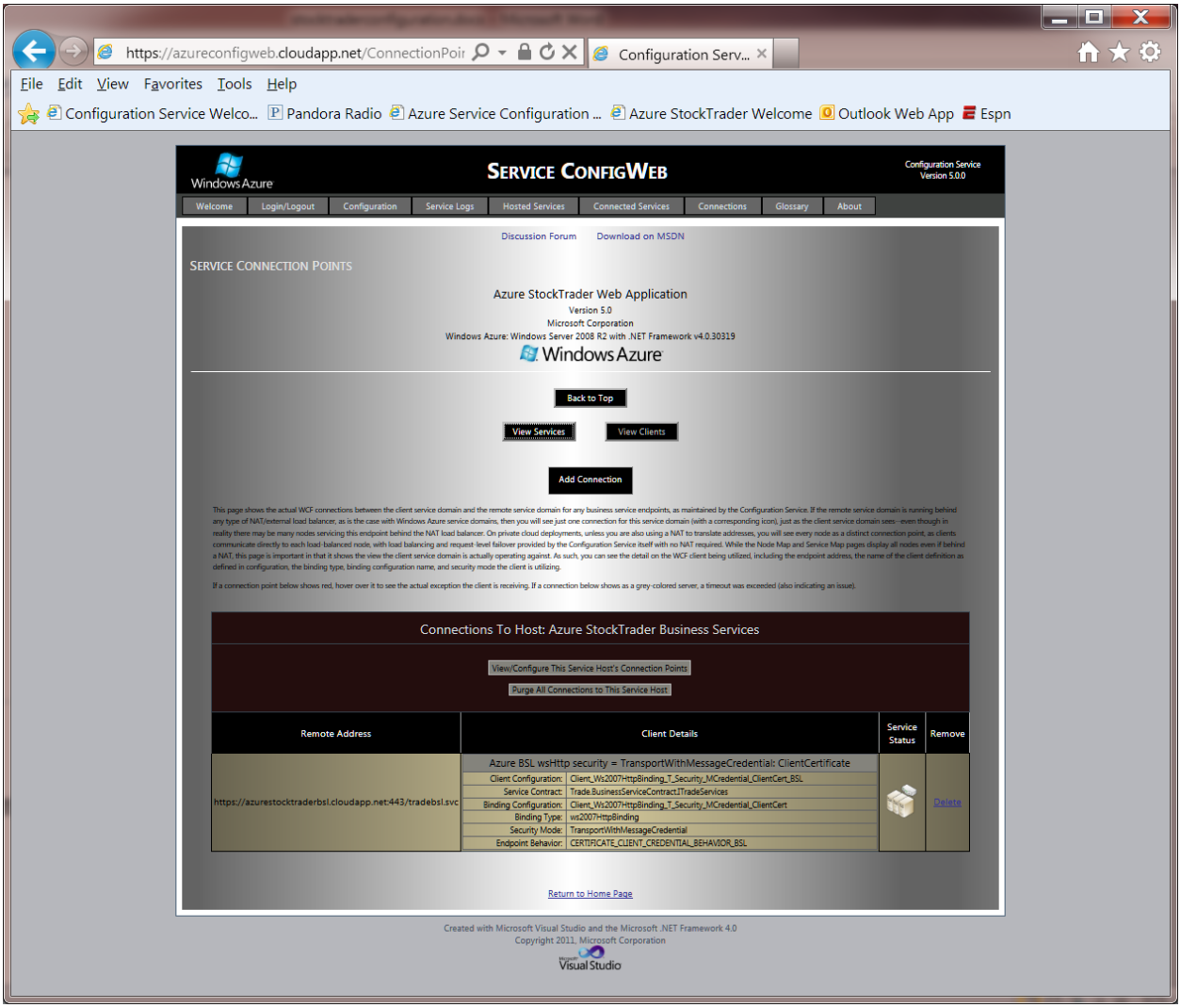
- Click on the Return to Connection Page link at the bottom of the page.



- Click on the **Add Connection** button.



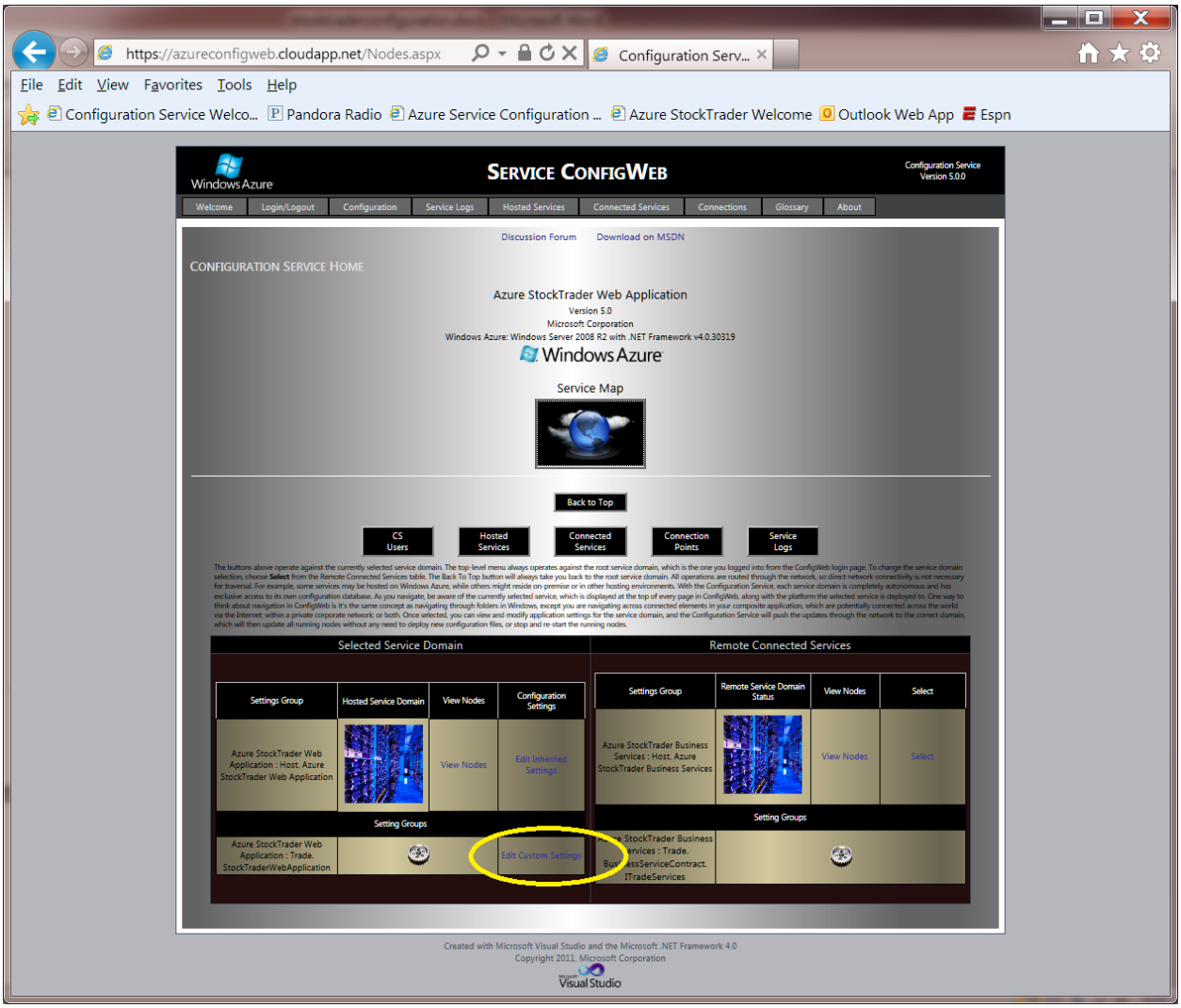
- Select the newly created Connected Service Definition: **Azure BSL wsHttp security = TransportWithMessageCredential: ClientCertificate**
- Enter the base DNS address to **YOUR** Azure StockTrader Business Services Azure Hosted Service.
- Click the **Add Connection** button.
- Click the **Return to Connections** link.



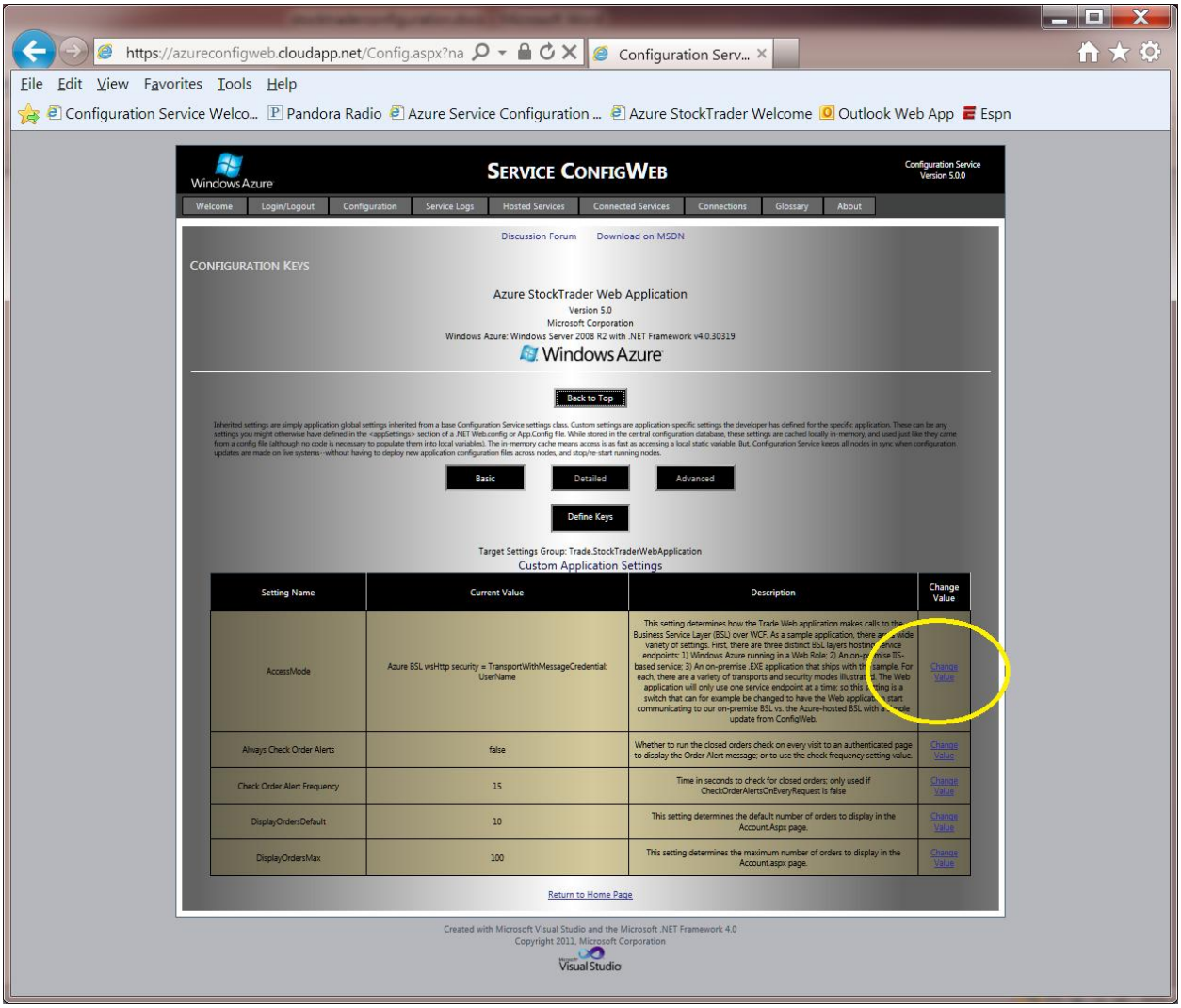
- We can see from this page that the StockTrader Web application is seeing the endpoint as online, and hence the client certificate is working correctly for authentication and being passed by the client to the service. The reference to the client certificate StockTraderBSLService.Com has been already configured in the StockTrader Web's client configuration we selected previously, and this client configuration is contained in the StockTrader web application's Web.config (so it knows where to find the client certificate to pass to the WCF BSL service).

- However, the Web application is still configured to use the other hosted service, via its AccessMode setting. Notice the title text for the connection is not green above (its black text), meaning the client connection exists, but it's not configured to be used by the Web app via the AccessMode setting. So the last thing we need to do is change the AccessMode setting so the Web application uses this endpoint/connection point(s).

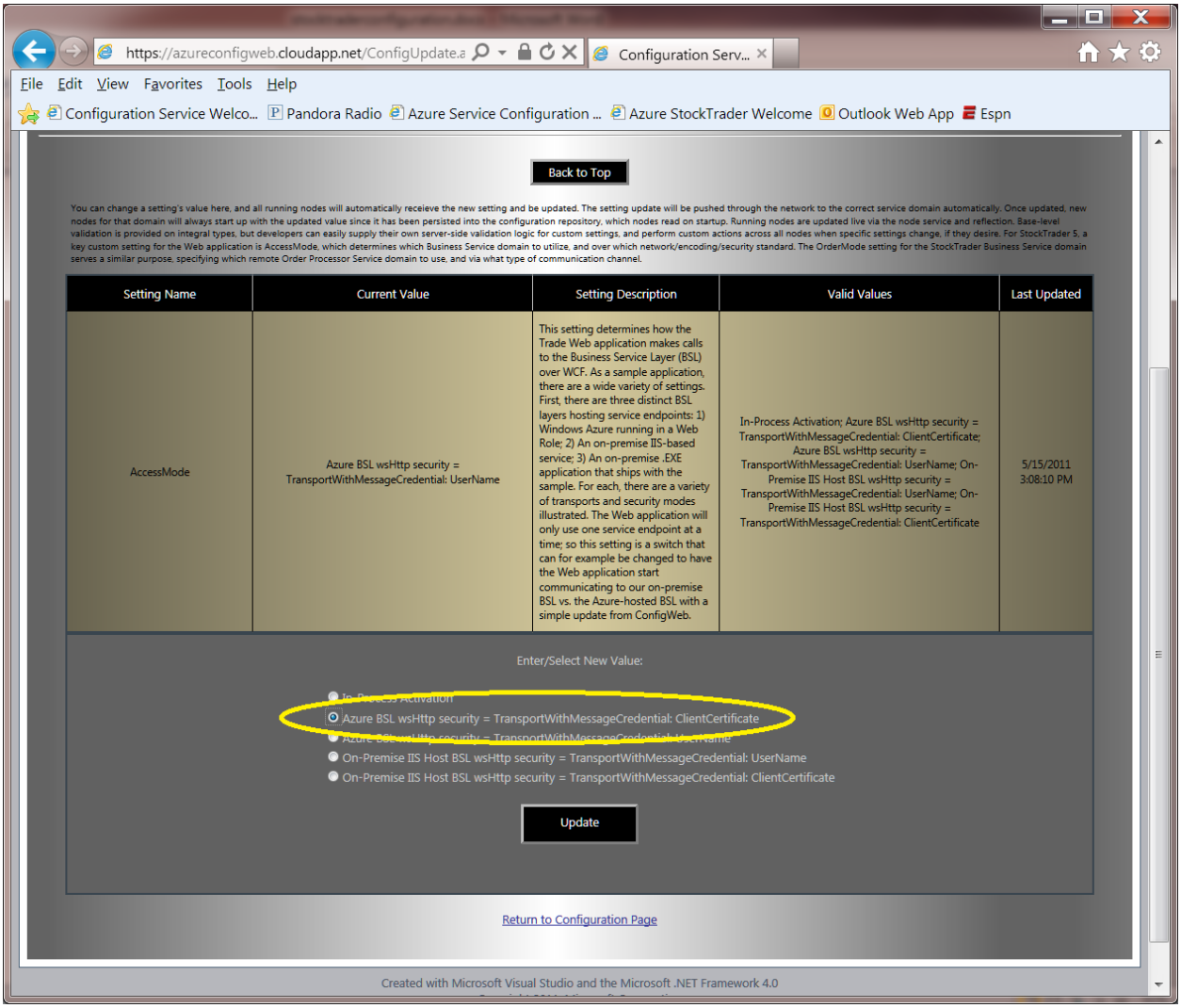
- Click on the Back to Top button to get back to the home page for ConfigWeb, with the Web application selected.



- Click on the Edit Custom Setting link for the Web application.



- Click on **Change Value** for the AccessMode setting.



- Select the '**Azure BSL wsHttp security = TransportWithMessageCredential: ClientCertificate**' option.

- Click the **Update** button.

OK! At this point, we have completely changed both the service and client, so now the StockTrader Web Application is making all service requests to the Business Services via an endpoint requiring a client-side X.509 certificate. User Name credentials are no longer being used. You can now login as an end-user to the Azure StockTrader web application. End users will not notice a difference, but you can verify the application is working in its new configuration.

Changing the Samples to Use Different Certificates vs. Default Install Certificates

Now that we have seen how the X.509 certificates are used, the next logical question is how to change them from the ones provided in the default install? After all, everyone using the default install will have the same certificates, so anyone could access our services when running in Client Certificate mode as set above.

To change certificates, you need:

- To create or obtain your own certificate. There is a batch file for creating .pfx self-signed certificates installed in the \certs folder with the StockTrader certificates for this purpose.

- You need to upload these certificates to the correct Azure Hosted Service domain(s).

- You need to modify the correct service and client behaviors in configuration to use these certificates as opposed to the default certificates. Simply do a global search for 'StockTraderBSLService.Com', for example, to find its uses in Config; but remember that 'StockTraderBSLClient.Com' will be found in both the Business Services .config files (app.configs and web.config); and the StockTrader Web app's web.config.

- You need to look at the method '`getAllowedThumbprints`' (see page 96) on the custom certificate validator and change this method to use your certificate(s) thumbprints.

These rules apply to both the StockTrader Business Services, and the Order Processor, since both can be set to use client certificates vs. Username client credentials.

Configuring the MSMQ Durable Message Queue Endpoint for Order Processing On-Premise StockTrader

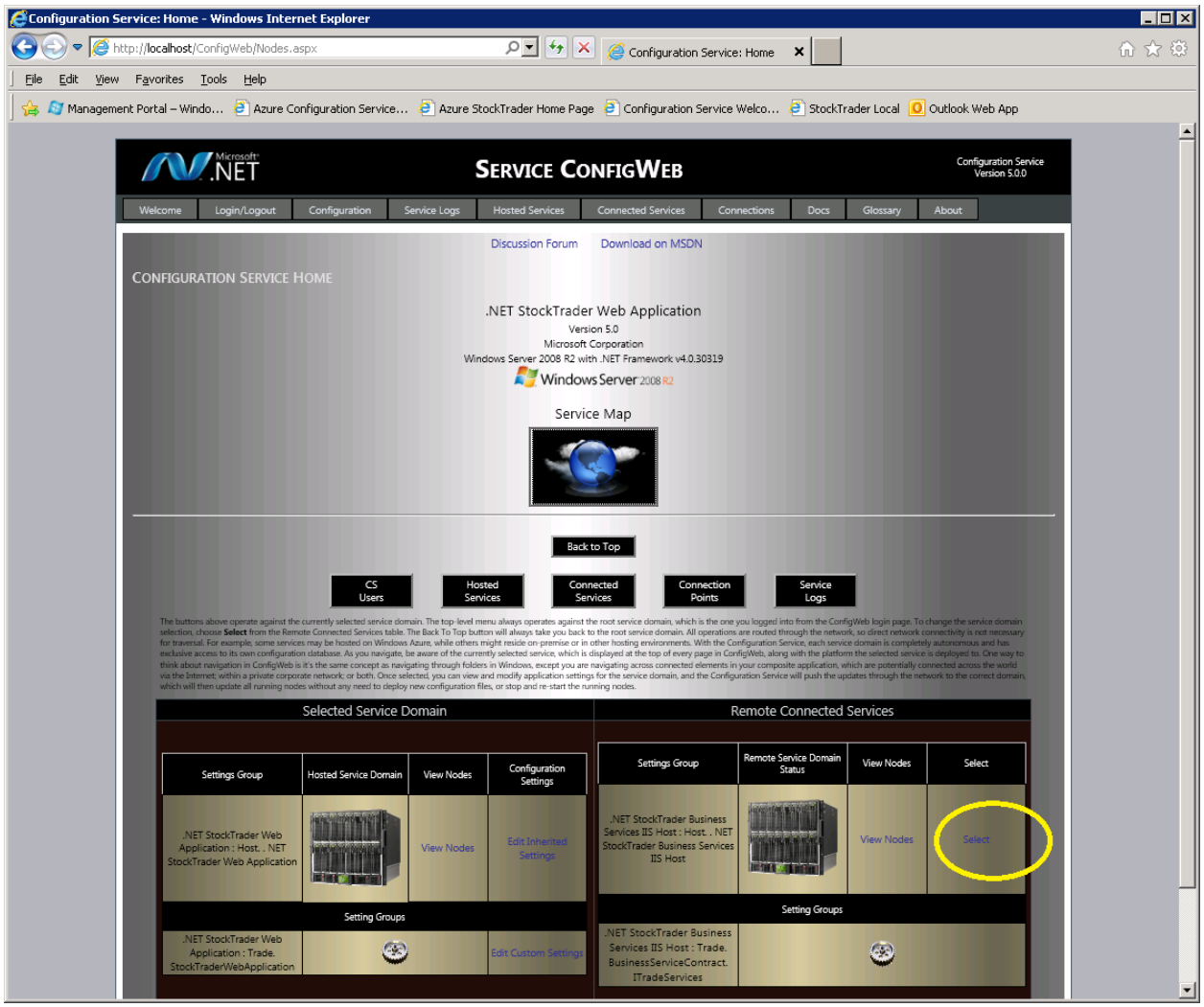
The following sections will step through an additional StockTrader configuration change using the Hosted Services pages in ConfigWeb. This topic relates only to on-premise .NET StockTrader, which uses MSMQ. For Windows Azure, the new Azure AppFabric Service Bus is bringing a cloud-optimized durable queuing mechanism to the party, but it is only in CTP right now (as of June 9, 2011, it will be released commercially later but you can sign up for the CTP if you want).

So we will use on-premise StockTrader in this walkthrough, not Azure StockTrader.

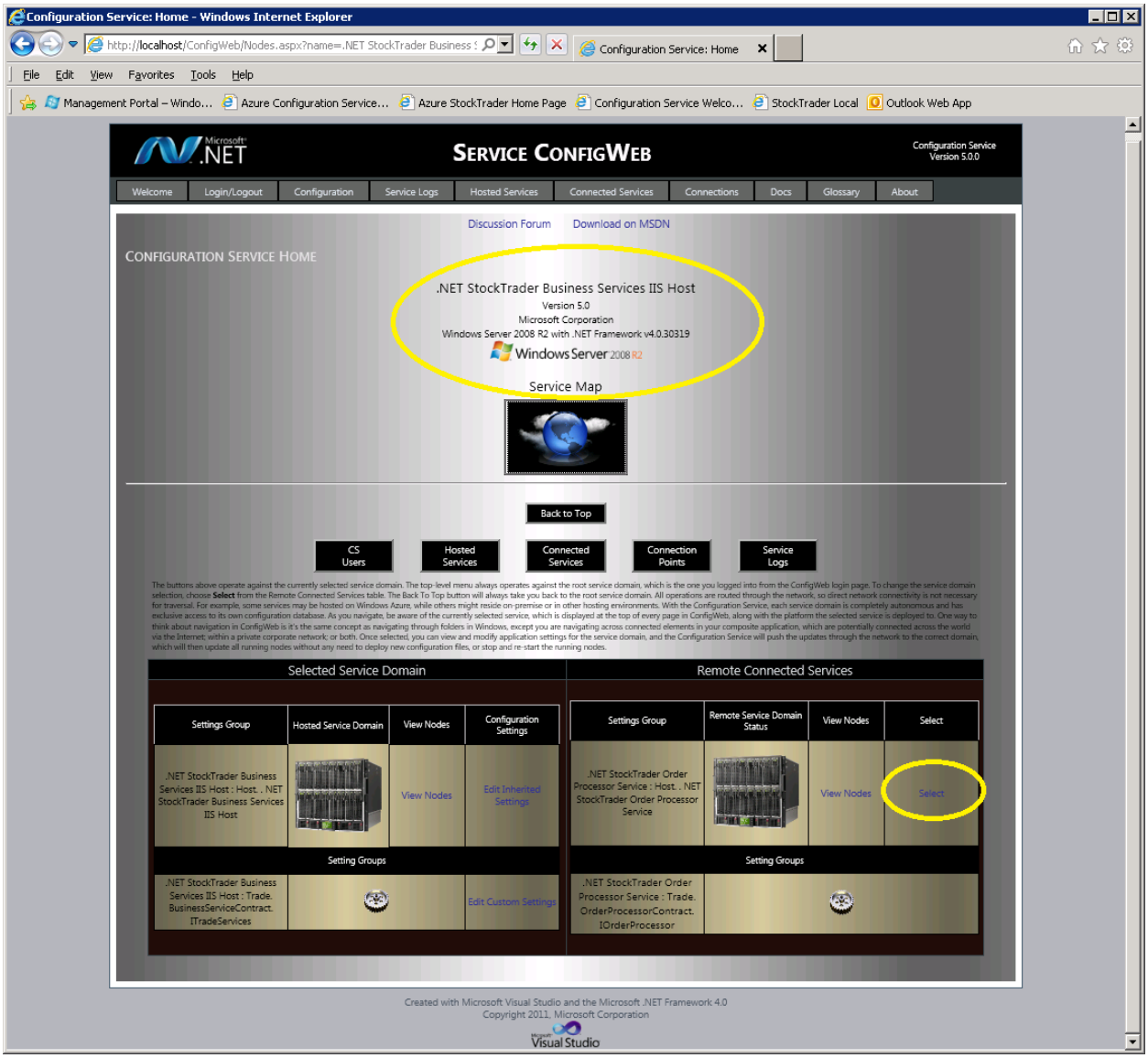
Activating and Using the Order Processor Transacted MSMQ Endpoint

By default, the OPS MSMQ endpoint is marked as de-activated in the OPS Repository. We can activate this MSMQ endpoint, which operates against a durable, transacted MSMQ in conjunction with a WCF service endpoint. The following section assumes you have StockTrader configured (or Azure StockTrader) for remote modes with OrderMode set to the net.tcp or http endpoint, and valid connections established to the OPS as covered in previous steps.

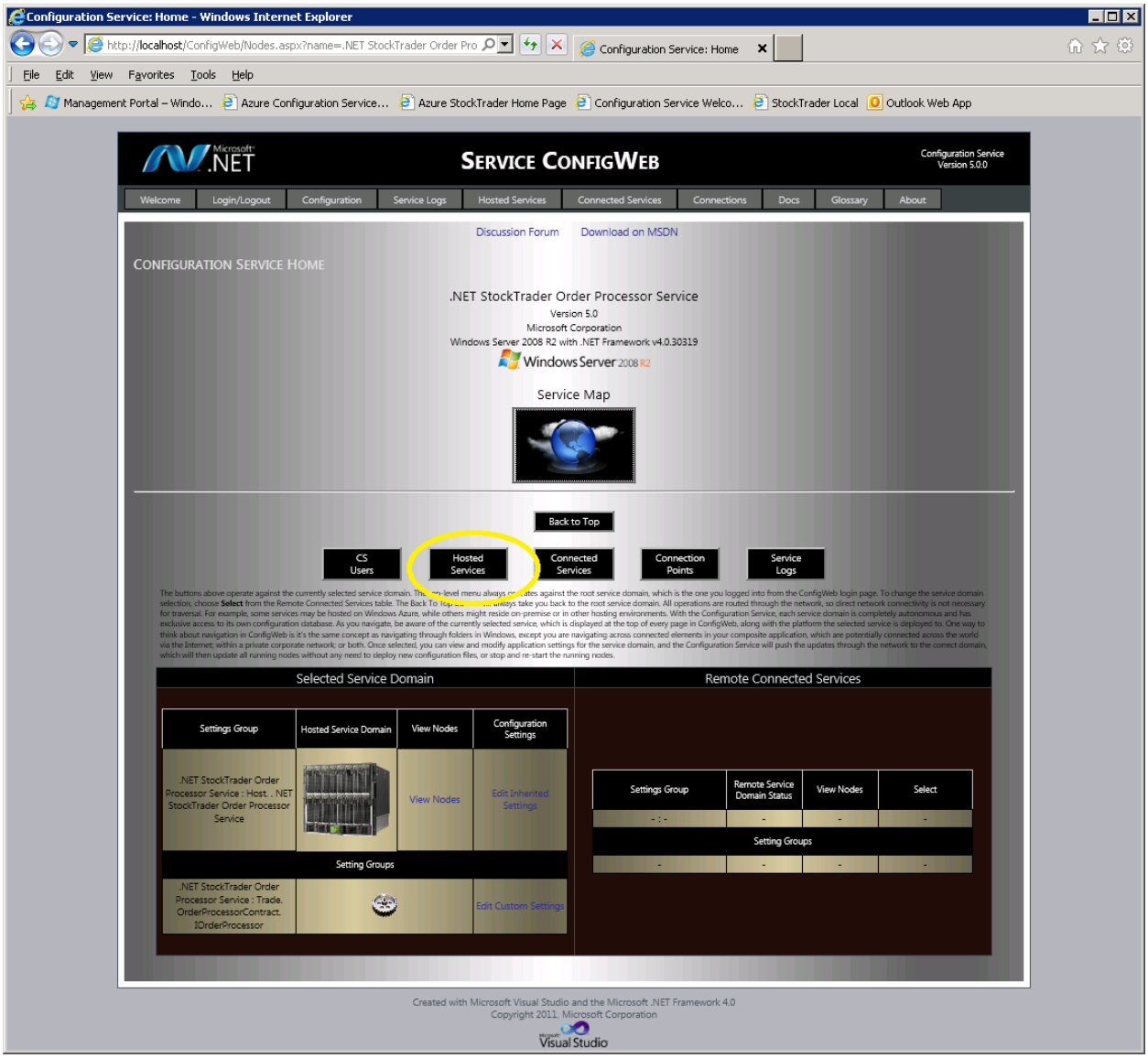
- Launch the Order Processor Windows Service Host from the StockTrader Start menu, if not already running.
- Login to the StockTrader Web application's Configuration Service via ConfigWeb.



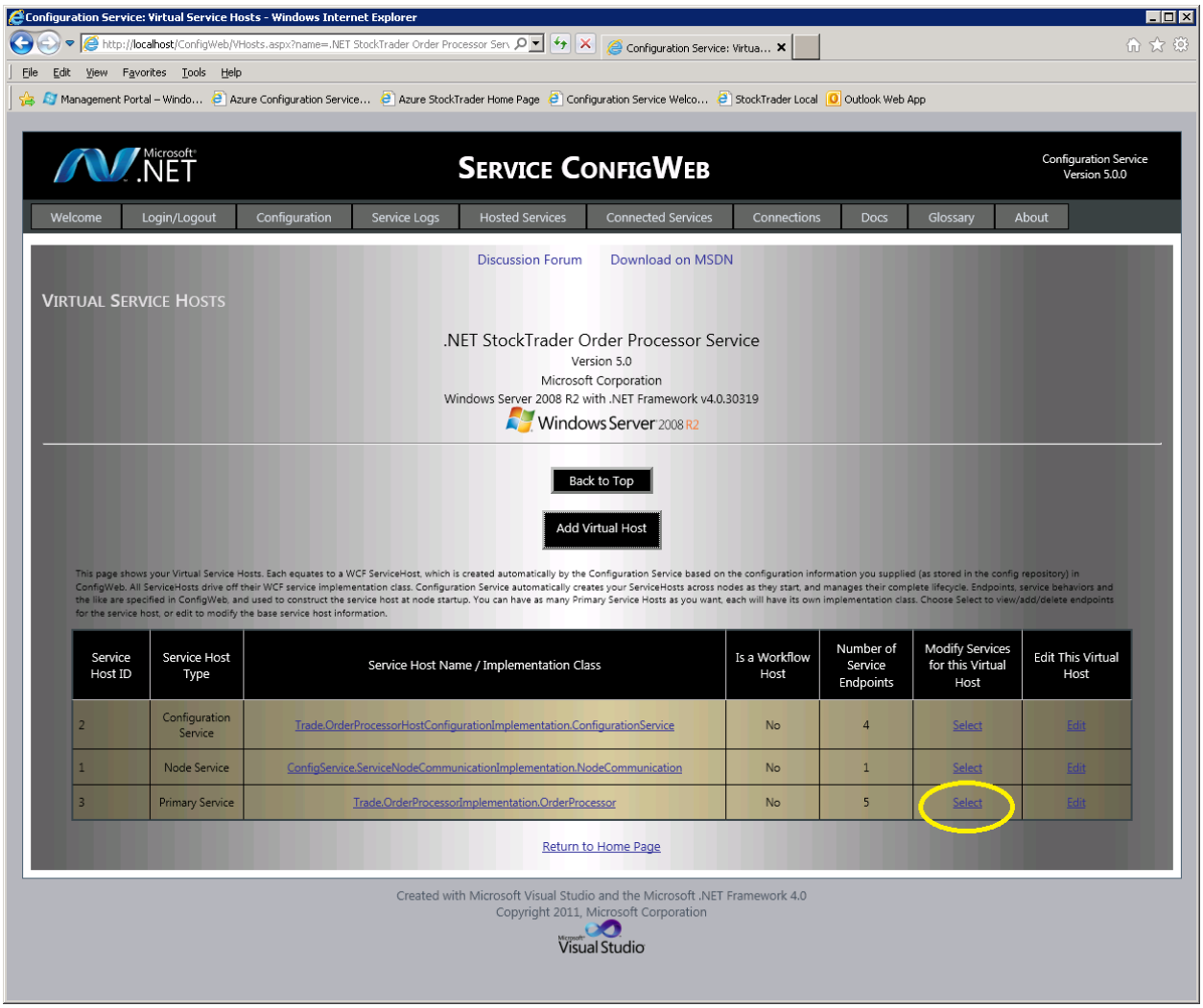
- Choose **Select** to navigate to the Business Service IIS Host configuration as the selected service domain. (note: you may be connected to the BSL self-host; in which case you should add connections for the Web app to the IIS Host service host, and set the StockTrader Web AccessMode setting to either the http or net.tcp BSL IIS Host endpoint setting.



- Choose **Select** again to select the Order Processor Service as the selected service domain.

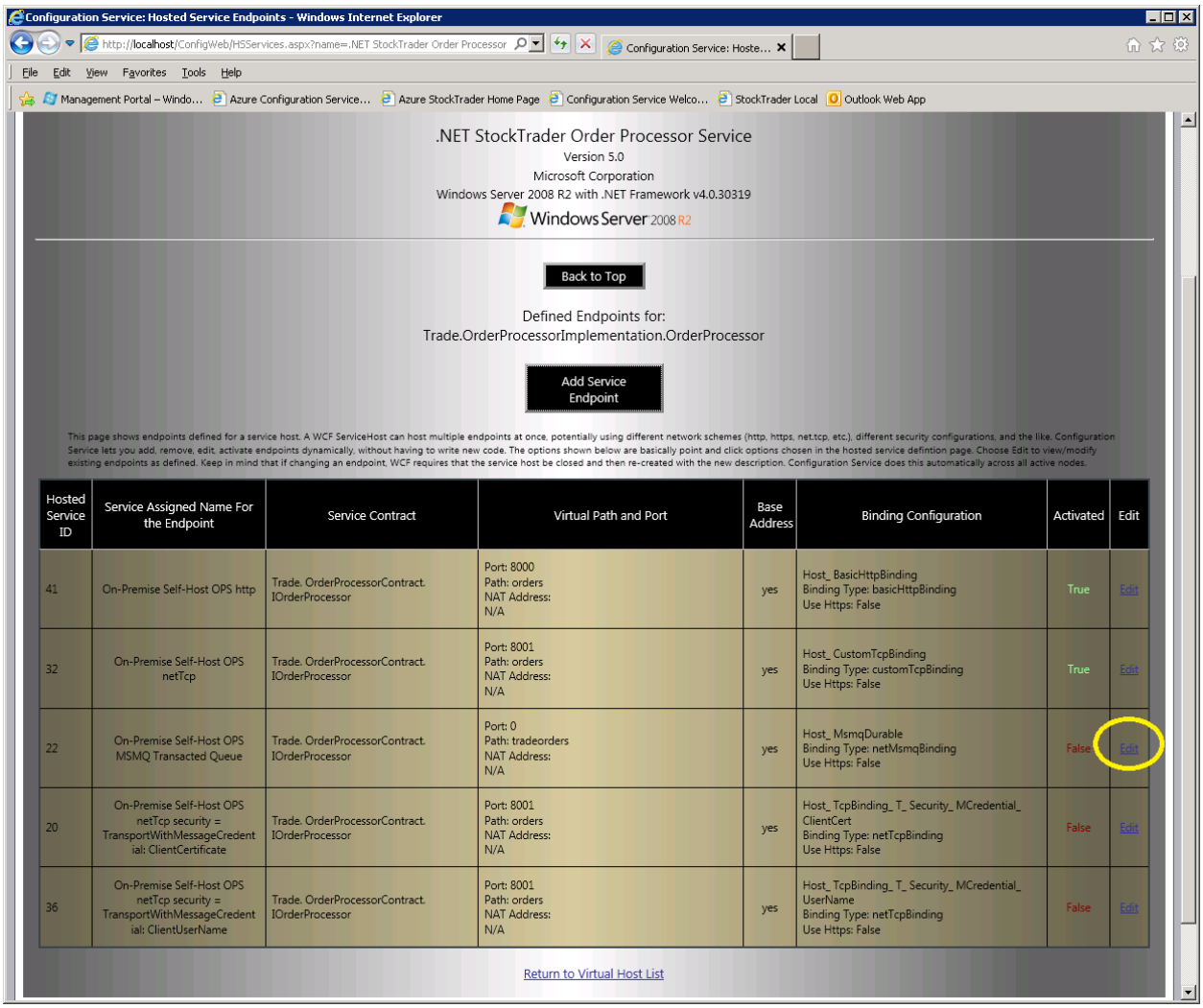


- Click on the Hosted Services button.



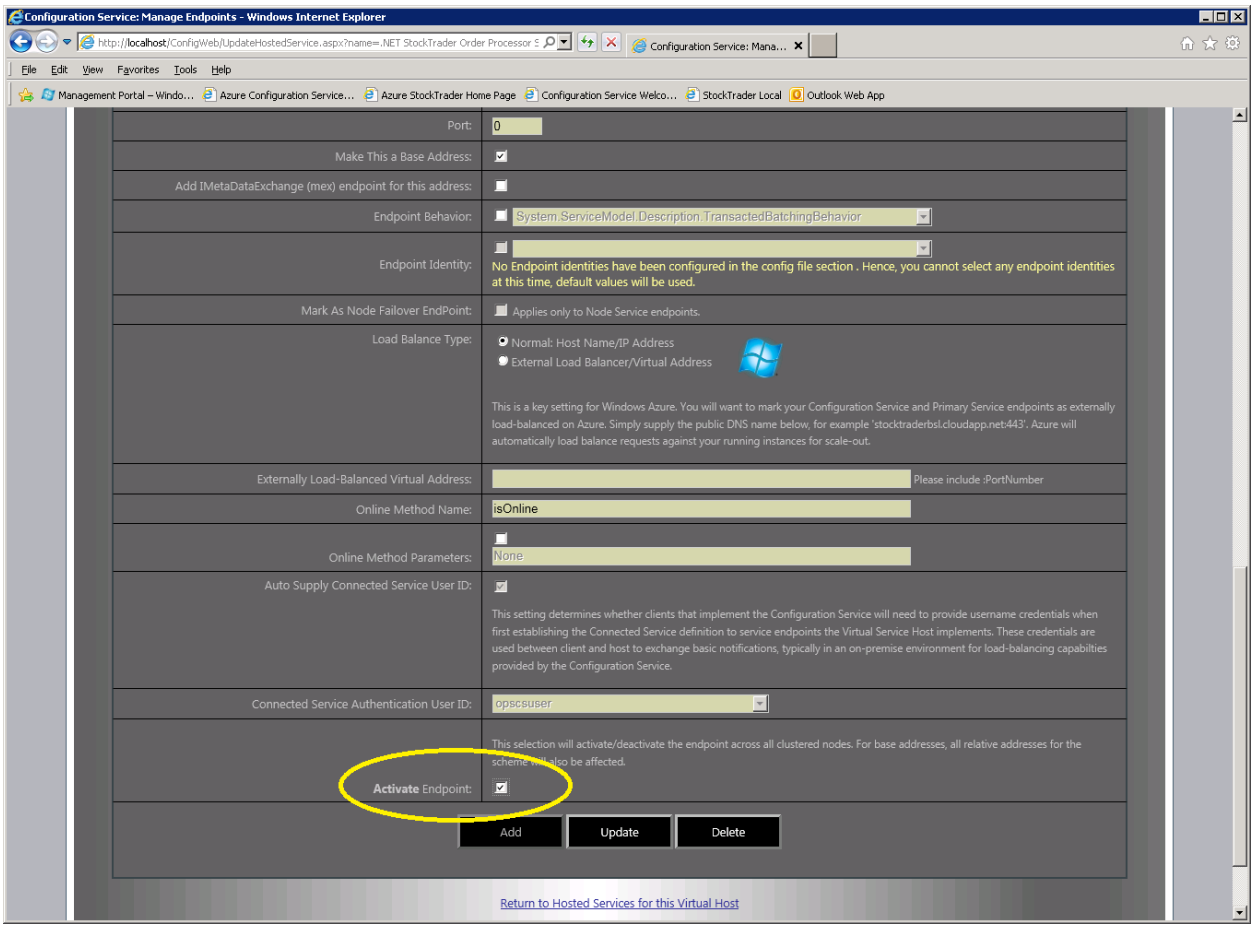
You will see a list of Virtual Service Hosts.

- Select the Trade.OrderProcessorImplementation.OrderProcessor Primary Service Host as shown.



- This is now a list of the **service endpoints** defined for this service host. Not all endpoints are active, and not all can be active at the same time.

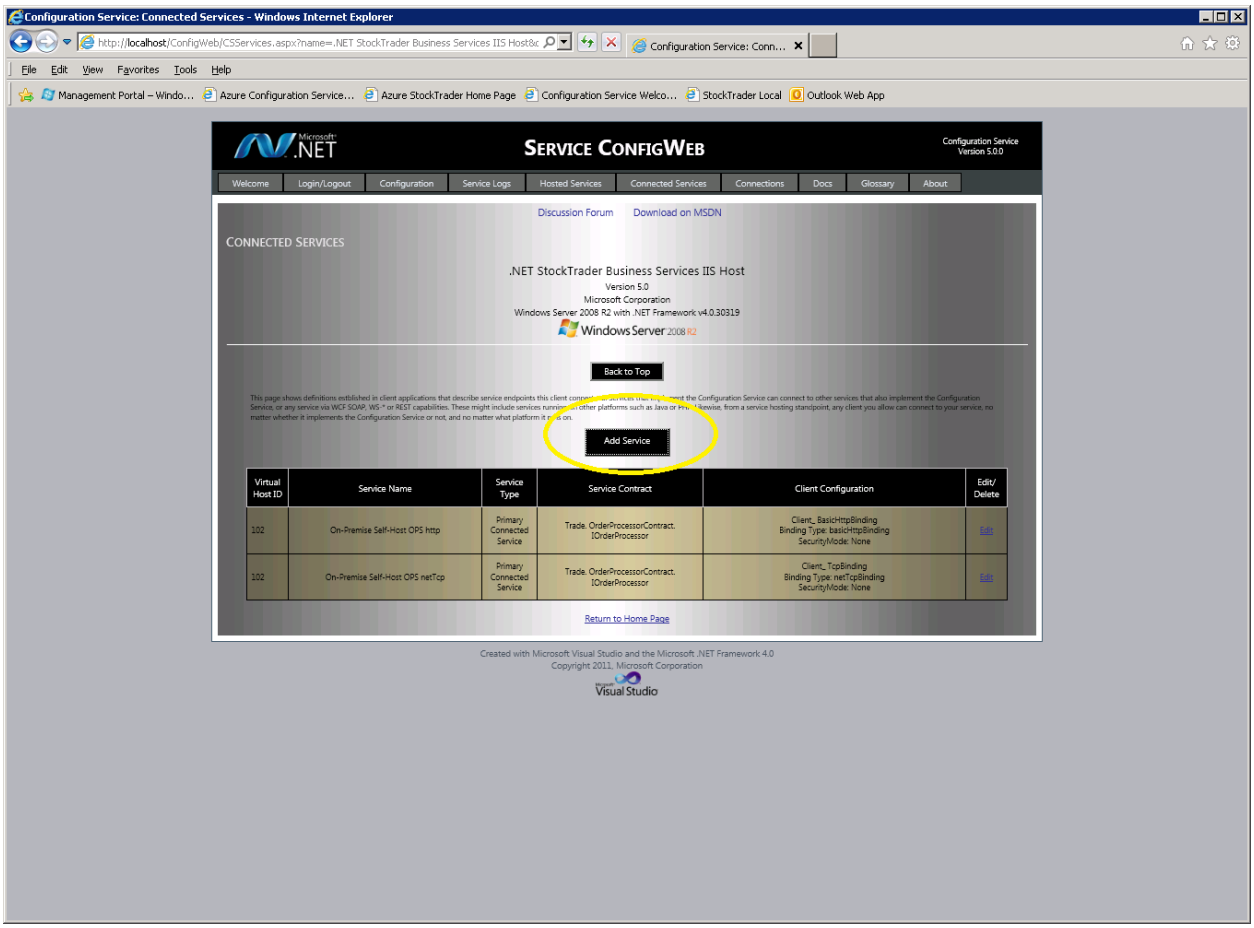
- Click the **Edit** link for the On-Premise Self-Host OPS MSMQ Transacted Queue endpoint.



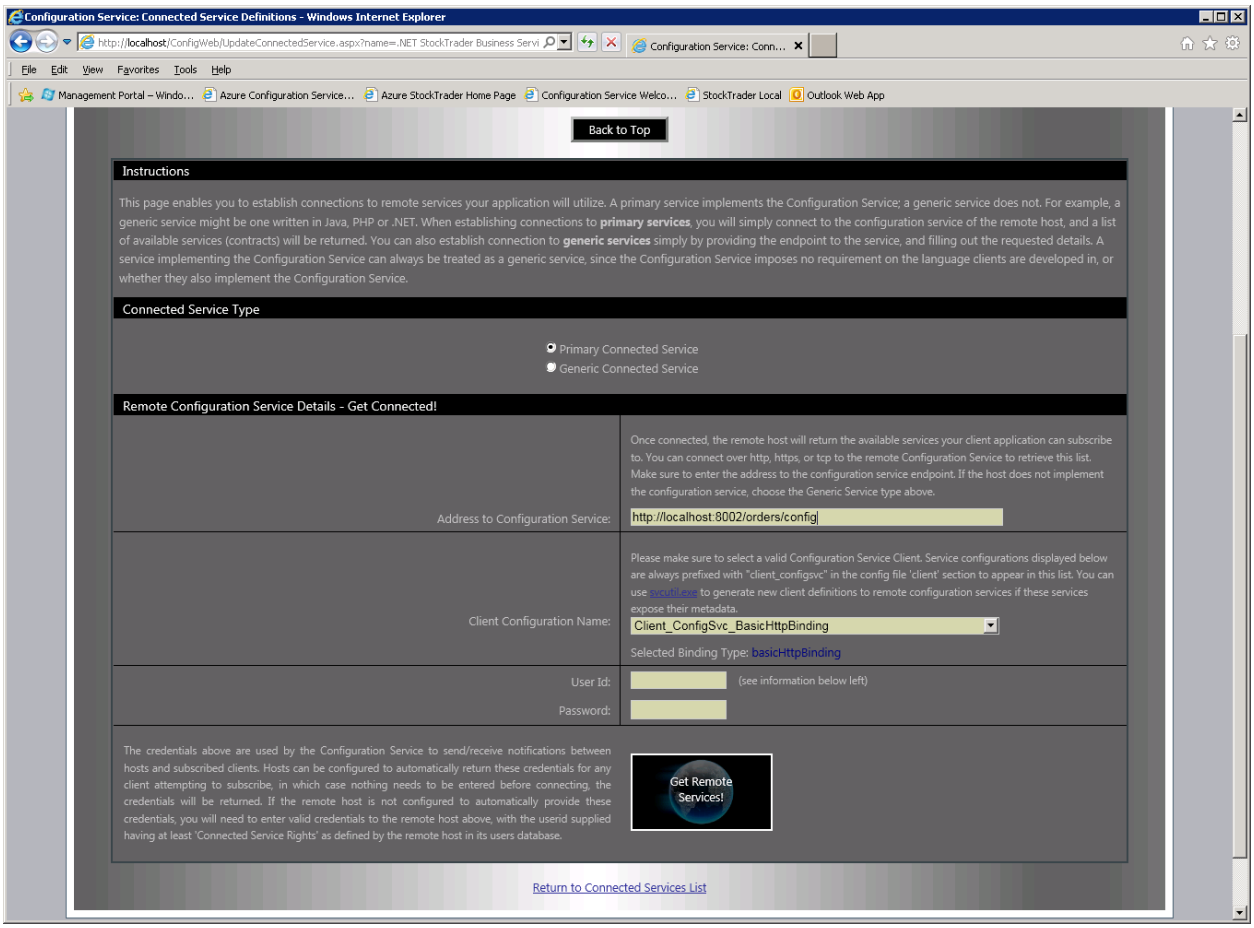
- Scroll down to the bottom of the page. Here you will select **Activate Endpoint**.
- Click the **Update** button.
- The service host will be restarted across all running nodes, this time with the MSMQ endpoint activated. You can look at the Main window of the Order Processor Windows host and see the service has been restarted with the new definition.
- Now we must create a new Connected Service definition for the IIS host, so it can use this new endpoint.
- Go back to the home page by clicking on the Back to Top button on the page.

- Select the IIS Host service domain as shown above.

- Now click on the **Connected Services** button.

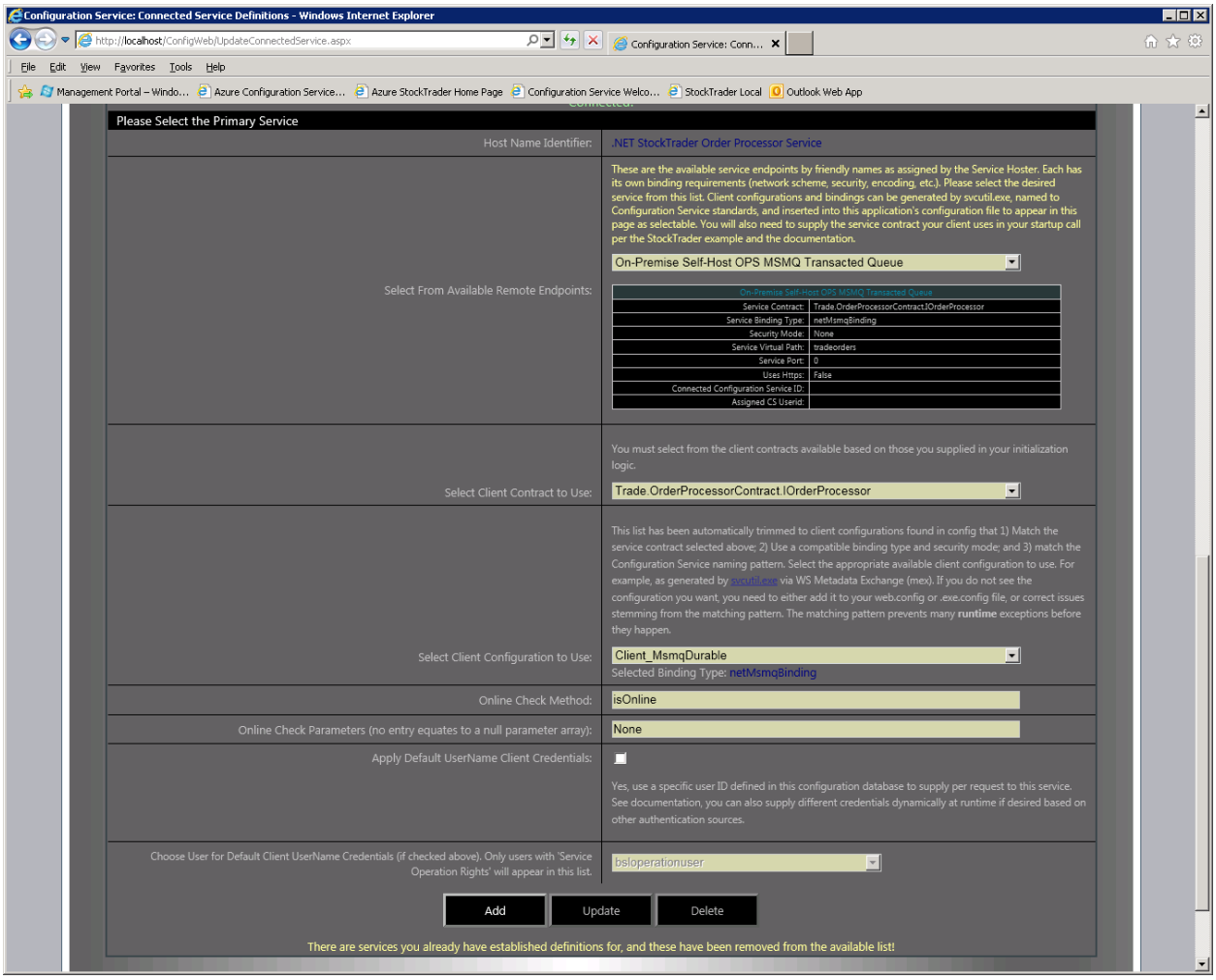


- Click on the **Add Service** button.



- Enter the address of the remote Order Processor configuration service. This is http://localhost:8002/orders/config.

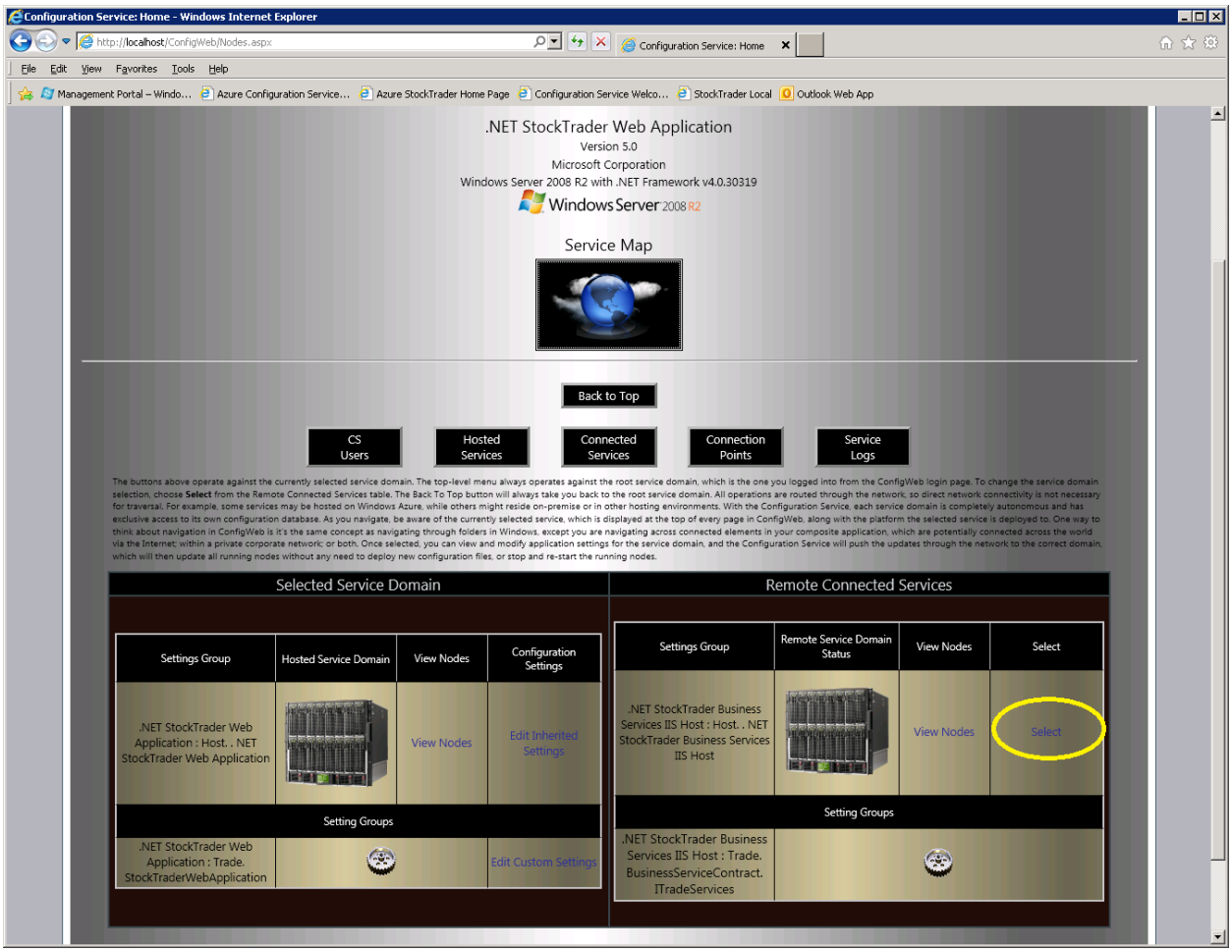
- Click **Get Remote Services!**



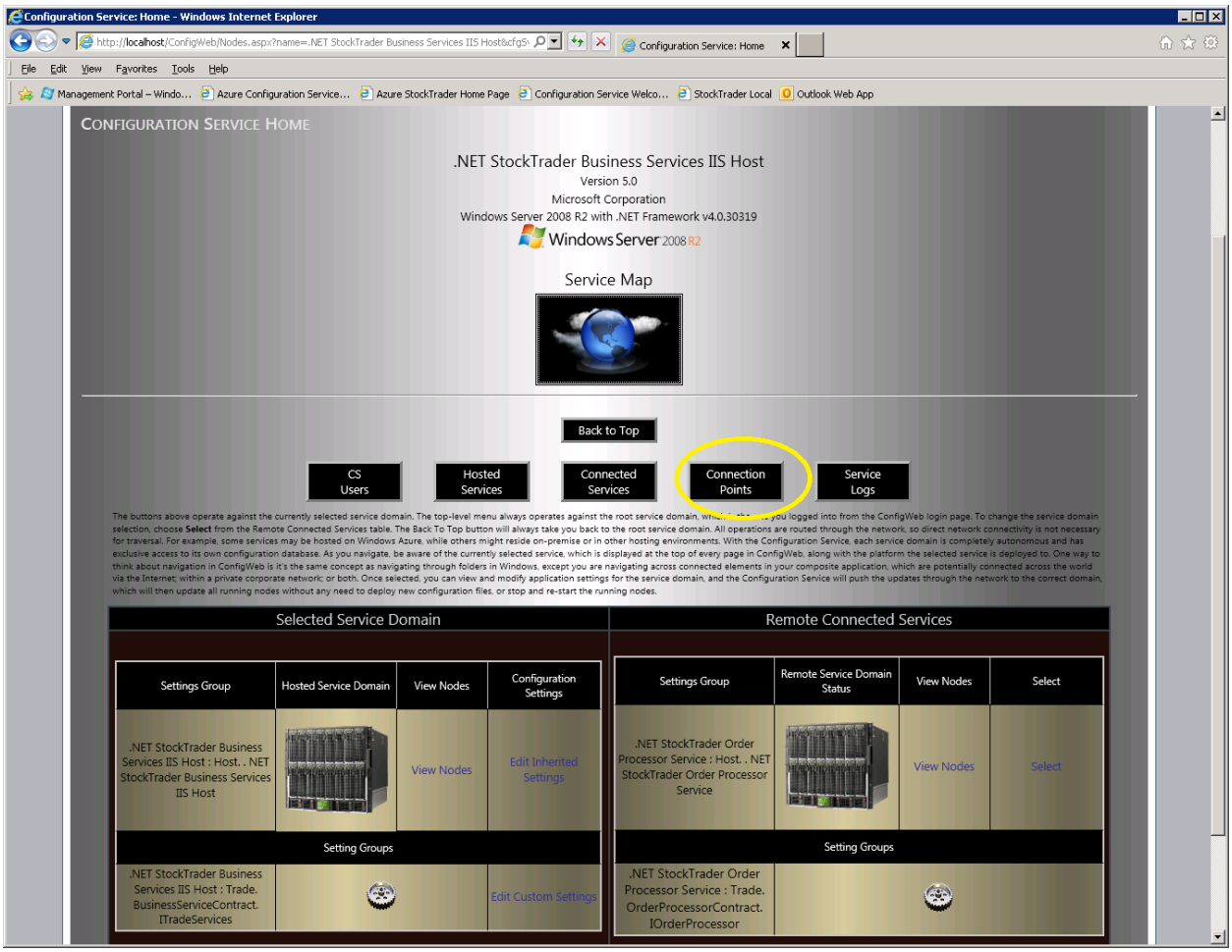
- Make sure the correct contract and client configuration are selected, they should be correct by default.

- Click the **Add** button.

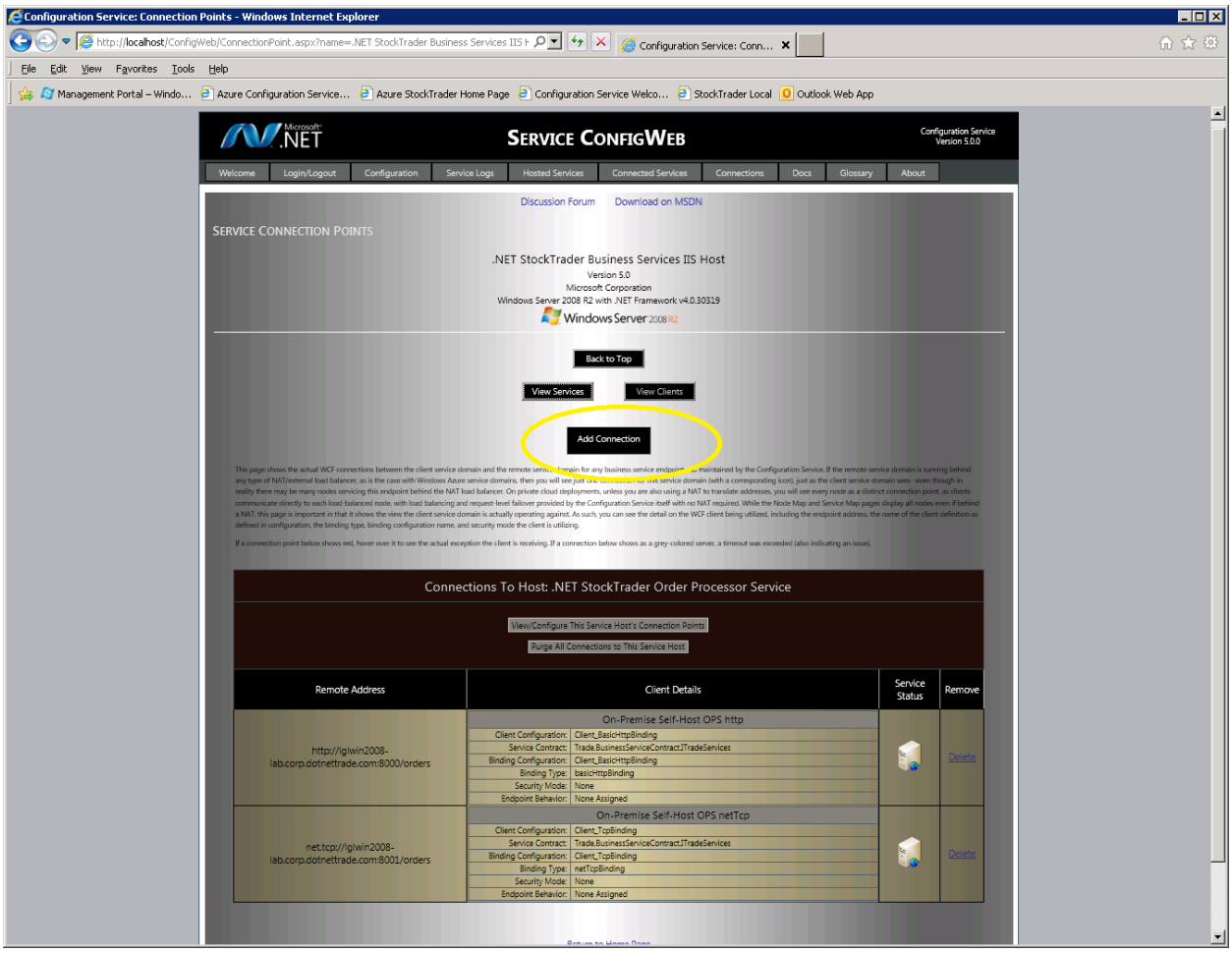
- Click on the Back to Top Page to go back to the root home page.



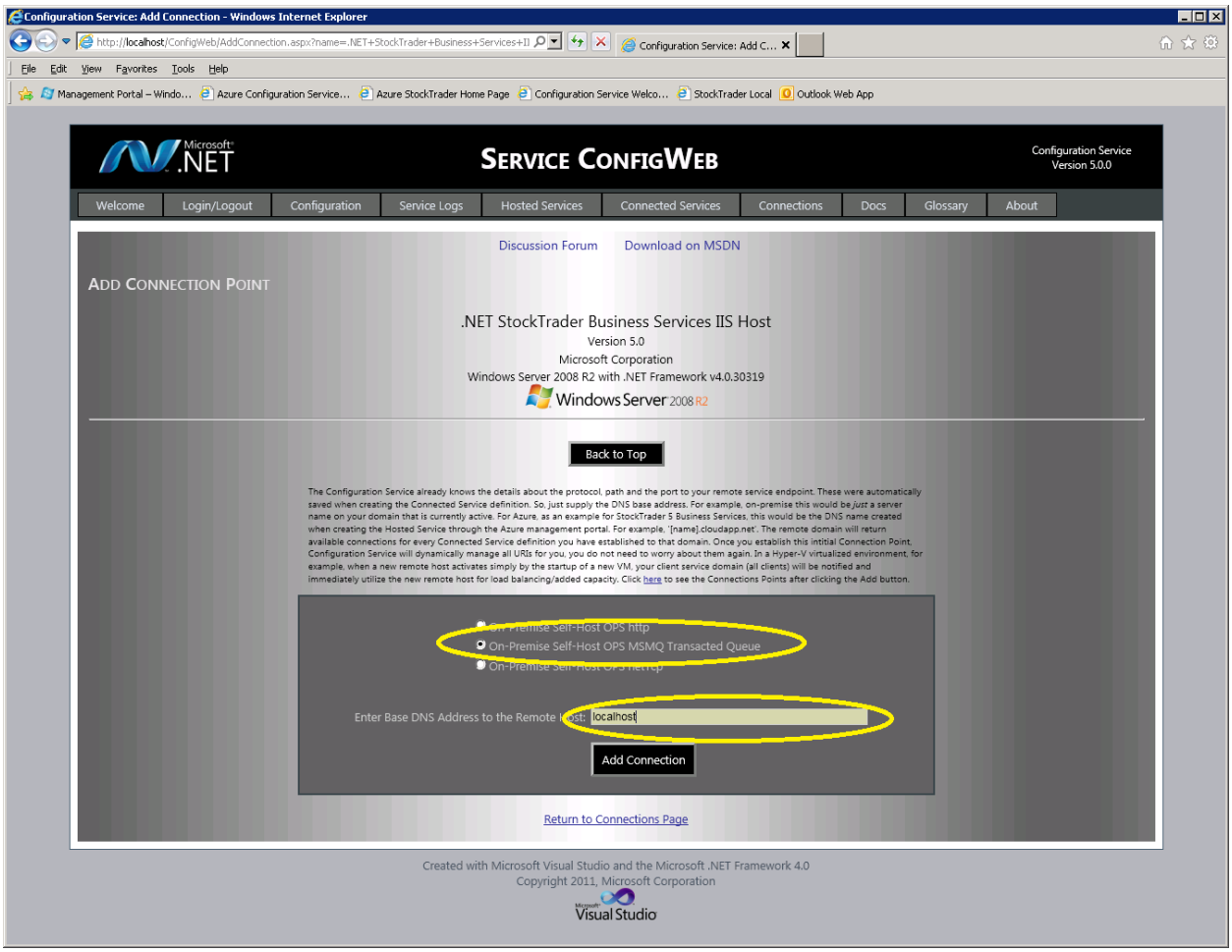
- Once again select the IIS Business Service host as shown, since we want to configure it.



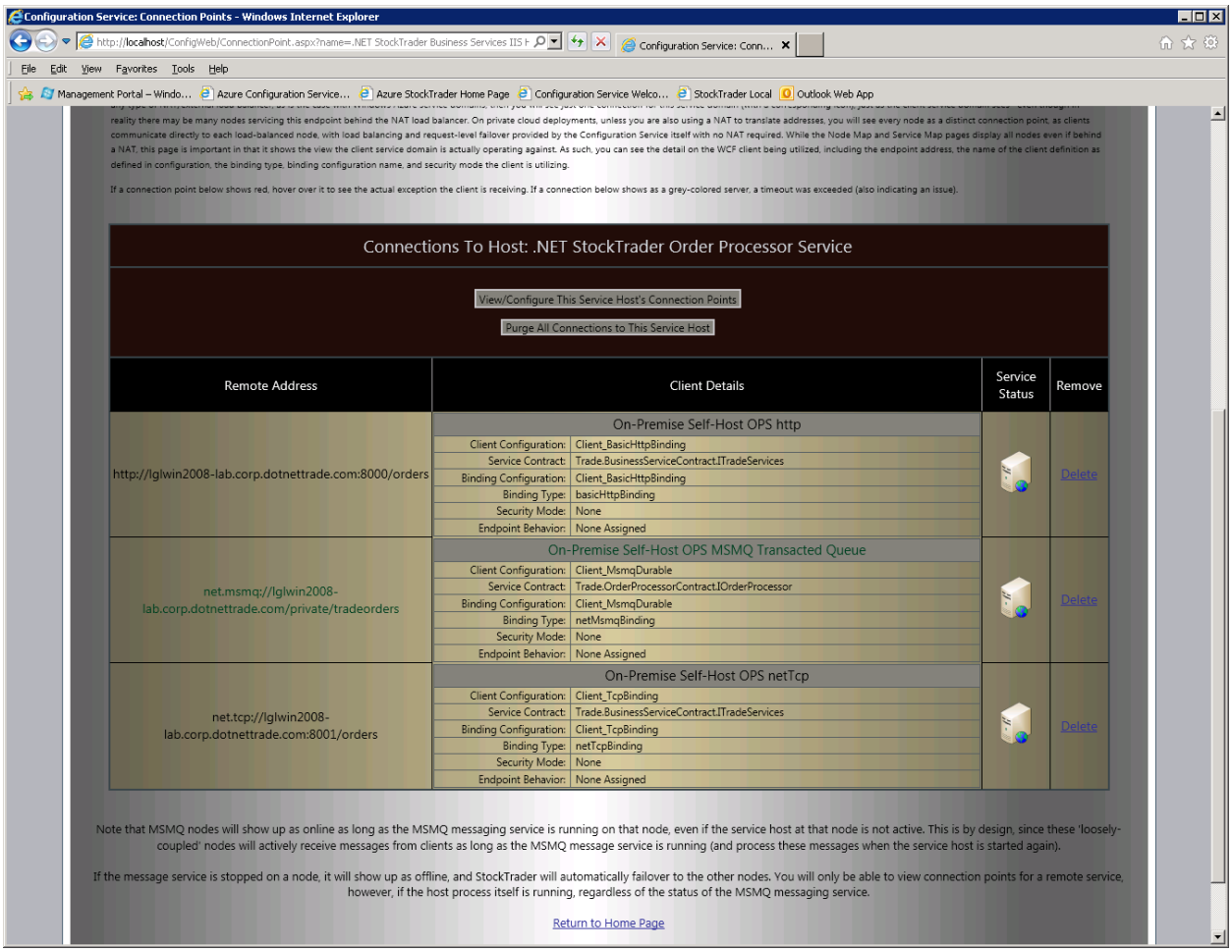
- Click on the **Connection Points** button (which will operate, like the other buttons, against the currently selected service domain).



- Click on the **Add Connection** Button.



- Select the MSMQ endpoint.
- Type in localhost as the host DNS name.
- Click Add Connection.
- Select the Return to Connections Page link at the bottom of the page when the connection has been added.



- You can see the MSMQ endpoint above. Note that in this screen shot, we already changed the **OrderMode** setting to On-Premise Self-Host OPS MSMQ Transacted Queue for the Business Service IIS host domain. Hence it shows as green, meaning currently being used (only one endpoint is used at a time for StockTrader—but not necessarily for all applications).

- You should make sure the BSL IIS Host is configured with **OrderMode** per above, or else the Trade application will not actually be using the MSMQ endpoint/durable message queue.

Note
If the MSMQ endpoint above shows up as red, you have an issue. Hover over the red icon to see the exception message (the complete exception will also be in the Service Logs page). It is likely you did not follow the post-install steps (see Readme.html in the start menu) to configure the MSMQ to allow permissions for the anonymous IIS account to write to the queue. If so, follow the readme post-install instructions related to MSMQ and then just refresh the page above.

New Hosts and Configuration Service Dynamic URI Management

You have used ConfigWeb to configure a variety of endpoints to a variety of hosts. You only have to do these steps once, even if new host nodes are started, however. When a new host now starts up, clients with definitions will all be notified, and Configuration Service will manage the URIs and load balancing. You can start the Order Processor program, for example, on a new server, and then in ConfigWeb the Service Map page and the Connections page will show it; and the BSL layer will already be using the new node for load balancing/failover. Note that both the Business Services and Order Processor Service can optionally be installed as **Windows NT services**, and set to Auto-start when a machine starts up or is restarted.

Re-Configuring StockTrader to Work with Oracle 11G

While the configuration repositories must be SQL Server/SQL Azure in this release, a complete Oracle DAL is provided for the core application and it can run against Oracle 11G databases. You will want to install the latest Oracle 11G 32-bit or 64-bit client, downloadable from Oracle Corporation's Web site. You will also want to make sure you install Oracle's latest ADO.NET ODP, which installs their Oracle.DataAcess assembly. The client and the ODP need to be installed on each computer running any component/service for StockTrader.

You will use TNS Names to create a name for the database, and simply update the configuration via ConfigWeb to use the Oracle DAL instead of the SQL Server DAL. To first create the Oracle database, you will find the DDL script located in:

[stocktrader setup dir]\stocktrader\DatabaseLoaders\TradeOracleLoader\Create_Oracle_Trade_Database_DDLScript

Simply use **SQLPlus** to create the database schema by running this file. To create the Oracle schema, please see the .NET StockTrader Oracle Database Loader program install directory for the DDL which can be run from SQLPlus. It is ideal to first create a new tablespace, then create a new user 'trade'; password 'trade' assigned to this tablespace, and then login to **SQL Plus** as 'trade' to create the Trade database schema from the provided DDL.

Next, create a TNS Names entry for your SQL Server instance, with the login trade, trade. This will be your reference to the database from the various StockTrader service domains, and this entry and the Oracle client hence must be installed on every node running .NET StockTrader.

The .NET Oracle Database loader program can then be used to load the Oracle database with data. Simply run this from the Start menu, and for the server name use your TNS name entry.

- Once loaded with data, login to ConfigWeb against the StockTrader Web application service domain, while the application is still configured to run against SQL Server.

- Next, select the Business Service domain, to configure its custom settings, and select **Edit Custom Settings** for the Business Services.

- Select the Detailed Settings (button at top of page).

- Click on Change Value for the setting named **DataAccessLayer**, and change the value from Trade.DALSQLServer to Trade.DALOracle.

- Click Update to apply the change.

- Next change the setting for **Stocks Database Name** to be your TNS name entry for your Oracle server. It is also under "Detailed" settings for the Business Services.

- Finally, change the **Stocks DatabaseUserID** and **Stocks Database Password** to be the values you set when creating the trade userid (which should have its own tablespace where the database lives) and the trade password.

- These application settings are also part of the Order Processor tier. If the OPS is running when making these changes to the BSL above, they are pushed down to the OPS. But you should check these corresponding settings for the Order Processor Service via ConfigWeb to make sure they are set the same as above.

- You can now try to login to the Trade Web application as an end-user, to check everything is working against Oracle!

Separation of Implementation from Schema and Contract

In the service-oriented world, a properly designed system cleanly separates implementation of services into autonomous black boxes, to avoid ever having to share classes/code between layers. When running with AccessMode set to any setting other than 'InProcess'; this is the way StockTrader is designed. However, StockTrader supports the 'InProcess' collapsed mode to run as a standalone Web application as well, which is also a perfectly valid configuration. Hence, the Visual Studio StockTrader Web Application solution does share projects and classes from Business Services to support this mode. You should keep in mind, as noted in the source code comments, that this is a special case; and that normally the only sharing between the StockTrader Web application projects and the Business Service projects would be the **BusinessServiceContract** project, and the **BusinessServiceDataContract** project. The same is true for the Business Service solution, which references Order Processor implementation projects to support the in-process order mode (OrderMode='InProcess Activation').

The information for the service and data contract can also optionally be generated via the **SVCUTIL.exe** tool that is part of the Windows 7.x SDK; for example if someone else is hosting the service outside of your organization; or it is written in a different platform like J2EE. This tool generates both the Data Contracts and the Service Contracts for building clients; it also generates the binding information to be used. This can then be used as-is; or easily modified into a custom client as we did with .NET StockTrader. We do not use a purely generated proxy because:

a) We already have a DataContract and ServiceContract designed in separate re-usable projects that can simply be imported into other solutions.
b) We use the WCF **ChannelFactory** class with cached instances of channels to Business Services (and the Order Processing Service) for performance reasons.

Why A Configuration Service and ConfigWeb?

As a composite application, .NET StockTrader can have many services running across multiple servers. In addition, as a performance sample application, .NET StockTrader has a wealth of different configuration settings to allow developers to test the performance of various different technologies, protocols and deployment topologies for an application. Without a way to centrally manage configuration data, it would be extremely difficult to ensure the composite application, as whole, was configured as desired and to keep configuration files in sync across various servers—especially if services are deployed in geographically disperse locations. The solution implemented in StockTrader is the configuration management service, which is a WCF Web service hosted in each component (StockTrader Web Application, StockTrader Business Services, and StockTrader Order Processor Service). The configuration system relies on a SQL-Server (or SQL Azure) based repository schema.

This service utilizes a central SQL repository/schema, and each element of StockTrader has its own separate repository. All of the data is not stored in a single shared repository because, to a large degree, this would degrade many of the intended benefits of service-orientation. For example, if

all elements of the application shared one central database repository, then all would need to be deployed within one application domain, and typically on the same subnet; or a subnet with direct connectivity to the single central database.

The configuration service takes a different approach such that each service can be deployed anywhere. The only requirement of course is that services that directly connect to each other must have network connectivity. But a composite application might be made up of services that connect several layers deep, in different geographic regions, and all elements might not require (or even be able to achieve) direct connections to all other elements. A composite application should not be required to be deployed on the same internal network just to surface an integrated management and configuration experience. Hence, the configuration management service maintains the principle that ***services should be autonomous—***and each gets its own configuration repository, although the schemas for the repository are common.

Starting Point for Implementing the Configuration Service

If you are interested in implementing the configuration service, you should start with the new Visual Studio 2010 template and tutorial. With 5.0, the process is very straightforward since everything is provided in base classes. Once done, you can use all the features of the Configuration Service and ConfigWeb for your own applications and services.