WPF Dev Camp - Module 6, Demo 2

Using Service Bus Relay to Connect to On-Premises Service

Lab version: 1.0

Last updated: 1/13/2015

Estimated demo delivery time: **10 minutes**.



Contents

[Overview 3](#_Toc408469608)

[Prerequisites 3](#_Toc408469609)

[Setup 3](#_Toc408469610)

[Exercise 1: Creating an Internal Service 4](#_Toc408469611)

[Task 1: Overview of Internal Service 4](#_Toc408469612)

[Task 2: Add Reference to Internal Service 5](#_Toc408469613)

[Task 3: Modify WCF Service to Call Internal Service 7](#_Toc408469614)

[Task 4: Demonstrating the Internal Service Running Locally 8](#_Toc408469615)

[Exercise 2: Connecting Cloud Service to On-Premises Service using Azure Service Bus Relay 11](#_Toc408469616)

[Task 1: Creating a Service Bus Namespace 11](#_Toc408469617)

[Task 2: Configure Internal Service to Listen to Service Bus Relay 15](#_Toc408469618)

[Task 3: Updating Expenses WCF Service to Call Internal Service via Service Bus Relay 17](#_Toc408469619)

[Task 4: Publishing Updated Expenses WCF Service to Azure Website 19](#_Toc408469620)

[Task 5: Testing the Service Bus Relay Connectivity from Cloud to On-Premises 19](#_Toc408469621)

# Overview

In this demo, we will show how to use an Azure Service Bus Relay to connect the Expenses WCF service running in an Azure Website to a WCF service that must remain on-premises.

# Prerequisites

The following are required to complete this demo:

* [Microsoft Visual Studio 2013](http://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx) (tested with Update 4)
* Internet connection
* [Microsoft Azure](http://azure.microsoft.com/en-us/pricing/free-trial/) subscription
* Expenses codebase

# Setup

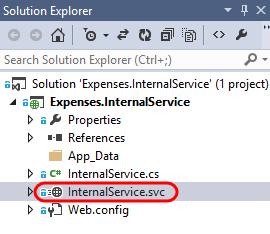
* Load and build the Expenses solution to ensure that it builds correctly.
* Log into the Microsoft Azure subscription that you will be using for demonstration.

# Exercise 1: Creating an Internal Service

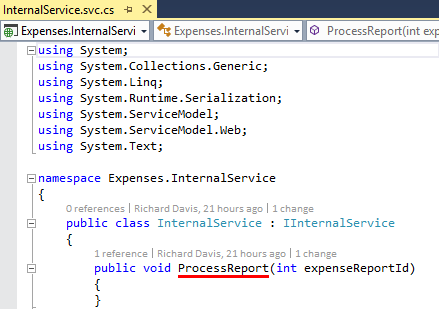
In this exercise, we will extend the Expenses scenario a bit by adding in the need for some services to remain on-premises. This exercise shows off the simple internal service definition and walks through the steps to include calls to it from the existing WCF service.

## Task 1: Overview of Internal Service

1. Load the Expenses.InternalService.sln solution in Visual Studio.
2. Double-click on InternalService.svc in Solution Explorer.

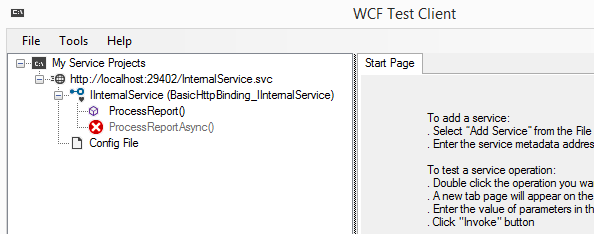


1. This service exposes an API to process submitted expense reports. It has been separated from the other WCF service because there is additional workflow that needs to take place on-premises for the time being.

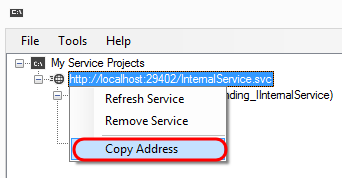


## Task 2: Add Reference to Internal Service

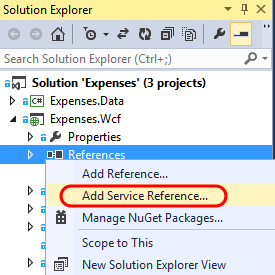
1. Select InternalService.svc in Solution Explorer and then press F5 to launch the internal service hosted locally in IIS Express. This should also launch the WCF Test Client.



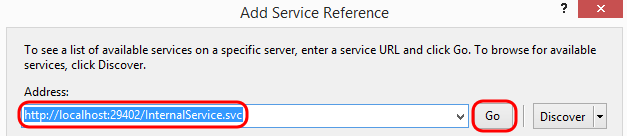
1. Right-click on the service project node for the internal service and select Copy Address.



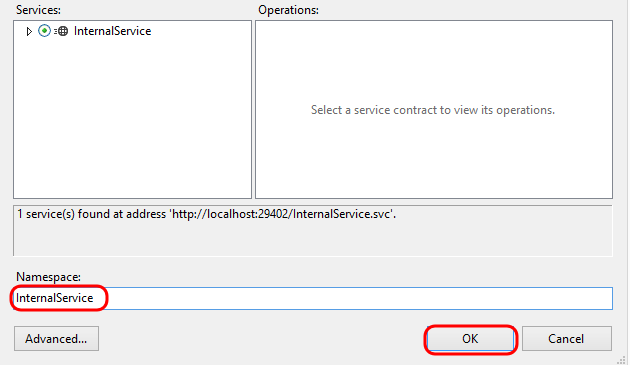
1. Load the Expenses.sln solution in a new instance of Visual Studio.
2. In the Expenses.Wcf project, right-click on the References node and select Add Service Reference.



1. Paste the URL to the internal service hosted locally into the Address field and then click Go.

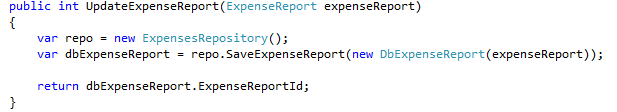


1. Note that the InternalService was found at the provided address. Change the Namespace to be InternalService and then click OK.



## Task 3: Modify WCF Service to Call Internal Service

1. Double-click ExpenseService.svc to open the service code in the editor.
2. Scroll down and locate the UpdateExpenseReport method. This method takes an expense report object and saves it to the database.



1. Insert the following code snippet just after the call to SaveExpenseReport, and just before the return. This will make a call to the internal service once the expense report has been approved for further processing.

if (dbExpenseReport.Status == DbExpenseReportStatus.Approved)

{

using (var client = new InternalService.InternalServiceClient())

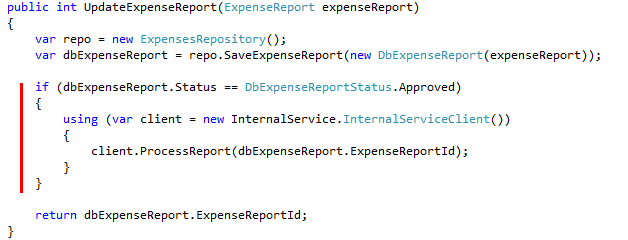
{

client.ProcessReport(dbExpenseReport.ExpenseReportId);

}

}

1. The final UpdateExpenseReport method should look like the following:

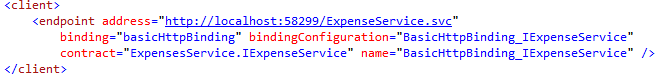


## Task 4: Demonstrating the Internal Service Running Locally

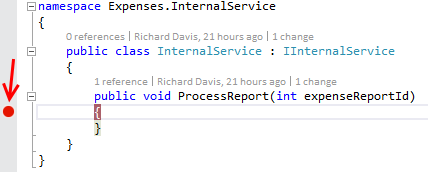
1. Ensure that connection string for the Expenses.Wcf project is set to be LocalDb.



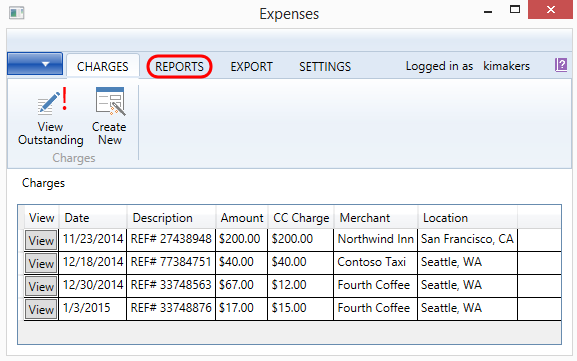
1. Ensure that the service endpoint configured for the WPF client application is set to match the localhost version of the Expenses.Wcf service (you may have the production website endpoint here if you are following along from previous demos, so just comment that one out and return to the default).



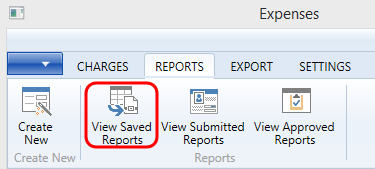
1. In the instance of Visual Studio that has the Expenses.InternalService project loaded and running, set a breakpoint within the ProcessReport method.



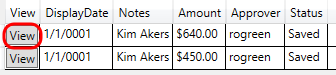
1. In the instance of Visual Studio containing the full WCF service and WPF client application, press F5 to launch both applications.
2. In the Expenses WFP application, click the Reports tab.



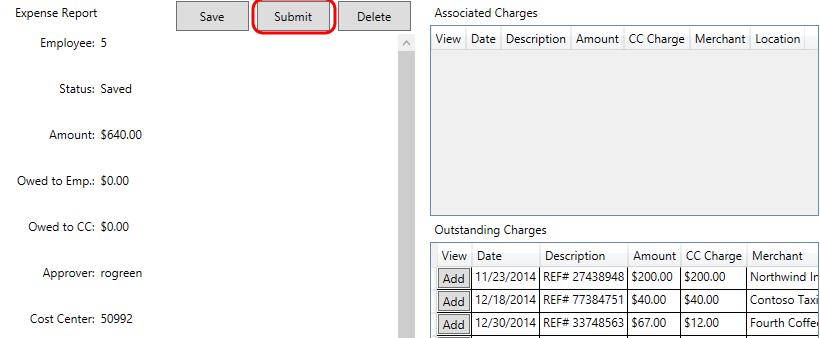
1. Click the View Saved Reports button.



1. Click on the View button next to one of the reports in the Saved state.



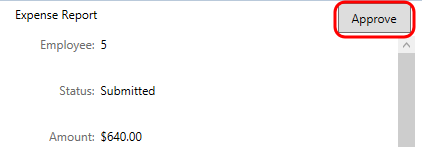
1. Click Submit.



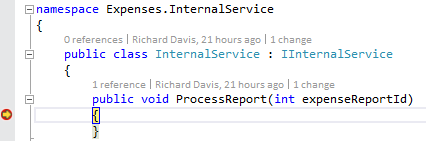
1. You are automatically taken to the Submitted Reports tab. Click the View button next to the report once again.



1. Click the Approve button.



1. The breakpoint that you set in the ProcessReport method within the internal service should be hit as expected.



1. Stop debugging in both instances of Visual Studio.

# Exercise 2: Connecting Cloud Service to On-Premises Service using Azure Service Bus Relay

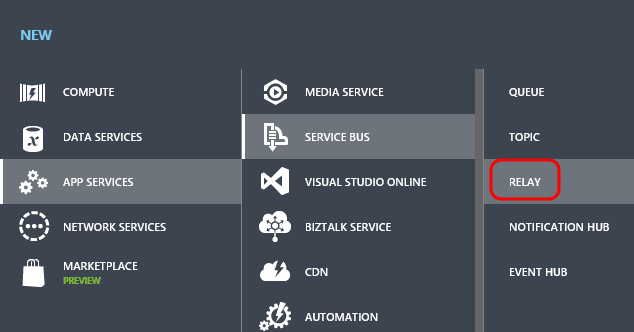
In this exercise, we will take the steps necessary to create a Service Bus Namespace, configure the new internal service running on-premises to listen using a relay, and configure the existing Expenses service to make calls using the relay.

## Task 1: Creating a Service Bus Namespace

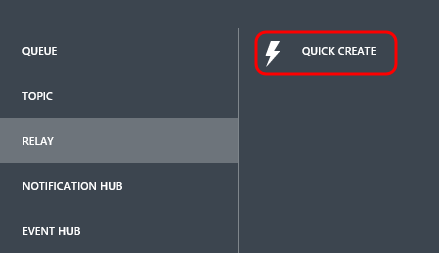
1. Log into the full Microsoft Azure portal [here](https://manage.windowsazure.com) (or if you are currently in the Preview portal at portal.azure.com you can click on your user name in the top-right corner and select the Full Azure Portal link).
2. Click the New button in the bottom-left corner.



1. Select App Services | Service Bus | Relay.



1. Click the Quick Create option.



1. Provide a globally unique name for your Service Bus namespace. Make sure that your selection results in a green checkmark.



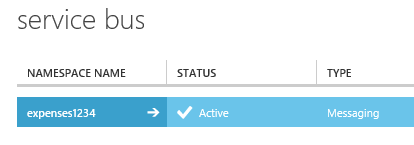
1. Select the same Region that you have been using for other Azure resources in previous demos. This isn’t required, but is generally a good idea for performance reasons.



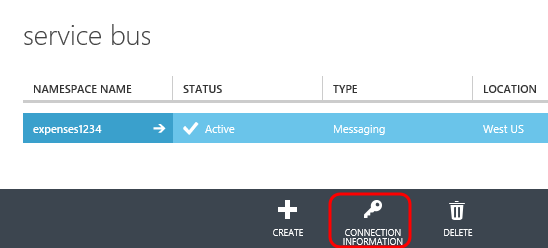
1. Select Create a Relay.



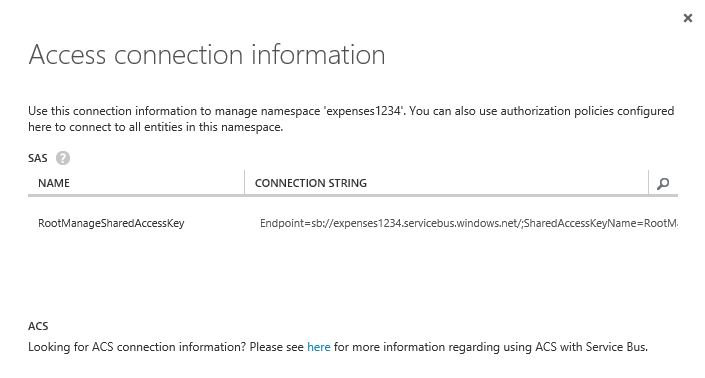
1. After a few moments the new Service Bus will be created and in the Active state.



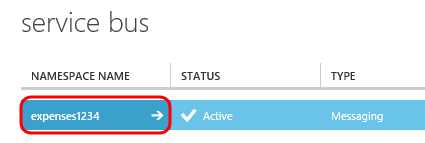
1. With the new namespace selected, click the Connection Information button near the bottom.



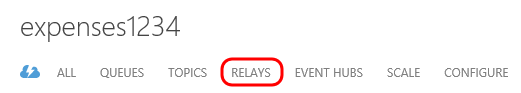
1. Namespaces created via the portal will be setup to use Shared Access Signature (SAS) authentication. This provides an access key that is then used to generate a SAS token that clients can use to authenticate. This is recommended for applications scenarios where there is no need to manage the notion of an authorized “user”. In the event that the application scenario requires a richer, identity-based authentication option to perhaps federate with other standards-based identity providers (ADFS, Microsoft accounts, Google, Facebook), you can also create namespaces that use the Access Control Service (ACS).



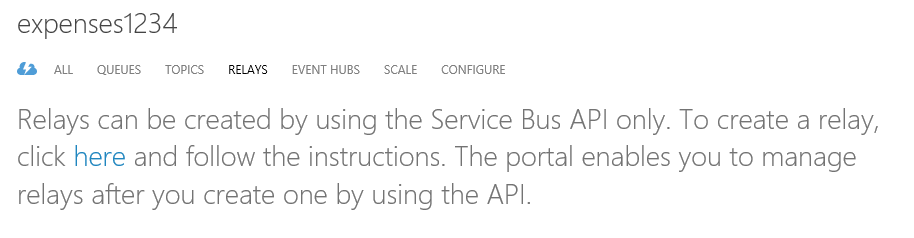
1. Press Escape to close the Access Connection Information window.
2. Drill into the new Service Bus namespace.

.

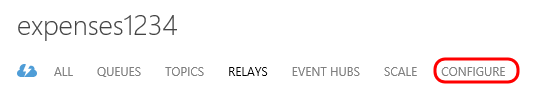
1. This namespace can contain a number of different objects in it including Queues, Topics, Relays, and so on. Click on the Relays tab.



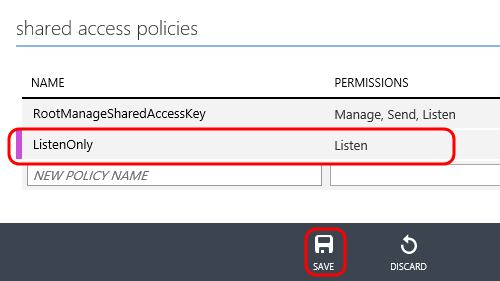
1. Relays are created using the Service Bus API, as stated by the information here.



1. Click on the Configure tab.



1. This tab allows you to configure Shared Access Policies and the keys associated with them. By default, there is a policy named RootManageSharedAccessKey with all permissions selected. The internal WCF service that we have created just needs the Listen permission, so create a new Shared Access Policy named “ListenOnly” and assign the appropriate permission. Click Save.



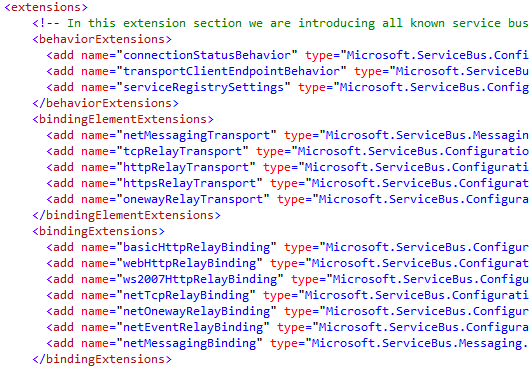
1. We will return to this configuration information shortly, so leave the window open.

## Task 2: Configure Internal Service to Listen to Service Bus Relay

1. Return to the instance of Visual Studio with Expenses.InternalService open.
2. Select Tools | NuGet Package Manager | Package Manager Console from the main menu.
3. Type the following command to install Microsoft Azure Service Bus package:

Install-Package WindowsAzure.ServiceBus -Version 2.5.3.0

1. Installing this NuGet package adds a reference to Microsoft.ServiceBus and other dependencies, adds all service bus extensions to the system.serviceModel section, and even adds in a placeholder Service Bus connection string (although we are not going to make use of that in this demo).



1. Let’s add in an additional service endpoint that will be responsible for listening for messages in our Service Bus Relay. Copy the following snippet and paste it just after the existing service endpoint definition:

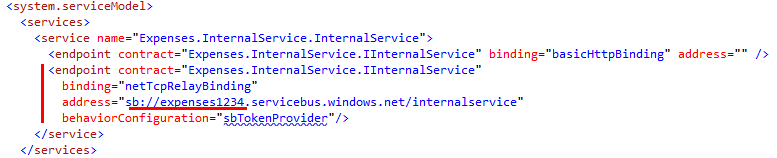
<endpoint contract="Expenses.InternalService.IInternalService"

binding="netTcpRelayBinding"

address="sb://{YOUR NAMESPACE}.servicebus.windows.net/internalservice"

behaviorConfiguration="sbTokenProvider"/>

1. Replace the {YOUR NAMESPACE} placeholder with the one that you just created.



1. The last step is to define the “sbTokenProvider” behavior configuration. Copy the following snippet and paste into the behaviors section:

<endpointBehaviors>

<behavior name="sbTokenProvider">

<transportClientEndpointBehavior>

<tokenProvider>

<sharedAccessSignature keyName="ListenOnly" key="{YOUR KEY}" />

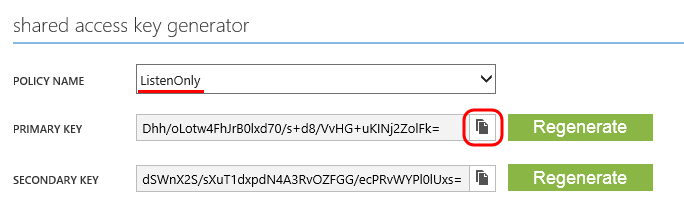
</tokenProvider>

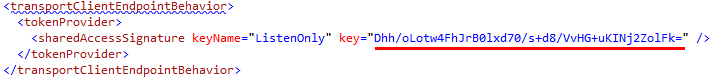
</transportClientEndpointBehavior>

</behavior>

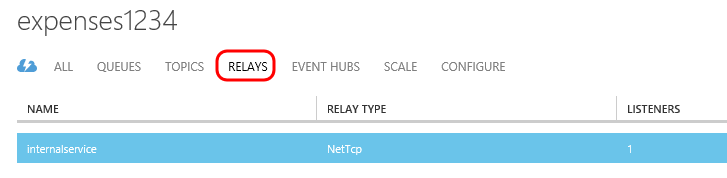
</endpointBehaviors>

1. Replace the {YOUR KEY} placeholder with the one that was generated when you created the ListenOnly Shared Access Policy earlier.



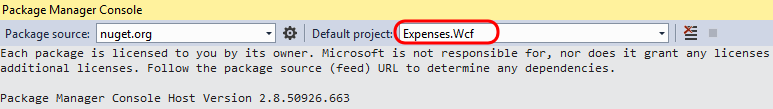


1. Select InternalService.svc in Solution Explorer and then press F5 to run the internal service locally. This will be hosted by IIS Express in this demo, but in a real scenario you would likely deploy the service to a server running the IIS role.
2. Return to the Azure portal and navigate to the Relays tab. Note that there is now a listener for the “internalservice” relay.

****

## Task 3: Updating Expenses WCF Service to Call Internal Service via Service Bus Relay

1. Return to the instance of Visual Studio that has the main Expenses solution open.
2. Select Tools | NuGet Package Manager | Package Manager Console from the main menu.
3. In the Package Manager Console window, ensure that the Default Project selected is Expenses.Wcf.



1. Type the following command to install Microsoft Azure Service Bus package:

Install-Package WindowsAzure.ServiceBus -Version 2.5.3.0

1. Open the Web.config file in the editor and add in the following client endpoint, replacing the {YOUR NAMESPACE} placeholder:

<endpoint name="internalservice"

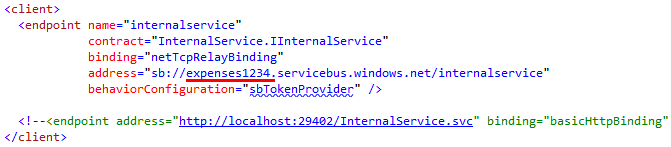
contract="InternalService.IInternalService"

binding="netTcpRelayBinding"

address="sb://{YOUR NAMESPACE}.servicebus.windows.net/internalservice"

behaviorConfiguration="sbTokenProvider" />

1. Comment out or remove the existing client endpoint. The client section should now look similar to the following:



1. Define the “sbTokenProvider” behavior configuration by copying the following snippet and pasting into the behaviors section:

<endpointBehaviors>

<behavior name="sbTokenProvider">

<transportClientEndpointBehavior>

<tokenProvider>

<sharedAccessSignature keyName="RootManageSharedAccessKey" key="{YOUR KEY}" />

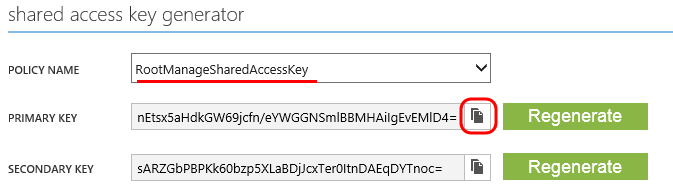
</tokenProvider>

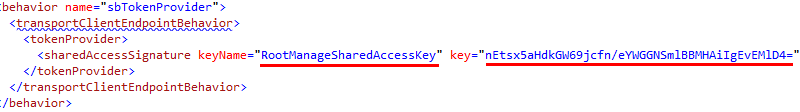
</transportClientEndpointBehavior>

</behavior>

</endpointBehaviors>

1. Replace the {YOUR KEY} placeholder with the one that was generated for the RootManageSharedAccessKey policy. We are using this policy since we also need to the Send permission.



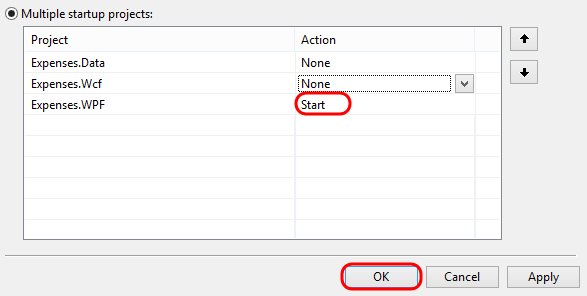


## Task 4: Publishing Updated Expenses WCF Service to Azure Website

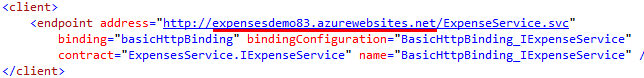
1. Publish the Expenses.Wcf project to the production Expenses website that was created in a previous demo (not the staging slot). To do this, right-click on the project and select Publish. You may need to create a new publish profile in order to publish to the production site.
2. In the browser tab that opens after successful publication, copy the address from the address bar to the clipboard.
3. Close the browser window tab and return to Visual Studio.
4. At this point, the internal service should be up and listening, and the Expenses WCF service configured to communicate via Service Bus Relay should be deployed into our production site.

## Task 5: Testing the Service Bus Relay Connectivity from Cloud to On-Premises

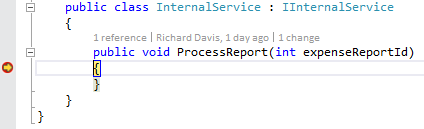
1. Right-click on the Expenses solution node and then select Set Startup Projects.
2. Ensure that the “Multiple startup projects” option is selected but that only the Expenses.WPF project is configured to “Start”. Click OK to save changes.



1. Open App.config and change the client endpoint to the address of the production service (paste in the contents of the clipboard to get the base address).



1. Press F5 to launch the Expenses WPF application.
2. Go through the expense report workflow scenario from before where you end with submitting a report and note that we hit the same breakpoint for the internal service.



1. Stop debugging in all instances of Visual Studio.