Demo - Using Service Bus Relay to Connect to On-Premises Service

Lab version: 1.0

Last updated: 1/14/2015

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



Contents

[Overview 3](#_Toc409007235)

[Prerequisites 3](#_Toc409007236)

[Setup 3](#_Toc409007237)

[Exercise 1: Creating an Internal Service 4](#_Toc409007238)

[Task 1: Overview of Internal Service 4](#_Toc409007239)

[Task 2: Add Reference to Internal Service 4](#_Toc409007240)

[Task 3: Modify ASP.NET Application to Call Internal Service 6](#_Toc409007241)

[Task 4: Demonstrating the Internal Service Running Locally 7](#_Toc409007242)

[Exercise 2: Connecting Cloud Application to On-Premises Service using Azure Service Bus Relay 11](#_Toc409007243)

[Task 1: Creating a Service Bus Namespace 11](#_Toc409007244)

[Task 2: Configure Internal Service to Listen to Service Bus Relay 15](#_Toc409007245)

[Task 3: Updating Expenses Application to Call Internal Service via Service Bus Relay 17](#_Toc409007246)

[Task 4: Publishing Updated Expenses Application to Azure Website 19](#_Toc409007247)

# Overview

In this demo, we will show how to use an Azure Service Bus Relay to connect the Expenses web application 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.Mvc codebase

# Setup

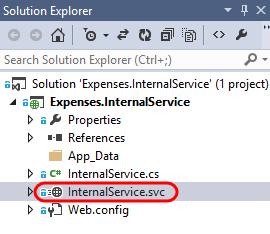
* Load and build the Expenses.Mvc solution to ensure that it builds correctly.
* Log into the Microsoft Azure subscription that you will be using for demonstration.

# Exercise 1: Creating an Internal Service

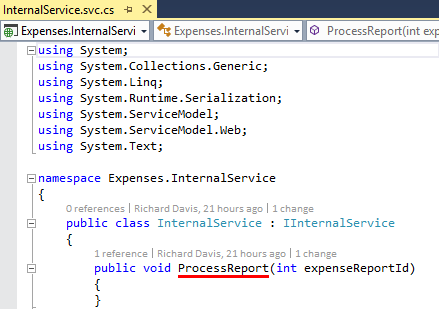
In this exercise, we will extend the Expenses scenario a bit by adding in the need for some services to remain on-premises. This exercise shows off the simple internal service definition and walks through the steps to include calls to it from the existing ASP.NET application.

## 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.

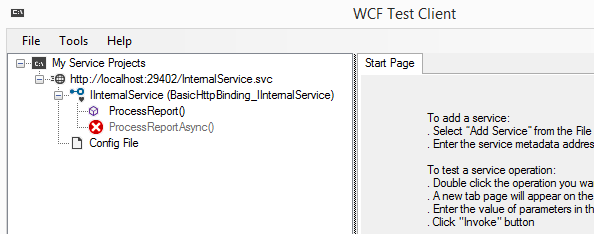


1. This service exposes an API to process submitted expense reports. It is needed because some data and processes need to remain on-premises for the time being.

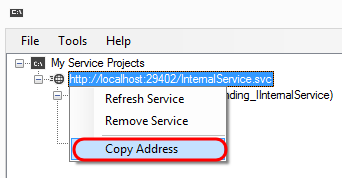


## Task 2: Add Reference to Internal Service

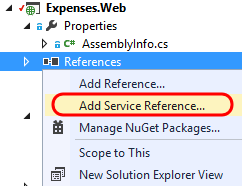
1. Select InternalService.svc in Solution Explorer and then press F5 to launch the internal service hosted locally in IIS Express. This should also launch the WCF Test Client.



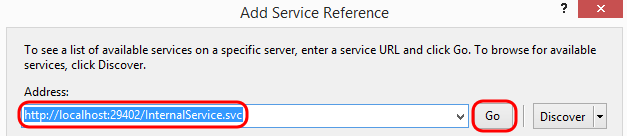
1. Right-click on the service project node for the internal service and select Copy Address.



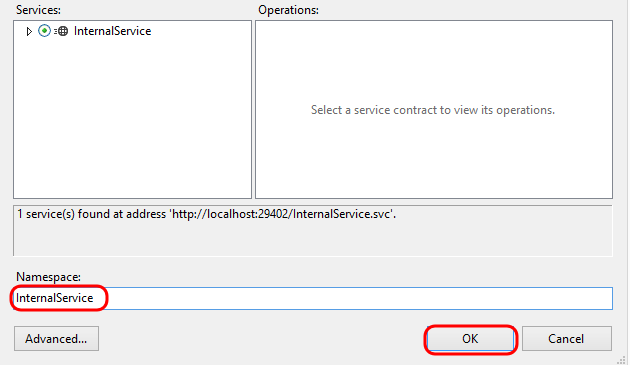
1. Load the Expenses.Mvc.sln solution in a new instance of Visual Studio.
2. In the Expenses.Web project, right-click on the References node and select Add Service Reference.



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

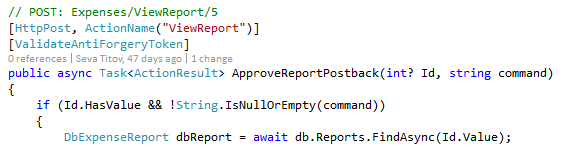


1. Note that the InternalService was found at the provided address. Change the Namespace to be InternalService and then click OK.



## Task 3: Modify ASP.NET Application to Call Internal Service

1. Double-click ExpensesController.cs to open the controller code in the editor.
2. Scroll down and locate the ApproveReportPostback method. This method is called when a manager approves an expense report.



1. Insert the following code snippet at the end of the method, just before the return statement. This will make a call to the internal service once the expense report has been approved for further processing.

if (dbReport.Status == DbExpenseReportStatus.Approved)

{

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

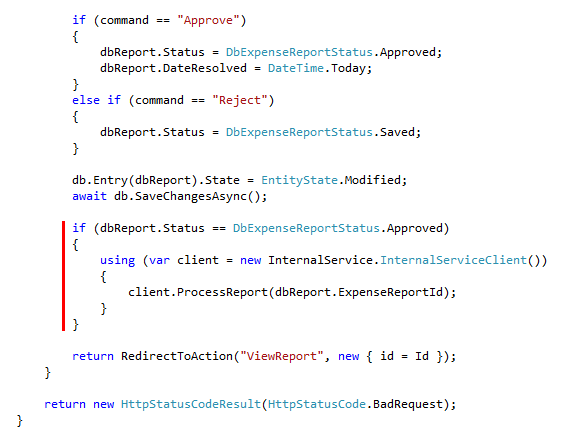
{

client.ProcessReport(dbReport.ExpenseReportId);

}

}

1. The end of the modified ApproveReportPostback method should look like the following:

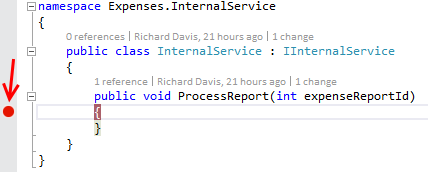


## Task 4: Demonstrating the Internal Service Running Locally

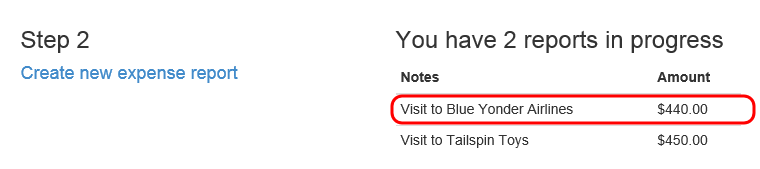
1. Ensure that connection string for the Expenses.Web project is set to be LocalDb.



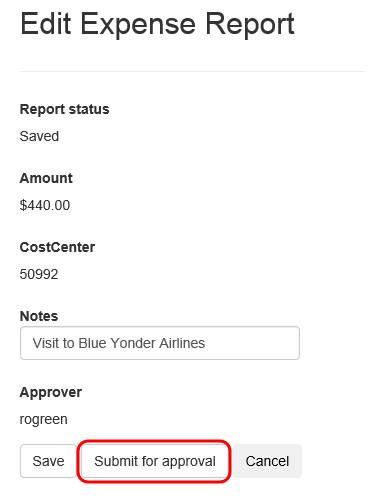
1. In the instance of Visual Studio that has the Expenses.InternalService project loaded and running, set a breakpoint within the ProcessReport method.



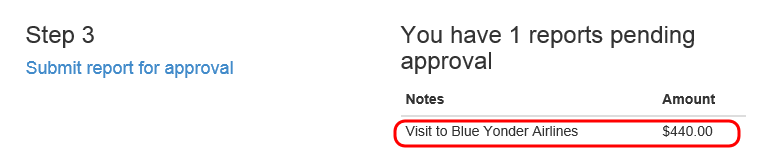
1. In the instance of Visual Studio containing the ASP.NET application, press F5 to launch.
2. Select one of the reports that are currently in progress. If you don’t have any shown as being in progress, you can reset the demo data by navigating to the Settings tab.



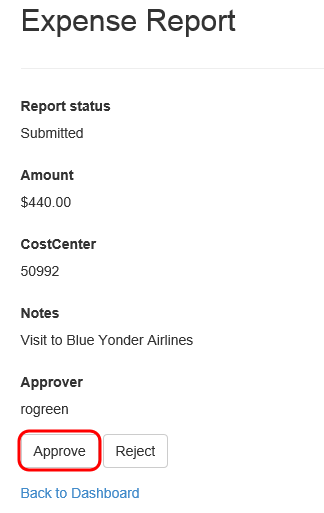
1. Click the Submit for Approval button.



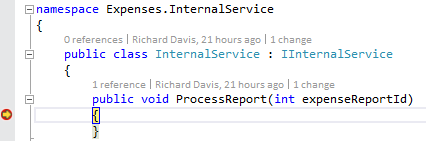
1. Click on the Home link to return to the dashboard.
2. Click on the report that is pending approval.



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 Application to On-Premises Service using Azure Service Bus Relay

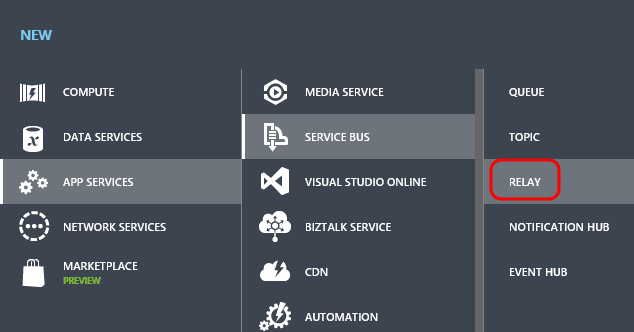
In this exercise, we will take the steps necessary to create a Service Bus Namespace, configure the new internal service running on-premises to listen using a relay, and configure the existing Expenses ASP.NET application to make calls using the relay.

## Task 1: Creating a Service Bus Namespace

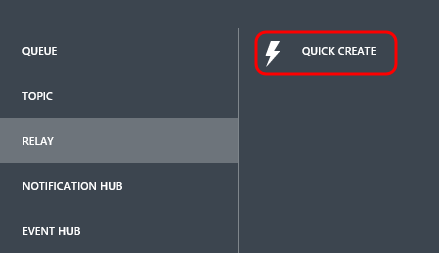
1. Log into the full Microsoft Azure portal [here](https://manage.windowsazure.com) (or if you are currently in the Preview portal at portal.azure.com you can click on your user name in the top-right corner and select the Full Azure Portal link).
2. Click the New button in the bottom-left corner.



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.



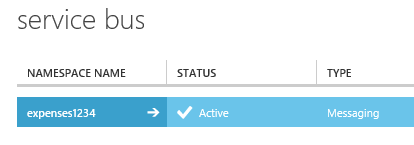
1. Select the same Region that you have been using for other Azure resources in previous demos. This isn’t required, but is generally a good idea for performance reasons.



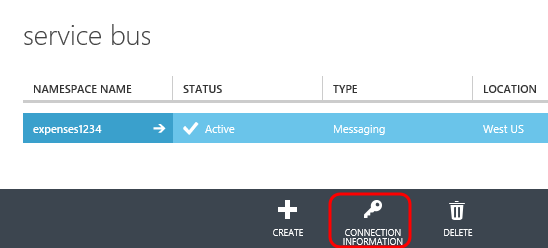
1. Select Create a Relay.



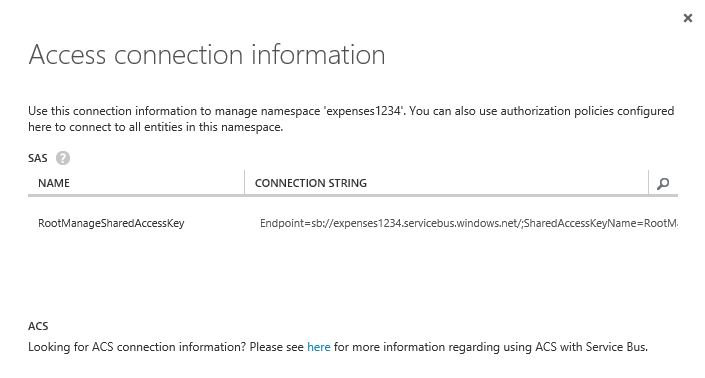
1. After a few moments the new Service Bus will be created and in the Active state.



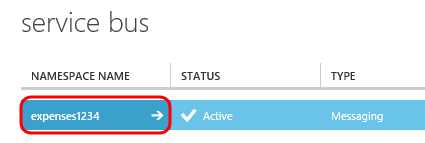
1. With the new namespace selected, click the Connection Information button near the bottom.



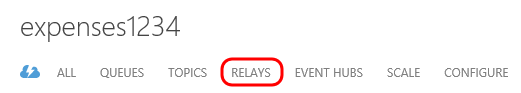
1. Namespaces created via the portal will be setup to use Shared Access Signature (SAS) authentication. This provides an access key that is then used to generate a SAS token that clients can use to authenticate. This is recommended for applications scenarios where there is no need to manage the notion 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).



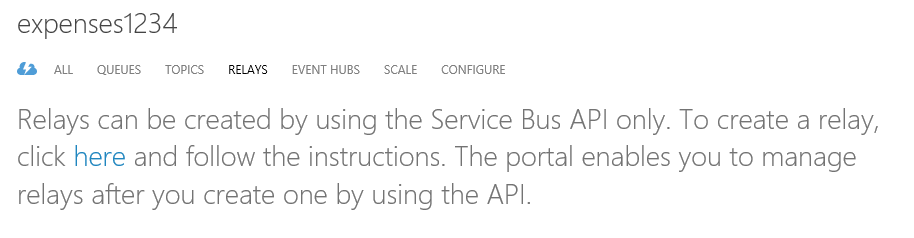
1. Press Escape to close the Access Connection Information window.
2. Drill into the new Service Bus namespace.

.

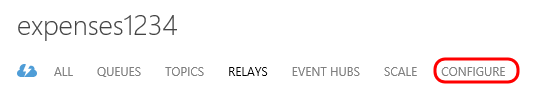
1. This namespace can contain a number of different objects in it including Queues, Topics, Relays, and so on. Click on the Relays tab.



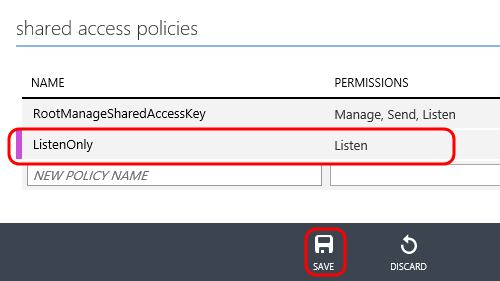
1. Relays are created using the Service Bus API, as stated by the information here.



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.



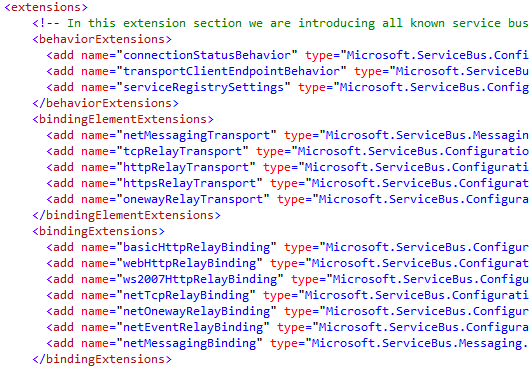
1. We will return to this configuration information shortly, so leave the window open.

## 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:

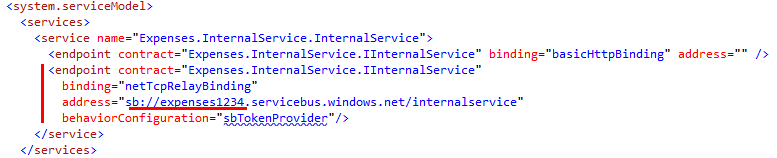
<endpoint contract="Expenses.InternalService.IInternalService"

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}" />

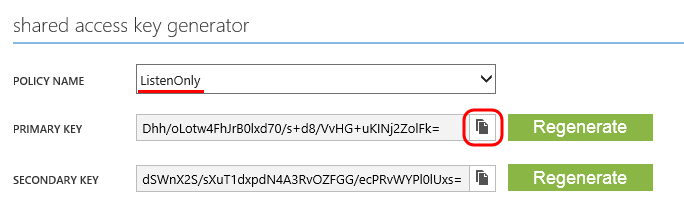
</tokenProvider>

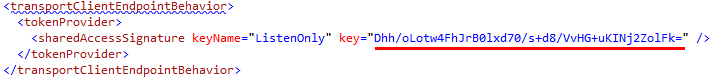
</transportClientEndpointBehavior>

</behavior>

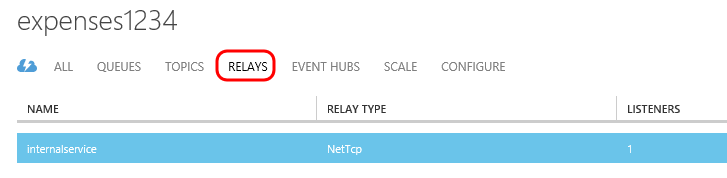
</endpointBehaviors>

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 Application to Call Internal Service via Service Bus Relay

1. Return to the instance of Visual Studio that has the Expenses.Mvc 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.Web.



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"

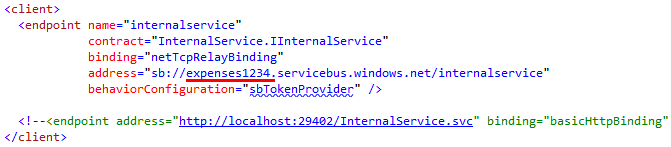
contract="InternalService.IInternalService"

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 (you may need to add the behaviors section to <system.serviceModel> if it does not exist yet):

<endpointBehaviors>

<behavior name="sbTokenProvider">

<transportClientEndpointBehavior>

<tokenProvider>

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

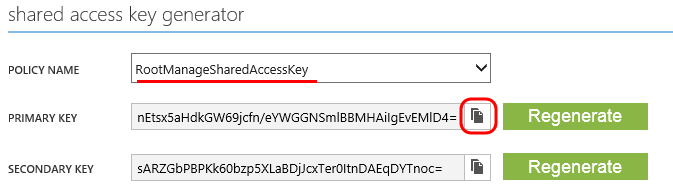
</tokenProvider>

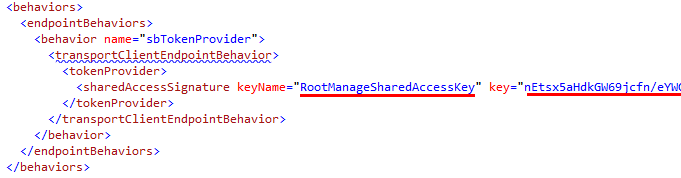
</transportClientEndpointBehavior>

</behavior>

</endpointBehaviors>

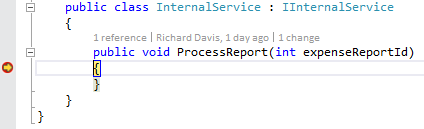
1. Replace the {YOUR KEY} placeholder with the one that was generated for the RootManageSharedAccessKey policy. We are using this policy since we also need to the Send permission.





## Task 4: Publishing Updated Expenses Application to Azure Website

1. Publish the Expenses.Web 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. Once the publication process is complete, a browser window will open and load the ASP.NET application running in Azure. Run through the steps as you previously did to submit a report for approval to verify that the Service Bus Relay is working as expected (it should hit the breakpoint in the internal service).



1. Stop debugging in all instances of Visual Studio.