

Hands-On Lab

Windows Azure: Worker Role Communication

Lab version: 1.0.0

Last updated: 12/22/2009



Contents

[Overview 3](#_Toc249249690)

[Exercise 1: Using Worker Role External Endpoints 8](#_Toc249249691)

[Task 1 – Exploring the AzureTalk Solution 9](#_Toc249249692)

[Task 2 – Hosting a WCF Service in a Worker Role 11](#_Toc249249693)

[Task 3 – Implementing the Chat Service 16](#_Toc249249694)

[Verification 24](#_Toc249249695)

[Exercise 2: Using Internal Endpoints for Inter-Role Communication 30](#_Toc249249696)

[Task 1 – Creating an Inter-Role Communication Endpoint 33](#_Toc249249697)

[Task 2 – Receiving Notifications from Other Worker Roles 41](#_Toc249249698)

[Task 3 – Sending Notifications to Other Worker Roles 43](#_Toc249249699)

[Verification 47](#_Toc249249700)

[Summary 49](#_Toc249249701)

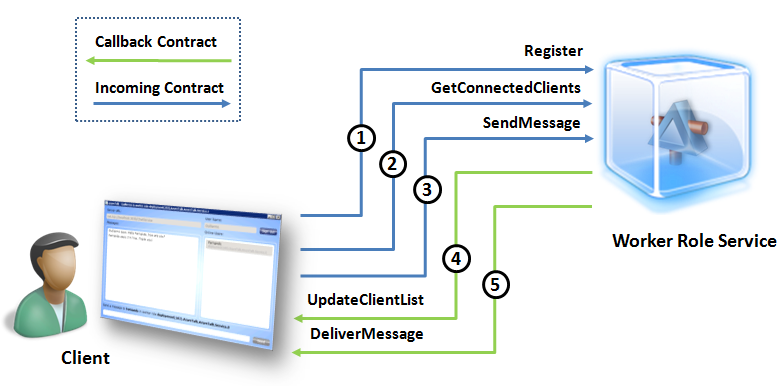
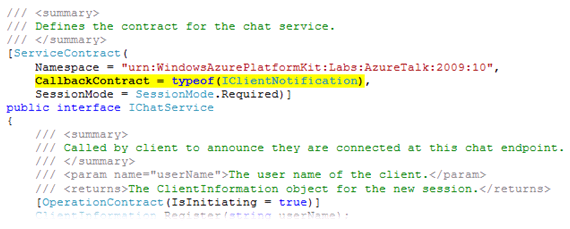
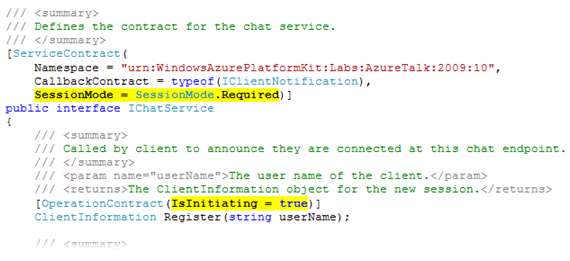
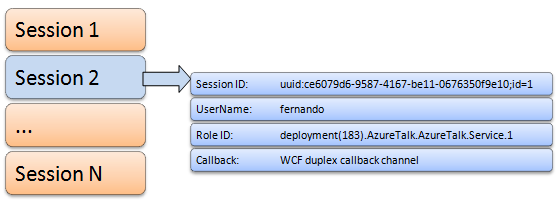
Overview

* 1. Web roles in Windows Azure provide support for HTTP and HTTPS protocols through their endpoints. For other TCP/IP protocols and ports, external endpoints allow worker roles to expose Internet-facing services over ports other than 80 and 443. These endpoints can be bound to any process and provide the means to communicate externally using a variety of TCP-based application protocols.
  2. Similarly, for inter-role communications, internal endpoints provide a way for roles to communicate directly with each other without the need to go through queues, although queues remain the preferred method for reliable asynchronous message processing.

1. In this hands-on lab, you will explore internal and external endpoints by implementing a WCF chat service and host it in a Windows Azure worker role.

**The AzureTalk Service**

In the solution, a Windows Azure worker role hosts the WCF service, which listens on an external TCP endpoint that you configure for this role. The service implements a duplex contract over a TCP channel that allows clients to register, send and receive messages, and receive notifications from the service over the same channel.

* 1. 
  2. Figure
  3. Chat service duplex contract
  4. A typical exchange between the client application and the worker role service involves the following operations:
  5. The client application announces its presence at this chat endpoint by calling the **Register** operation.
  6. The client calls the **GetConnectedClients** operation to retrieve a list of users currently online.
  7. The client calls the **SendMessage** operation to send messages to other active users.
  8. The service calls the **UpdateClientList** operation to notify the client whenever other users connect or disconnect.
  9. The service calls the **DeliverMessage** operation to forward messages sent to a client by other users.
  10. In order for the service to deliver notifications and messages from other users, the client application needs to implement a callback contract, which the service specifies in its own contract using the **CallbackContract** attribute.
  11. 
  12. Figure
  13. Duplex service specifying its callback contract
  14. The contract for the chat service indicates that it requires sessions through its **SessionMode** setting. In addition, because the contract marks the **Register** operation as being initiating by means of the **IsInitiating** attribute, whenever a client invokes this particular operation, the WCF infrastructure establishes a session with the client and assigns an identifier to this session. Thereafter, and as long as the client maintains the session open, WCF assigns the same session ID to every operation invoked by the client and makes it available in the operation context through its **SessionId** property. The chat service uses this identifier to correlate every exchange from a particular client.
  15. 
  16. Figure
  17. Service contract session requirements
  18. In addition to the WCF session, the chat service also maintains its own notion of a session, which allows it to keep track of each connected client. The service assigns a **SessionInformation** object to each active client and uses it to register information about its session. To manage sessions, the service keeps an internal dictionary of **SessionInformation** objects and uses the session ID assigned by WCF as the key to access the sessions it contains.
  19. When creating a new session during client registration, the service records the session ID, a user name for the client, and a callback channel to the client. The callback channel allows the service to invoke operations on the client application including forwarding messages from peers and alerting when other users connect or disconnect. Additionally, the session contains an ID for the worker role where the client session is active. This piece of information will become significant in Exercise 2, when you implement an inter-role communication mechanism that enables communication between clients connected to different worker roles.
  20. 
  21. Figure
  22. Managing sessions in the Chat Service

1. Because multiple clients using the service can read and update sessions concurrently, the service uses a **SessionManager** class to control access to the session information dictionary in a thread-safe manner.
   1. The code provided with the lab already contains the chat service contracts, an implementation for the Session Manager, and a client application that you can use to test the service. During the course of the lab, you will configure the internal and external endpoints of the worker role and implement the chat service.

# Objectives

* 1. In this Hands-On Lab, you will learn how to:
  + Expose non-HTTP services in Windows Azure from a worker role through external endpoints
  + Use internal endpoints for inter-role communication

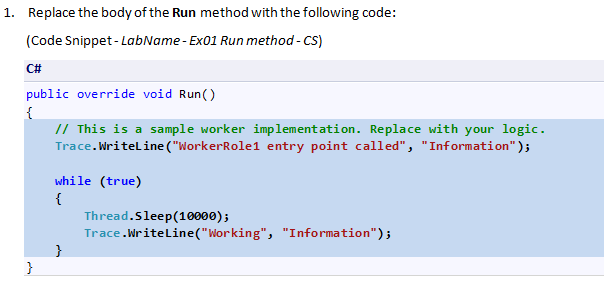
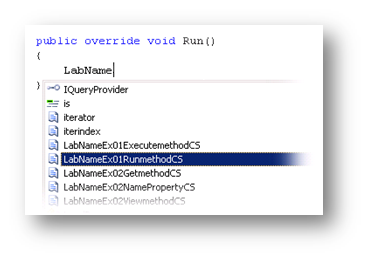
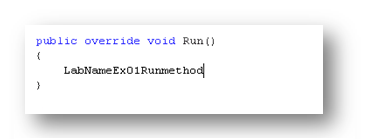
# Prerequisites

* 1. The following is required to complete this hands-on lab:
  + IIS 7 (with ASP.NET, WCF HTTP Activation)
  + [Microsoft .NET Framework 3.5 SP1](http://www.microsoft.com/downloads/details.aspx?FamilyId=AB99342F-5D1A-413D-8319-81DA479AB0D7)
  + [Microsoft Visual Studio 2008 SP1 (or above)](http://msdn.microsoft.com/vstudio/products/)
  + [Windows Azure Tools for Microsoft Visual Studio (November 2009)](http://go.microsoft.com/fwlink/?LinkID=128752)

# Setup

* 1. For convenience, much of the code used in this hands-on lab is available as Visual Studio code snippets. To check the prerequisites of the lab and install the code snippets:
  2. Run the **SetupLab.cmd** script located in the lab's **Source\Setup** folder to check dependencies and install any missing prerequisites.
  3. Once you have verified every prerequisite, follow the instructions to install the code snippets.

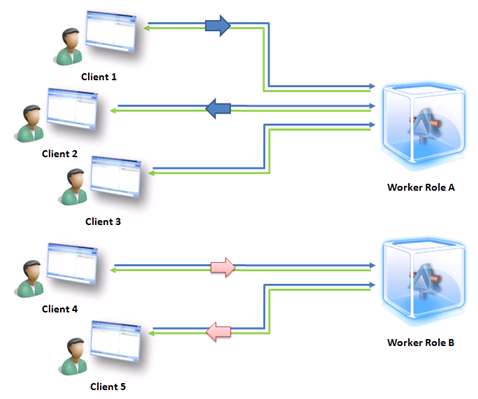
# Using the Code Snippets

* 1. With code snippets, you have all the code you need at your fingertips. The lab document will tell you exactly when you can use them. For example,
  2. 
  3. To add this code snippet in Visual Studio, you simply place the cursor where you would like the code to be inserted, start typing the snippet name (without spaces or hyphens), in this case *LabNameEx01RunmethodCS*, watch as Intellisense picks up the snippet name, and then hit the TAB key twice once the snippet you want is selected. The code will be inserted at the cursor location.
  4. 
  5. Figure
  6. Hit TAB to select the highlighted snippet.
  7. 
  8. Figure
  9. Hit TAB again and the snippet will expand
  10. To insert a code snippet using the mouse rather than the keyboard, right-click where you want the code snippet to be inserted, select **Insert Snippet** followed by **My Code Snippets** and then pick the relevant snippet from the list.
  11. To learn more about Visual Studio IntelliSense Code Snippets, including how to create your own, please see <http://msdn.microsoft.com/en-us/library/ms165392.aspx>.

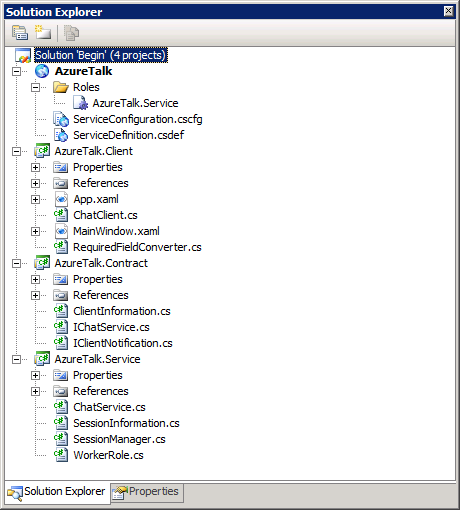
# Exercises

* 1. This Hands-On Lab comprises the following exercises:
  2. Using Worker Role External Endpoints
  3. Using Internal Endpoints for Inter-Role Communication
  4. Estimated time to complete this lab: **60 minutes**.

Exercise 1: Using Worker Role External Endpoints

* 1. During this exercise, you will implement the WCF chat service. This involves defining an external endpoint for the worker role, implementing the service contract and updating the worker role to host the service at the external endpoint.
  2. Windows Azure allows you to create as many instances of a worker role as you require, which means that you are able to host the chat service in many different nodes simultaneously. However, in this initial implementation, each instance maintains its own list of sessions. This allows clients connected to the same worker role to communicate with each other but prevents them from exchanging messages with peers that have active sessions in other worker roles. In the next exercise, you will see how to communicate worker roles and exchange session information between them.
  3. 
  4. Figure
  5. Clients can only communicate with peers connected to the same worker role

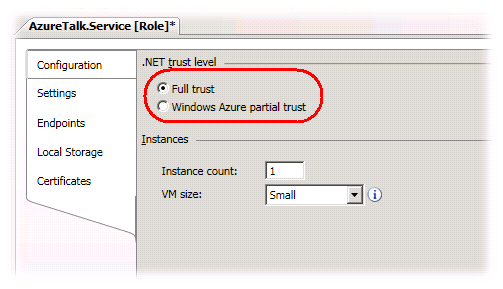
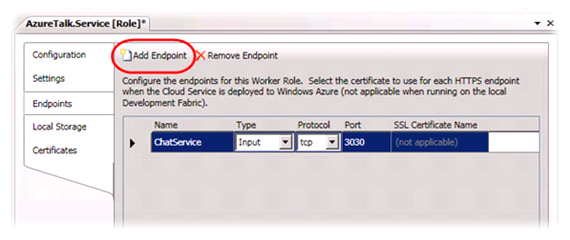
Task 1 – Exploring the AzureTalk Solution

1. In this task, you open the starting solution and become familiar with the code.
   1. Open Microsoft Visual Studio 2008 in elevated administrator mode, from **Start | All Programs | Microsoft Visual Studio 2008** by right clicking the **Microsoft Visual Studio 2008** shortcut and choosing **Run as Administrator**.
   2. In the **File** menu, choose **Open** and then **Project/Solution**. In the **Open Project** dialog, browse to **Ex1-WorkerExternalEndPoints\Begin** in the **Source** folder of the lab, select **Begin.sln** in the folder for the language of your preference (Visual C# or Visual Basic) and click **Open**.
   3. The solution contains the following projects:
      1. 
      2. Figure
      3. Solution Explorer showing the AzureTalk solution

|  |  |
| --- | --- |
| **AzureTalk** | A standard cloud service project configured to support a single worker role named **AzureTalk.Service**. |
| **AzureTalk.Client** | A WPF client application that can connect to the chat service to exchange messages with peers. It implements the service callback contract and can receive notifications from the service. In this hands-on lab, you will use this application to test the chat service. |
| **AzureTalk.Contract** | A class library project that contains the service and data contracts for the chat service and the callback contract implement by the client application. The client application and the service share this project. |
| **AzureTalk.Service** | A worker role that hosts the chat service and listens over an external TCP endpoint. |

1. ..

Task 2 – Hosting a WCF Service in a Worker Role

* 1. In this task, you configure the worker role to define an external endpoint and then create a WCF service host to listen at this endpoint.
  2. Enable full trust for the worker role. To do this, in **Solution Explorer**, expand the **Roles** node in the **AzureTalk** cloud project, right-click the **AzureTalk.Service** role and choose **Properties**. In the role **Properties** window, select the **Configuration** tab and choose the **Full Trust** option.
     1. 
     2. Figure
     3. Configuring the trust level of the worker role
  3. Define an external endpoint for the worker role. In the role **Properties** window, change to the **Endpoints** tab and click **Add Endpoint**. Set the name of the new endpoint to “*ChatService*”, leave the **Type** as “*Input*” and the **Protocol** as “*tcp*”, and then set the **Port** number to the value “**3030**”. The worker role will use this TCP endpoint to host the chat service.
     1. 
     2. Figure
     3. Defining the external endpoint for the worker role
  4. Press **CTRL + S** to save the changes to the worker role configuration.
  5. Open the **WorkerRole.cs** file (for Visual C# projects) or **WorkerRole.vb** file (for Visual Basic projects) in the **AzureTalk.Service** project. This file contains the entry point of the worker role.
  6. In the **WorkerRole** class, define a WCF **ServiceHost** member field.
     1. (Code Snippet – *Windows Azure Worker Role Communication – Ex01 ServiceHost - CS*)
     2. C#
     3. **/// <summary>ServiceHost object for internal and external endpoints.</summary>**
     4. **private ServiceHost serviceHost;**
     5. (Code Snippet – *Windows Azure Worker Role Communication – Ex01 ServiceHost - VB*)
     6. Visual Basic
     7. **''' <summary>ServiceHost object for internal and external endpoints.</summary>**
     8. **Private serviceHost As ServiceHost**
  7. Add the **StartChatService** method to the **WorkerRole** class. This method creates and configures the WCF service host instance for the chat service.
     1. (Code Snippet – *Windows Azure Worker Role Communication – Ex01 StartChatService - CS*)
     2. C#
     3. **/// <summary>**
     4. **/// Starts the service host object for the internal**
     5. **/// and external endpoints of the chat service.**
     6. **/// </summary>**
     7. **/// <param name="retries">Specifies the number of retries to**
     8. **/// start the service in case of failure.</param>**
     9. **private void StartChatService(int retries)**
     10. **{**
     11. **// recycle the role if host cannot be started**
     12. **// after the specified number of retries**
     13. **if (retries == 0)**
     14. **{**
     15. **RoleEnvironment.RequestRecycle();**
     16. **return;**
     17. **}**
     18. **Trace.TraceInformation("Starting chat service host...");**
     19. **this.serviceHost = new ServiceHost(typeof(ChatService));**
     20. **// Recover the service in case of failure.**
     21. **// Log the fault and attempt to restart the service host.**
     22. **this.serviceHost.Faulted += (sender, e) =>**
     23. **{**
     24. **Trace.TraceError("Host fault occured. Aborting and restarting the host. Retry count: {0}", retries);**
     25. **this.serviceHost.Abort();**
     26. **this.StartChatService(--retries);**
     27. **};**
     28. **// use NetTcpBinding with no security**
     29. **NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);**
     30. **// define an external endpoint for client traffic**
     31. **RoleInstanceEndpoint externalEndPoint =**
     32. **RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["ChatService"];**
     33. **this.serviceHost.AddServiceEndpoint(**
     34. **typeof(IChatService),**
     35. **binding,**
     36. **String.Format("net.tcp://{0}/ChatService", externalEndPoint.IPEndpoint));**
     37. **try**
     38. **{**
     39. **this.serviceHost.Open();**
     40. **Trace.TraceInformation("Chat service host started successfully.");**
     41. **}**
     42. **catch (TimeoutException timeoutException)**
     43. **{**
     44. **Trace.TraceError("The service operation timed out. {0}",**
     45. **timeoutException.Message);**
     46. **}**
     47. **catch (CommunicationException communicationException)**
     48. **{**
     49. **Trace.TraceError("Could not start chat service host. {0}",**
     50. **communicationException.Message);**
     51. **}**
     52. **}**

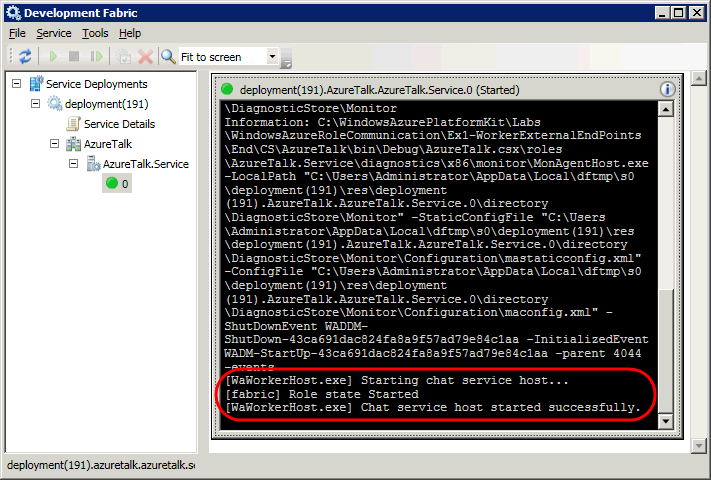
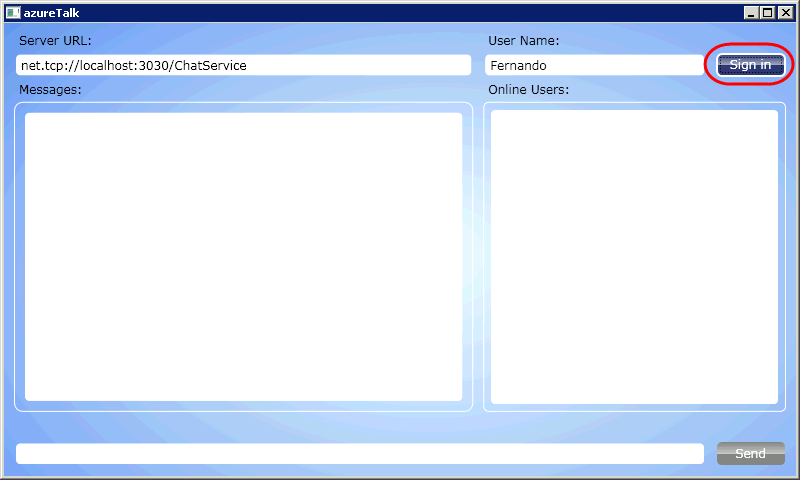
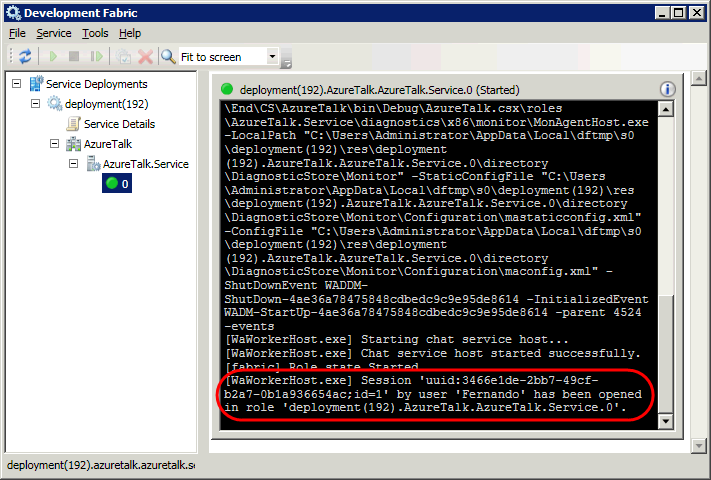
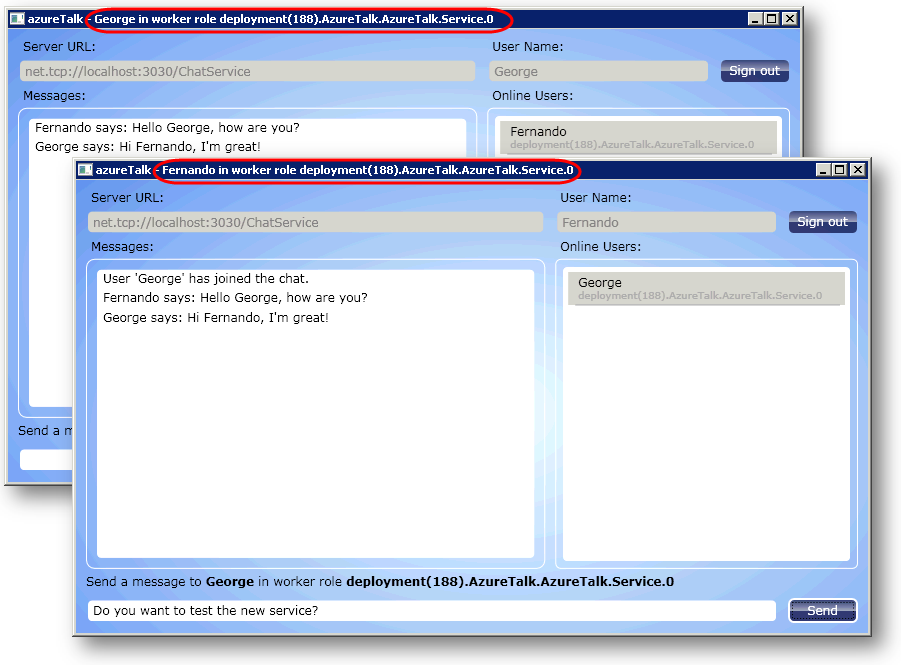
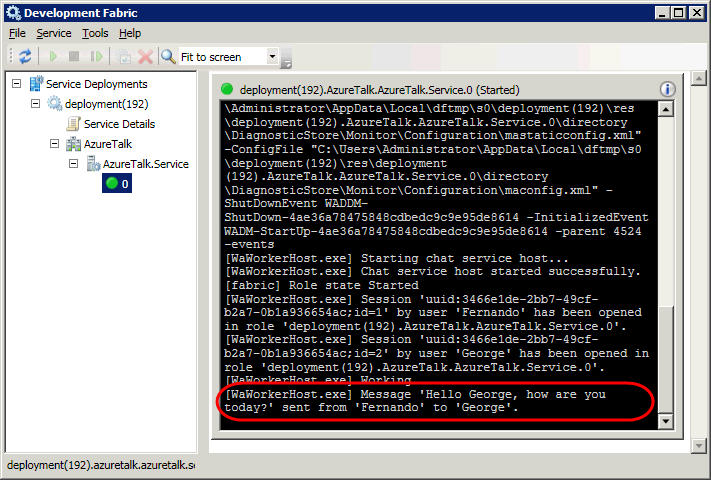
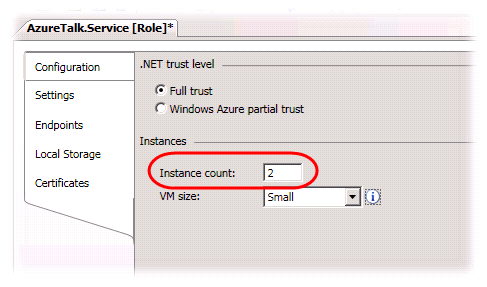
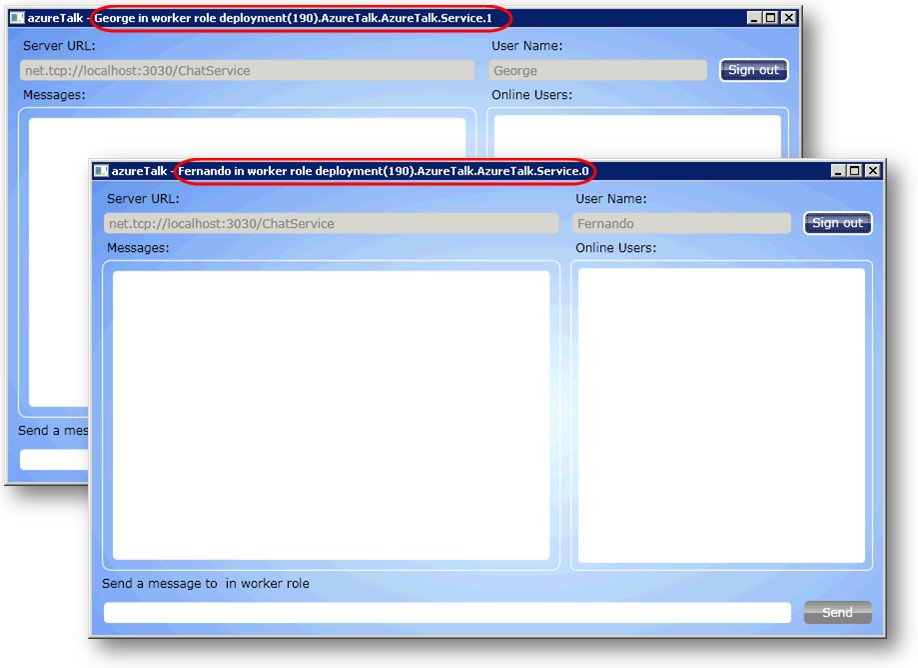
(Code Snippet – *Windows Azure Worker Role Communication – Ex01 StartChatService - VB*)

* + 1. Visual Basic
    2. **''' <summary>**
    3. **''' Starts the service host object for the internal**
    4. **''' and external endpoints of the chat service.**
    5. **''' </summary>**
    6. **''' <param name="retries">Specifies the number of retries to**
    7. **''' start the service in case of failure.</param>**
    8. **Private Sub StartChatService(ByVal retries As Integer)**
    9. **' recycle the role if host cannot be started**
    10. **' after the specified number of retries**
    11. **If retries = 0 Then**
    12. **RoleEnvironment.RequestRecycle()**
    13. **Return**
    14. **End If**
    15. **Trace.TraceInformation("Starting chat service host...")**
    16. **Me.serviceHost = New ServiceHost(GetType(ChatService))**
    17. **' Recover the service in case of failure.**
    18. **' Log the fault and attempt to restart the service host.**
    19. **AddHandler Me.serviceHost.Faulted, Function(sender, e) OnFaulted(sender, e, retries)**
    20. **' use NetTcpBinding with no security**
    21. **Dim binding As New NetTcpBinding(SecurityMode.None)**
    22. **' define an external endpoint for client traffic**
    23. **Dim externalEndPoint As RoleInstanceEndpoint = RoleEnvironment.CurrentRoleInstance.InstanceEndpoints("ChatService")**
    24. **Me.serviceHost.AddServiceEndpoint(GetType(IChatService), binding, String.Format("net.tcp://{0}/ChatService", externalEndPoint.IPEndpoint))**
    25. **Try**
    26. **Me.serviceHost.Open()**
    27. **Trace.TraceInformation("Chat service host started successfully.")**
    28. **Catch timeoutException As TimeoutException**
    29. **Trace.TraceError("The service operation timed out. {0}", timeoutException.Message)**
    30. **Catch communicationException As CommunicationException**
    31. **Trace.TraceError("Could not start chat service host. {0}", communicationException.Message)**
    32. **End Try**
    33. **End Sub**
    34. **Private Function OnFaulted(ByVal sender As Object, ByVal e As Object, ByVal retries As Integer) As Object**
    35. **Trace.TraceError("Host fault occured. Aborting and restarting the host. Retry count: {0}", retries)**
    36. **Me.serviceHost.Abort()**
    37. **retries -= 1**
    38. **Me.StartChatService(retries)**
    39. **Return Nothing**
    40. **End Function**
    41. **Note:** The **StartChatService** method creates and configures a service host to expose the chat service using the implementation provided by the **ChatService** class.
    42. The code configures a single endpoint for the contract defined by the **IChatService** interface in the **AzureTalk.Contract** project using a **NetTcpBinding** binding to enable TCP message delivery using a binary encoding. For this service, the binding configuration specifies no transport security.
    43. To determine the service endpoint address, the method uses the **RoleEnvironment** to obtain a reference the “**ChatService**” endpoint for the current instance, which you previously defined for the worker role. Note that this address is the internal address as seen by the worker role instance located behind the load balancer.
    44. To provide a certain degree of fault tolerance, the method subscribes to the **Faulted** event of the service host to restart the service in case of failure and attempts its recovery by re-starting the host. After a number of failed retries, the worker role requests to be recycled. Note that for a Visual Basic project, the handler for the Faulted event is located in a separate method, unlike the C# code, which uses a lambda expression to define the event handler.
  1. Next, update the **Run** method of the worker role to create and start the chat service. To do this, insert a call to the **StartChatService** method as shown (highlighted) in the code below.
     1. C#
     2. public override void Run()
     3. {
     4. Trace.TraceInformation("Worker Process entry point called.");
     5. **this.StartChatService(3);**
     6. while (true)
     7. {
     8. Thread.Sleep(300000);
     9. Trace.TraceInformation("Working...");
     10. }
     11. }
     12. Visual Basic
     13. Public Overrides Sub Run()
     14. Trace.TraceInformation("Worker Process entry point called.")
     15. **Me.StartChatService(3)**
     16. Do
     17. Thread.Sleep(300000)
     18. Trace.TraceInformation("Working...")
     19. Loop
     20. End Sub
  2. Press **CTRL + S** to save the changes to the file.

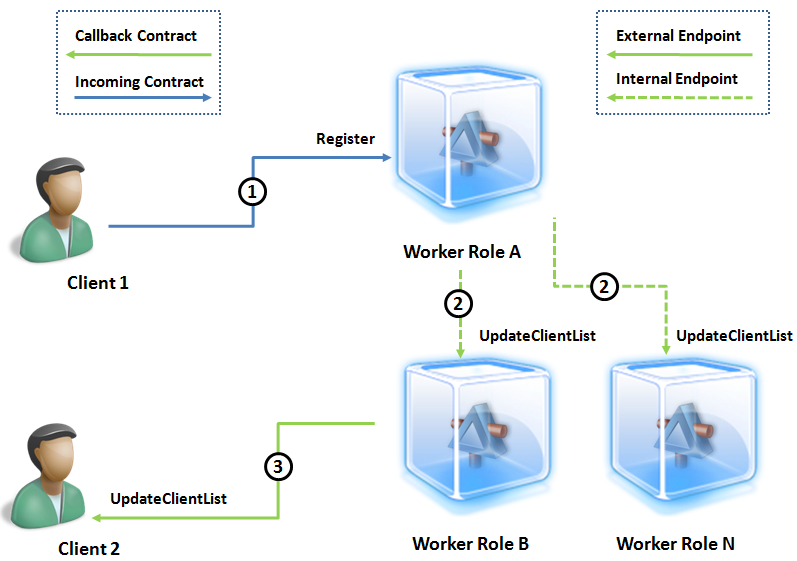
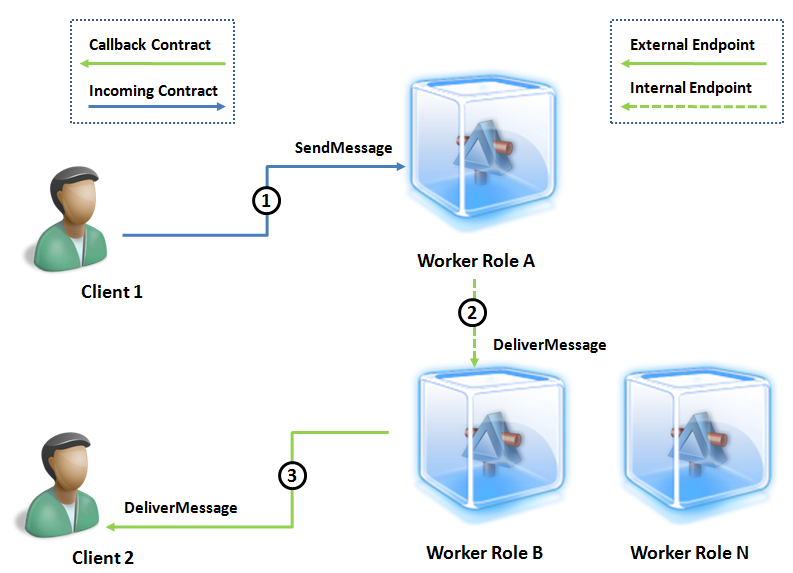
Task 3 – Implementing the Chat Service

* 1. In this task, you implement the chat service as specified in the contract defined by the **IChatService** interface of the **AzureTalk.Contract** project.
  2. Open **ChatService.cs** (for Visual C# projects) or **ChatService.vb** (for Visual Basic projects) in the **AzureTalk.Service** project. This file contains a skeleton implementation of the chat service contract. In the next steps, you will implement the service operations.
  3. Locate the **Register** operation and replace its body with the following code.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 Register - CS*)
     2. C#
     3. /// <summary>
     4. /// Called by clients to announce they are connected at this chat endpoint.
     5. /// </summary>
     6. /// <param name="userName">The user name of the client.</param>
     7. /// <returns>The ClientInformation object for the new session.</returns>
     8. public ClientInformation Register(string userName)
     9. {
     10. **// retrieve session information**
     11. **string roleId = RoleEnvironment.CurrentRoleInstance.Id;**
     12. **string sessionId = OperationContext.Current.SessionId;**
     13. **IClientNotification callback = OperationContext.Current.GetCallbackChannel<IClientNotification>();**
     14. **SessionInformation session;**
     15. **if (SessionManager.CreateOrUpdateSession(sessionId, userName, roleId, callback, out session))**
     16. **{**
     17. **// ensure that the session is killed when channel is closed**
     18. **OperationContext.Current.Channel.Closed += (sender, e) =>**
     19. **{**
     20. **SessionManager.RemoveSession(sessionId);**
     21. **NotifyConnectedClients(session);**
     22. **Trace.TraceInformation("Session '{0}' by user '{1}' has been closed in role '{2}'.", sessionId, userName, roleId);**
     23. **};**
     24. **Trace.TraceInformation("Session '{0}' by user '{1}' has been opened in role '{2}'.", sessionId, userName, roleId);**
     25. **}**
     26. **// Notify clients connected to this role**
     27. **NotifyConnectedClients(session);**
     28. **return new ClientInformation()**
     29. **{**
     30. **SessionId = sessionId,**
     31. **UserName = userName,**
     32. **RoleId = roleId**
     33. **};**
     34. }
     35. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 Register - VB*)
     36. Visual Basic
     37. ''' <summary>
     38. ''' Called by clients to announce they are connected at this chat endpoint.
     39. ''' </summary>
     40. ''' <param name="userName">The user name of the client.</param>
     41. ''' <returns>The ClientInformation object for the new session.</returns>
     42. Public Function Register(ByVal userName As String) As ClientInformation Implements IChatService.Register
     43. **' retrieve session information**
     44. **Dim roleId As String = RoleEnvironment.CurrentRoleInstance.Id**
     45. **Dim sessionId As String = OperationContext.Current.SessionId**
     46. **Dim callback As IClientNotification = OperationContext.Current.GetCallbackChannel(Of IClientNotification)()**
     47. **Dim session As SessionInformation**
     48. **If SessionManager.CreateOrUpdateSession(sessionId, userName, roleId, callback, session) Then**
     49. **' ensure that the session is killed when channel is closed**
     50. **AddHandler OperationContext.Current.Channel.Closed, Function(sender, e) OnClosed(sender, e, sessionId, session, userName, roleId)**
     51. **Trace.TraceInformation("Session '{0}' by user '{1}' has been opened in role '{2}'.", sessionId, userName, roleId)**
     52. **End If**
     53. **' Notify clients connected to this role**
     54. **NotifyConnectedClients(session)**
     55. **Return New ClientInformation() With {.SessionId = sessionId, .UserName = userName, .RoleId = roleId}**
     56. End Function
     57. **Note:**  The **Register** implementation obtains the ID of the current worker role and the ID for the session established by the WCF infrastructure. In addition, it retrieves the callback channel to the client instance that called the service, which it can use to call operations on the client through its **IClientNotify** callback contract. With this information, the service calls the Session Manager to either create a new session or, if the client has registered previously, return an existing session.
     58. For new sessions, the method attaches a handler to the **Closed** event of the channel to remove the session once the channel is closed.
     59. The code invokes **NotifyConnectedClients**, both in the main body and in the handler for the **Closed** event, to inform other clients connected to this worker role that a new session has started or ended.
  4. If you implement your code in Visual Basic, add the **OnClosed** function.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 OnClosed - VB*)
     2. Visual Basic
     3. **Private Function OnClosed(ByVal sender As Object, ByVal e As Object, ByVal sessionId As String, ByVal session As SessionInformation, ByVal userName As String, ByVal roleId As String) As Object**
     4. **SessionManager.RemoveSession(sessionId)**
     5. **NotifyConnectedClients(session)**
     6. **Trace.TraceInformation("Session '{0}' by user '{1}' has been closed in role '{2}'.", sessionId, userName, roleId)**
     7. **Return Nothing**
     8. **End Function**
  5. Next, replace the body of the **SendMessage** operation with the code shown below.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 SendMessage - CS*)
     2. C#
     3. /// <summary>
     4. /// Sends a message to a user.
     5. /// </summary>
     6. /// <param name="message">The message to send.</param>
     7. /// <param name="sessionId">The recipient's session Id.</param>
     8. public void SendMessage(string message, string sessionId)
     9. {
     10. **string fromSessionId = OperationContext.Current.SessionId;**
     11. **this.DeliverMessage(message, fromSessionId, sessionId);**
     12. }
     13. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 SendMessage - VB*)
     14. Visual Basic
     15. ''' <summary>
     16. ''' Sends a message to a user.
     17. ''' </summary>
     18. ''' <param name="message">The message to send.</param>
     19. ''' <param name="sessionId">The recipient's session Id.</param>
     20. Public Sub SendMessage(ByVal message As String, ByVal sessionId As String) Implements IChatService.SendMessage
     21. **Dim fromSessionId As String = OperationContext.Current.SessionId**
     22. **Me.DeliverMessage(message, fromSessionId, sessionId)**
     23. End Sub
     24. **Note:** The **SendMessage** operation retrieves the current session ID from the execution context of the service method, which is the same as the one used by the **Register** operation to register the client with the Session Manager, and then calls the **DeliverMessage** method to send the message to its destination. You will implement the **DeliverMessage** method shortly.
  6. Finally, to complete the implementation of the contract, replace the body of the **GetConnectedClients** operation with the code shown below, which returns the list of active client sessions registered with the Session Manager.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 GetConnectedClients - CS*)
     2. C#
     3. /// <summary>
     4. /// Returns a list of connected clients.
     5. /// </summary>
     6. /// <returns>The list of active sessions.</returns>
     7. public IEnumerable<ClientInformation> GetConnectedClients()
     8. {
     9. **return from session in SessionManager.GetActiveSessions()**
     10. **select new ClientInformation()**
     11. **{**
     12. **SessionId = session.SessionId,**
     13. **UserName = session.UserName,**
     14. **RoleId = session.RoleId**
     15. **};**
     16. }
     17. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 GetConnectedClients - VB*)
     18. Visual Basic
     19. ''' <summary>
     20. ''' Returns a list of connected clients.
     21. ''' </summary>
     22. ''' <returns>The list of active sessions.</returns>
     23. Public Function GetConnectedClients() As IEnumerable(Of ClientInformation) Implements IChatService.GetConnectedClients
     24. **Return \_**
     25. **From session In SessionManager.GetActiveSessions() \_**
     26. **Select New ClientInformation() With {.SessionId = session.SessionId, .UserName = session.UserName, .RoleId = session.RoleId}**
     27. End Function
     28. **Note:** The code retrieves a list of active sessions from the Session Manager and returns a projection of **ClientInformation** objects. These objects, which are defined in the data contract of the service, contain a subset of the session information more suitable for transmission to clients.
  7. Add a **DeliverMessage** method to the **ChatService** class. This method sends messages to clients connected to the worker role.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 DeliverMessage - CS*)
     2. C#
     3. **/// <summary>**
     4. **/// Delivers a message to a client in the current worker role.**
     5. **/// </summary>**
     6. **/// <param name="message">The message to forward.</param>**
     7. **/// <param name="fromSessionId">The session ID of the message originator.</param>**
     8. **/// <param name="toSessionId">The session ID of the message recipient.</param>**
     9. **public void DeliverMessage(string message, string fromSessionId, string toSessionId)**
     10. **{**
     11. **SessionInformation fromSession = SessionManager.GetSession(fromSessionId);**
     12. **SessionInformation toSession = SessionManager.GetSession(toSessionId);**
     13. **if ((fromSession != null) && (toSession != null))**
     14. **{**
     15. **// retrieve the callback channel to the client**
     16. **IClientNotification callback = toSession.Callback;**
     17. **if (callback != null)**
     18. **{**
     19. **callback.DeliverMessage(message, fromSessionId, toSessionId);**
     20. **Trace.TraceInformation("Message '{0}' sent from '{1}' to '{2}'.",**
     21. **message, fromSession.UserName, toSession.UserName);**
     22. **}**
     23. **}**
     24. **}**
     25. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 DeliverMessage - VB*)
     26. Visual Basic
     27. **''' <summary>**
     28. **''' Delivers a message to a client in the current worker role.**
     29. **''' </summary>**
     30. **''' <param name="message">The message to forward.</param>**
     31. **''' <param name="fromSessionId">The session ID of the message originator.</param>**
     32. **''' <param name="toSessionId">The session ID of the message recipient.</param>**
     33. **Public Sub DeliverMessage(ByVal message As String, ByVal fromSessionId As String, ByVal toSessionId As String)**
     34. **Dim fromSession As SessionInformation = SessionManager.GetSession(fromSessionId)**
     35. **Dim toSession As SessionInformation = SessionManager.GetSession(toSessionId)**
     36. **If (fromSession IsNot Nothing) AndAlso (toSession IsNot Nothing) Then**
     37. **' retrieve the callback channel to the client**
     38. **Dim callback As IClientNotification = toSession.Callback**
     39. **If callback IsNot Nothing Then**
     40. **callback.DeliverMessage(message, fromSessionId, toSessionId)**
     41. **Trace.TraceInformation("Message '{0}' sent from '{1}' to '{2}'.", message, fromSession.UserName, toSession.UserName)**
     42. **End If**
     43. **End If**
     44. **End Sub**
     45. **Note:** The code in this method retrieves the source and target sessions and then uses the callback channel to the recipient, which is stored in the target session, to deliver the message over the duplex channel. To do this, it invokes the **DeliverMessage** operation of the **IClientNotification** contract implemented by the client.
  8. As a final step, add the **NotifyConnectedClients** method to the class. This method notifies clients connected to this worker role about session activities, namely clients connecting and disconnecting.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 NotifyConnectedClients - CS*)
     2. C#
     3. **/// <summary>**
     4. **/// Notifies clients connected to this worker role to update their active sessions list when a new client connects or disconnects.**
     5. **/// </summary>**
     6. **/// <param name="clientInfo">The ClientInformation object for the client.</param>**
     7. **private static void NotifyConnectedClients(ClientInformation clientInfo)**
     8. **{**
     9. **foreach (SessionInformation client in SessionManager.GetActiveSessions())**
     10. **{**
     11. **if (client.Callback != null)**
     12. **{**
     13. **try**
     14. **{**
     15. **client.Callback.UpdateClientList(clientInfo);**
     16. **}**
     17. **catch (TimeoutException timeoutException)**
     18. **{**
     19. **Trace.TraceError("Unable to notify client '{0}'. The service operation timed out. {1}", client.UserName, timeoutException.Message);**
     20. **((ICommunicationObject)client).Abort();**
     21. **client.Callback = null;**
     22. **}**
     23. **catch (CommunicationException communicationException)**
     24. **{**
     25. **Trace.TraceError("Unable to notify client '{0}'. There was a communication problem. {1} - {2}", client.UserName, communicationException.Message, communicationException.StackTrace);**
     26. **((ICommunicationObject)client).Abort();**
     27. **client.Callback = null;**
     28. **}**
     29. **}**
     30. **}**
     31. **}**
     32. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex01 NotifyConnectedClients - VB*)
     33. Visual Basic
     34. **''' <summary>**
     35. **''' Notifies clients connected to this worker role to update their active sessions list when a new client connects or disconnects.**
     36. **''' </summary>**
     37. **''' <param name="clientInfo">The ClientInformation object for the client.</param>**
     38. **Private Shared Sub NotifyConnectedClients(ByVal clientInfo As ClientInformation)**
     39. **For Each client As SessionInformation In SessionManager.GetActiveSessions()**
     40. **If client.Callback IsNot Nothing Then**
     41. **Try**
     42. **client.Callback.UpdateClientList(clientInfo)**
     43. **Catch timeoutException As TimeoutException**
     44. **Trace.TraceError("Unable to notify client '{0}'. The service operation timed out. {1}", client.UserName, timeoutException.Message)**
     45. **CType(client, ICommunicationObject).Abort()**
     46. **client.Callback = Nothing**
     47. **Catch communicationException As CommunicationException**
     48. **Trace.TraceError("Unable to notify client '{0}'. There was a communication problem. {1} - {2}", client.UserName, communicationException.Message, communicationException.StackTrace)**
     49. **CType(client, ICommunicationObject).Abort()**
     50. **client.Callback = Nothing**
     51. **End Try**
     52. **End If**
     53. **Next client**
     54. **End Sub**
     55. **Note:** The **NotifyConnectedClients** method retrieves a list of active sessions from the Session Manager and then iterates over this list, invoking the **UpdateClientList** operation of the callback contract to notify clients when a client connects or disconnects.
     56. Only clients connected to the current worker role have a valid **Callback** member in the **SessionInformation** object; for clients in other worker roles this member is null.

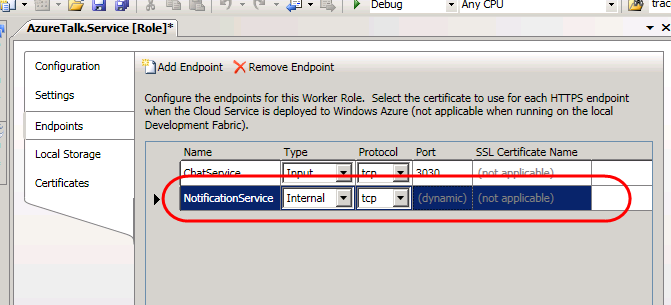
Verification

* 1. You are now ready to test the solution. First, you will run the verification against a single instance of the worker role. To test the service, you will start two instances of the client application and exchange messages between them to determine that messages and notifications flow between clients connected to the service in the worker role. Next, you will repeat the procedure after increasing the number of running instances of the worker role to two.
  2. Press **F5** to launch the cloud project in the development fabric.
  3. Switch to the development fabric UI and ensure that the service has started successfully. Notice that only a single instance of the worker role is currently active.
     1. 
     2. Figure
     3. Chat service executing successfully in the development fabric
  4. In **Solution Explorer**, right-click the **AzureTalk.Client** project, point to **Debug** and select **Start new instance**.
  5. In the main window of the application, enter a user name for the first client and click **Sign In**. Note that at this point, this is the only client connected to the service and the **Online Users** list is predictably empty.
     1. 
     2. Figure
     3. Signing in to the chat service
     4. **Note:** The following instructions assume that you are deploying locally and testing against the service running in the development fabric. You may wish to test the service running in Windows Azure, in which case, after you have deployed the service package and before you sign in, you will need to enter the URL that points to your deployment in the client application **Server URL** field as shown below:
     5. ***net.tcp://<YOUR\_DEPLOYED\_PROJECT>.cloudapp.net:3030/ChatService***
     6. where ***<YOUR\_DEPLOYED\_PROJECT>*** is the name you chose for your service deployment. For example,
     7. ***net.tcp://azureTalk.cloudapp.net:3030/ChatService***
  6. In the development fabric UI, select the worker role instance to display its log and view the entry for the new session that started.
     1. 
     2. Figure
     3. Worker role log showing the new session
  7. Start a second instance of the client application and sign in using a different user name. Notice that the title bar of the application shows the name of the current user and the ID of the worker role where the client is connected. Both clients should display the same connected role ID because there is only one instance of the worker role currently running.
     1. 
     2. Figure
     3. Verifying the connected worker role ID of the client
  8. In the second client, notice how the list of online users includes the name of the user registered by the first client. Switch to the first instance of the client application and see how the server immediately alerted this instance about the new session started by the other client.
  9. In one of the instances of the client application, select a user from the list of online users, type a message and click **Send**. Switch to the other instance to see the message delivered.
  10. Optionally, start additional instances and exchange messages with different users. In the development fabric UI, view the messages that users exchange.
      1. 
      2. Figure
      3. Worker role log showing messages exchanged by clients
  11. Finally, click **Sign out** in one of the instances of the client application. Switch to the other instance and notice how the server immediately notified the client about the session that ended.
  12. In the development fabric, view the worker role log to see an event logged for the ending session.
  13. Press **Shift + F5** to stop the application running in the development fabric.
  14. Next, you will configure the service to start a second instance of the worker role and repeat the previous steps.
  15. In **Solution Explorer**, expand the **Roles** node in the **AzureTalk** project, right-click the **AzureTalk.Service** role and select **Properties**. In the **Properties** window, select the **Configuration** tab and increase the value of the **Instance count** to **2**.
      1. 
      2. Figure
      3. Configuring additional instances of the worker role
  16. Press **F5** to launch the cloud project in the development fabric.
  17. In **Solution Explorer**, right-click the **AzureTalk.Client** project, point to **Debug** and select **Start new instance**.
  18. In the main window of the application, enter a user name for the first client and click **Sign In**.
  19. Start a second instance of the client application and sign in using a different user name. Ensure that the second client connects to a different worker role than the first client by verifying the connected role ID of each client in the title bar of the application.
      1. **Note:** The load balancer determines which instance of the worker role responds to a client's request. The development fabric typically assigns connections in round robin fashion, so the second client should normally start a session in a different worker role when it connects. If necessary, restart one of the client instances until both clients have sessions in different worker roles.
  20. Notice that the list of active users in either client does not show the other client so they are unable to exchange messages.
      1. 
      2. Figure
      3. Clients connected to different worker roles cannot communicate
  21. Start a third instance of the client and sign in. Notice that this time, the client connected to the same instance of the worker role as the new client is shown in the list of online users and vice versa.
      1. **Note:** In the current implementation, worker roles do not share session information and consequently, behave as independent servers. Clients connected to one role cannot exchange messages with clients in any other role. In the next exercise, you will remove this restriction by enabling worker roles to exchange messages and session information and allow all clients to communicate, regardless of the worker role where they establish their session.

Exercise 2: Using Internal Endpoints for Inter-Role Communication

* 1. So far, even though multiple instances of the worker role can each host its own endpoint for the chat service, they all operate as independent servers. Clients connected to a given instance are unable to exchange messages with peers in other worker role instances because roles do not share session information.
  2. In this exercise, you extend the worker role implementation to allow different instances to exchange session information and client notifications over an internal endpoint. To forward messages and session activation events, worker roles need to implement not only the incoming contract of the chat service but also the callback contract implemented by clients. This change allows roles to act as a bridge between peers active in different roles and changes the flow of information in the following manner.
  3. **Client Registration**
  4. During registration, worker roles notify every other worker role through their internal endpoints. Each alerted worker role then notifies its directly connected clients about the new session.
  5. 
  6. Figure
  7. Roles exchanging session information via an internal endpoint
  8. **Client 1** calls the **Register** operation to start a new session in **Worker Role A**.
  9. **Worker Role A** registers the session with its Session Manager and then calls **UpdateClientList** in each instance of the internal channel endpoint to notify other worker roles about the new session.
  10. **Worker Role B** receives the notification and then calls **UpdateClientList** in the callback channel to notify **Client 2** that **Client 1** has connected.
  11. **Sending Messages**
  12. When sending messages to peers with session in other worker roles, clients first send the message to their role, which then forwards the message through the callback interface to the second role using an internal endpoint. The target role then delivers the message to the recipient using the callback channel to the client. In this case, only the worker roles for the origin and destination are involved in the exchange.
  13. 
  14. Figure
  15. Users connected to different worker roles exchanging messages
  16. **Client 1** calls **SendMessage** to send the message to **Worker Role A** indicating **Client 2** as the recipient.
  17. **Worker Role A** queries the Session Manager, determines that **Client 2** has a session in **Worker Role B** and then calls **DeliverMessage** over the internal channel endpoint to forward the message only to this role.
  18. **Worker Role B** receives the message and then calls **DeliverMessage** to deliver the message to **Client 2** over the callback channel.

Task 1 – Creating an Inter-Role Communication Endpoint

* 1. In this task, you configure the worker role to define an internal endpoint. Next, you update the service host configuration to add a new WCF service endpoint using the callback channel contract and set it to listen at the address provided by the internal endpoint.
  2. If it is not already open, launch Microsoft Visual Studio 2008 in elevated administrator mode, from **Start | All Programs | Microsoft Visual Studio 2008** by right clicking the **Microsoft Visual Studio 2008** shortcut and choosing **Run as Administrator**.
  3. In the **File** menu, choose **Open** and then **Project/Solution**. In the **Open Project** dialog, browse to **Ex2-InterRoleCommunication\Begin** in the **Source** folder of the lab, select **Begin.sln** in the folder for the language of your preference (Visual C# or Visual Basic) and click **Open**. Alternatively, you may continue with the solution that you obtained after completing **Exercise 1**.
  4. Define an internal endpoint for the worker role. To do this, in **Solution Explorer**, expand the **Roles** node in the **AzureTalk** cloud project, right-click the **AzureTalk.Service** role and choose **Properties**. In the role **Properties** window, change to the **Endpoints** tab and click **Add Endpoint**. Set the name of the new endpoint to “*NotificationService*”, change the **Type** to “*Internal*” and leave the **Protocol** as “*tcp*”. The worker role will use this TCP endpoint to receive notifications from other worker roles.
     1. 
     2. Figure
     3. Defining an internal endpoint for the worker role
  5. Press **CTRL + S** to save the changes to the worker role configuration.
  6. Open the **WorkerRole.cs** file(for Visual C# projects) or **WorkerRole.vb** file (for Visual Basic projects) in the **AzureTalk.Service** project.
  7. In the **WorkerRole** class, define a WCF **ChannelFactory** member that the worker role will use to create the channel objects for communicating with other roles.
     1. (Code Snippet – *Windows Azure Worker Role Communication – Ex02 ChannelFactory - CS*)
     2. C#
     3. public class WorkerRole : RoleEntryPoint
     4. {
     5. **/// <summary>Channel factory for inter-role notifications.</summary>**
     6. **private static ChannelFactory<IClientNotification> factory;**
     7. /// <summary>ServiceHost object for internal and external endpoints.</summary>
     8. private ServiceHost serviceHost;
     9. ...
     10. (Code Snippet – *Windows Azure Worker Role Communication – Ex02 ChannelFactory - VB*)
     11. Visual Basic
     12. Public Class WorkerRole
     13. Inherits RoleEntryPoint
     14. **' <summary>Channel factory for inter-role notifications.</summary>**
     15. **Private Shared factory As ChannelFactory(Of IClientNotification)**
     16. ' <summary>ServiceHost object for internal and external endpoints.</summary>
     17. Private serviceHost As ServiceHost
     18. ...
     19. **Note:** The worker role caches the channel factory, which is a thread-safe object, in a static member to avoid the cost of re-creating it each time it needs to communicate with other worker roles.
  8. Update the **StartChatService** method to configure a new internal endpoint for the service host and create the channel factory for inter-role communication. To do this, insert the following (highlighted) code immediately after the code that defines the external endpoint.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 InternalEndpoint - CS*)
     2. C#
     3. /// <summary>
     4. /// Starts the service host object for the internal
     5. /// and external endpoints of the chat service.
     6. /// </summary>
     7. /// <param name="retries">Specifies the number of retries to
     8. /// start the service in case of failure.</param>
     9. private void StartChatService(int retries)
     10. {
     11. ...
     12. // define an external endpoint for client traffic
     13. RoleInstanceEndpoint externalEndPoint =
     14. RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["ChatService"];
     15. this.serviceHost.AddServiceEndpoint(
     16. typeof(IChatService),
     17. binding,
     18. String.Format("net.tcp://{0}/ChatService", externalEndPoint.IPEndpoint));
     19. **// define an internal endpoint for inter-role traffic**
     20. **RoleInstanceEndpoint internalEndPoint =**
     21. **RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["NotificationService"];**
     22. **this.serviceHost.AddServiceEndpoint(**
     23. **typeof(IClientNotification),**
     24. **binding,**
     25. **String.Format("net.tcp://{0}/NotificationService", internalEndPoint.IPEndpoint));**
     26. **// create channel factory for inter-role communication**
     27. **WorkerRole.factory = new ChannelFactory<IClientNotification>(binding);**
     28. try
     29. {
     30. this.serviceHost.Open();
     31. Trace.TraceInformation("Chat service host started successfully.");
     32. ...
     33. }
     34. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 InternalEndpoint - VB*)
     35. Visual Basic
     36. ''' <summary>
     37. ''' Starts the service host object for the internal
     38. ''' and external endpoints of the chat service.
     39. ''' </summary>
     40. ''' <param name="retries">Specifies the number of retries to
     41. ''' start the service in case of failure.</param>
     42. Private Sub StartChatService(ByVal retries As Integer)
     43. ...
     44. ' define an external endpoint for client traffic
     45. Dim externalEndPoint As RoleInstanceEndpoint = RoleEnvironment.CurrentRoleInstance.InstanceEndpoints("ChatService")
     46. Me.serviceHost.AddServiceEndpoint(GetType(IChatService), binding, String.Format("net.tcp://{0}/ChatService", externalEndPoint.IPEndpoint))
     47. **' define an internal endpoint for inter-role traffic**
     48. **Dim internalEndPoint As RoleInstanceEndpoint = RoleEnvironment.CurrentRoleInstance.InstanceEndpoints("NotificationService")**
     49. **Me.serviceHost.AddServiceEndpoint(GetType(IClientNotification), binding, String.Format("net.tcp://{0}/NotificationService", internalEndPoint.IPEndpoint))**
     50. **' create channel factory for inter-role communication**
     51. **WorkerRole.factory = New ChannelFactory(Of IClientNotification)(binding)**
     52. Try
     53. Me.serviceHost.Open()
     54. Trace.TraceInformation("Chat service host started successfully.")
     55. ...
     56. End Sub
  9. Add a new **NotifyAllNodes** method to the **WorkerRole** class. The worker role uses this method to inform other roles whenever a client starts or ends a session in this role.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 NotifyAllNodes - CS*)
     2. C#
     3. **/// <summary>**
     4. **/// Notifies all available worker roles to update their active sessions list**
     5. **/// when a new client connects or disconnects.**
     6. **/// </summary>**
     7. **/// <param name="session">The SessionInformation object for the client.</param>**
     8. **internal static void NotifyAllNodes(SessionInformation session)**
     9. **{**
     10. **// iterate over all instances of the internal endpoint except the current role - no need to notify itself**
     11. **var current = RoleEnvironment.CurrentRoleInstance;**
     12. **var endPoints = current.Role.Instances**
     13. **.Where(instance => instance != current)**
     14. **.Select(instance => instance.InstanceEndpoints["NotificationService"]);**
     15. **foreach (var ep in endPoints)**
     16. **{**
     17. **EndpointAddress address =**
     18. **new EndpointAddress(String.Format("net.tcp://{0}/NotificationService", ep.IPEndpoint));**
     19. **IClientNotification client = WorkerRole.factory.CreateChannel(address);**
     20. **try**
     21. **{**
     22. **client.UpdateClientList(session);**
     23. **((ICommunicationObject)client).Close();**
     24. **}**
     25. **catch (TimeoutException timeoutException)**
     26. **{**
     27. **Trace.TraceError("Unable to notify worker role instance '{0}'. The service operation timed out. {1}", ep.RoleInstance.Id, timeoutException.Message);**
     28. **((ICommunicationObject)client).Abort();**
     29. **}**
     30. **catch (CommunicationException communicationException)**
     31. **{**
     32. **Trace.TraceError("Unable to notify worker role instance '{0}'. There was a communication problem. {1} - {2}", ep.RoleInstance.Id, communicationException.Message, communicationException.StackTrace);**
     33. **((ICommunicationObject)client).Abort();**
     34. **}**
     35. **}**
     36. **}**
     37. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 NotifyAllNodes - VB*)
     38. Visual Basic
     39. **''' <summary>**
     40. **''' Notifies all available worker roles to update their active sessions list**
     41. **''' when a new client connects or disconnects.**
     42. **''' </summary>**
     43. **''' <param name="session">The SessionInformation object for the client.</param>**
     44. **Friend Shared Sub NotifyAllNodes(ByVal session As SessionInformation)**
     45. **' iterate over all instances of the internal endpoint except the current role - no need to notify itself**
     46. **Dim current = RoleEnvironment.CurrentRoleInstance**
     47. **Dim endPoints = current.Role.Instances.Where(Function(instance) instance IsNot current).Select(Function(instance) instance.InstanceEndpoints("NotificationService"))**
     48. **For Each ep In endPoints**
     49. **Dim address As New EndpointAddress(String.Format("net.tcp://{0}/NotificationService", ep.IPEndpoint))**
     50. **Dim client As IClientNotification = WorkerRole.factory.CreateChannel(address)**
     51. **Try**
     52. **client.UpdateClientList(session)**
     53. **CType(client, ICommunicationObject).Close()**
     54. **Catch timeoutException As TimeoutException**
     55. **Trace.TraceError("Unable to notify worker role instance '{0}'. The service operation timed out. {1}", ep.RoleInstance.Id, timeoutException.Message)**
     56. **CType(client, ICommunicationObject).Abort()**
     57. **Catch communicationException As CommunicationException**
     58. **Trace.TraceError("Unable to notify worker role instance '{0}'. There was a communication problem. {1} - {2}", ep.RoleInstance.Id, communicationException.Message, communicationException.StackTrace)**
     59. **CType(client, ICommunicationObject).Abort()**
     60. **End Try**
     61. **Next ep**
     62. **End Sub**
     63. **Note:** Internally, worker roles expose an endpoint that uses the **IClientNotification** contract. This is the same contract used by worker roles to communicate back with clients. **NotifyAllNodes** iterates over every instance of the worker role, except the current instance, since the worker role does not need to notify itself, retrieves its internal endpoint, and invokes the **UpdateClientList** operation to alert each worker role about session activity in the current instance.
  10. Next, add a new **ForwardMessage** method to the **WorkerRole** class. This method uses the internal channel to forward messages from clients in the current role to the role where the target session is active.
      1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 ForwardMessage - CS*)
      2. C#
      3. **/// <summary>**
      4. **/// Forwards a message from the current role to the role of the destination session.**
      5. **/// </summary>**
      6. **/// <param name="message">The message to forward.</param>**
      7. **/// <param name="fromSessionId">The ID of the source session.</param>**
      8. **/// <param name="toSessionId">The ID of the target session.</param>**
      9. **public static void ForwardMessage(string message, string fromSessionId, string toSessionId)**
      10. **{**
      11. **SessionInformation session = SessionManager.GetSession(toSessionId);**
      12. **if (session == null)**
      13. **{**
      14. **return;**
      15. **}**
      16. **// retrieve the endpoint for the role instance where the target session is active**
      17. **var targetRole = RoleEnvironment.CurrentRoleInstance.Role.Instances**
      18. **.Where(role => role.Id == session.RoleId).FirstOrDefault();**
      19. **if (targetRole != null)**
      20. **{**
      21. **var ep = targetRole.InstanceEndpoints["NotificationService"];**
      22. **if (ep != null)**
      23. **{**
      24. **EndpointAddress address =**
      25. **new EndpointAddress(String.Format("net.tcp://{0}/NotificationService", ep.IPEndpoint));**
      26. **IClientNotification client = WorkerRole.factory.CreateChannel(address);**
      27. **try**
      28. **{**
      29. **client.DeliverMessage(message, fromSessionId, toSessionId);**
      30. **((ICommunicationObject)client).Close();**
      31. **}**
      32. **catch (TimeoutException timeoutException)**
      33. **{**
      34. **Trace.TraceError("Unable to forward message to instance '{0}'. The service operation timed out. {1}", ep.RoleInstance.Id, timeoutException.Message);**
      35. **((ICommunicationObject)client).Abort();**
      36. **}**
      37. **catch (CommunicationException communicationException)**
      38. **{**
      39. **Trace.TraceError("Unable to forward message to instance '{0}'. There was a communication problem. {1} - {2}", ep.RoleInstance.Id, communicationException.Message, communicationException.StackTrace);**
      40. **((ICommunicationObject)client).Abort();**
      41. **}**
      42. **}**
      43. **}**
      44. **}**
      45. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 ForwardMessage - VB*)
      46. Visual Basic
      47. **' <summary>**
      48. **' Forwards a message from the current role to the role of the destination session.**
      49. **' </summary>**
      50. **' <param name="message">The message to forward.</param>**
      51. **' <param name="fromSessionId">The ID of the source session.</param>**
      52. **' <param name="toSessionId">The ID of the target session.</param>**
      53. **Public Shared Sub ForwardMessage(ByVal message As String, ByVal fromSessionId As String, ByVal toSessionId As String)**
      54. **Dim session As SessionInformation = SessionManager.GetSession(toSessionId)**
      55. **If session Is Nothing Then**
      56. **Return**
      57. **End If**
      58. **' retrieve the endpoint for the role instance where the target session is active**
      59. **Dim targetRole = RoleEnvironment.CurrentRoleInstance.Role.Instances.Where(Function(role) role.Id = session.RoleId).FirstOrDefault()**
      60. **If targetRole IsNot Nothing Then**
      61. **Dim ep = targetRole.InstanceEndpoints("NotificationService")**
      62. **If ep IsNot Nothing Then**
      63. **Dim address As New EndpointAddress(String.Format("net.tcp://{0}/NotificationService", ep.IPEndpoint))**
      64. **Dim client As IClientNotification = WorkerRole.factory.CreateChannel(address)**
      65. **Try**
      66. **client.DeliverMessage(message, fromSessionId, toSessionId)**
      67. **CType(client, ICommunicationObject).Close()**
      68. **Catch timeoutException As TimeoutException**
      69. **Trace.TraceError("Unable to forward message to instance '{0}'. The service operation timed out. {1}", ep.RoleInstance.Id, timeoutException.Message)**
      70. **CType(client, ICommunicationObject).Abort()**
      71. **Catch communicationException As CommunicationException**
      72. **Trace.TraceError("Unable to forward message to instance '{0}'. There was a communication problem. {1} - {2}", ep.RoleInstance.Id, communicationException.Message, communicationException.StackTrace)**
      73. **CType(client, ICommunicationObject).Abort()**
      74. **End Try**
      75. **End If**
      76. **End If**
      77. **End Sub**
      78. **Note:** The **ForwardMessage** method retrieves the target session information to determine its role ID, obtains a reference to the corresponding worker role and then retrieves its internal endpoint. Finally, it creates a new channel instance and calls the **DeliverMessage** operation to forward the message to the target worker role over the internal endpoint.

Task 2 – Receiving Notifications from Other Worker Roles

* 1. In order to allow communication between clients connected to different instances of the service, worker roles must act as proxies for remote clients. Clients send messages and notifications to their role, which in turn forwards them to the remote worker role using the same contract that the service uses to communicate back with clients.
  2. In this task, you extend the chat service class to implement the client notification contract.
  3. Open the **ChatService.cs** file (for Visual C# projects) or **ChatService.vb** file (for Visual Basic projects) in the **AzureTalk.Service** project.
  4. Add the **IClientNotification** interface to the list of contracts implemented by the **ChatService** class.
     1. C#
     2. public class ChatService : IChatService**, IClientNotification**
     3. {
     4. ...
     5. Visual Basic
     6. Public Class ChatService
     7. Implements IChatService**, IClientNotification**
     8. ...
  5. Add the **UpdateClientList** method to the **ChatService** class. Worker role instances use this operation to notify their peers whenever a client starts or ends a session.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 UpdateClientList - CS*)
     2. C#
     3. **/// <summary>**
     4. **/// Receives notifications when a new client connects or disconnects in another worker role.**
     5. **/// </summary>**
     6. **/// <param name="clientInfo">The ClientInformation object for the client.</param>**
     7. **public void UpdateClientList(ClientInformation clientInfo)**
     8. **{**
     9. **if (clientInfo.IsActive)**
     10. **{**
     11. **SessionInformation session;**
     12. **if (SessionManager.CreateOrUpdateSession(clientInfo.SessionId, clientInfo.UserName, clientInfo.RoleId, null, out session))**
     13. **{**
     14. **Trace.TraceInformation("Remote session '{0}' by user '{1}' has been opened in role '{2}'.", session.SessionId, session.UserName, session.RoleId);**
     15. **}**
     16. **}**
     17. **else**
     18. **{**
     19. **SessionManager.RemoveSession(clientInfo.SessionId);**
     20. **Trace.TraceInformation("Remote session '{0}' by user '{1}' has been closed in role '{2}'.", clientInfo.SessionId, clientInfo.UserName, clientInfo.RoleId);**
     21. **}**
     22. **NotifyConnectedClients(clientInfo);**
     23. **}**
     24. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 UpdateClientList - VB*)
     25. Visual Basic
     26. **' <summary>**
     27. **' Receives notifications when a new client connects or disconnects in another worker role.**
     28. **' </summary>**
     29. **' <param name="clientInfo">The ClientInformation object for the client.</param>**
     30. **Public Sub UpdateClientList(ByVal clientInfo As ClientInformation) Implements IClientNotification.UpdateClientList**
     31. **If clientInfo.IsActive Then**
     32. **Dim session As SessionInformation**
     33. **If SessionManager.CreateOrUpdateSession(clientInfo.SessionId, clientInfo.UserName, clientInfo.RoleId, Nothing, session) Then**
     34. **Trace.TraceInformation("Remote session '{0}' by user '{1}' has been opened in role '{2}'.", session.SessionId, session.UserName, session.RoleId)**
     35. **End If**
     36. **Else**
     37. **SessionManager.RemoveSession(clientInfo.SessionId)**
     38. **Trace.TraceInformation("Remote session '{0}' by user '{1}' has been closed in role '{2}'.", clientInfo.SessionId, clientInfo.UserName, clientInfo.RoleId)**
     39. **End If**
     40. **NotifyConnectedClients(clientInfo)**
     41. **End Sub**
     42. **Note:** The **UpdateClientList** method determines whether a session is active and then registers it with the Session Manager. If a session has ended, it calls the Session Manager to remove it. In both cases, it calls **NotifyConnectedClients** to inform every client connected to the worker role about the session activity.
  6. At this point, you may wish to review the **DeliverMessage** method, which you implemented in the previous exercise to send messages to the client. Here, you take advantage of this method to implement the **DeliverMessage** operation of the **IClientNotification** contract. No additional code is required to fulfill the contract for a Visual C# project. For a Visual Basic project, update the method signature to indicate that the **DeliverMessage** method implements the **DeliverMessage** operation of the **IClientNotification** contract explicitly.
     1. Visual Basic
     2. ''' <summary>
     3. ''' Delivers a message to a client in the current worker role.
     4. ''' </summary>
     5. ''' <param name="message">The message to forward.</param>
     6. ''' <param name="fromSessionId">The session ID of the message originator.</param>
     7. ''' <param name="toSessionId">The session ID of the message recipient.</param>
     8. Public Sub DeliverMessage(ByVal message As String, ByVal fromSessionId As String, ByVal toSessionId As String) **Implements IClientNotification.DeliverMessage**
     9. ' Method Implementation
     10. End Sub

Task 3 – Sending Notifications to Other Worker Roles

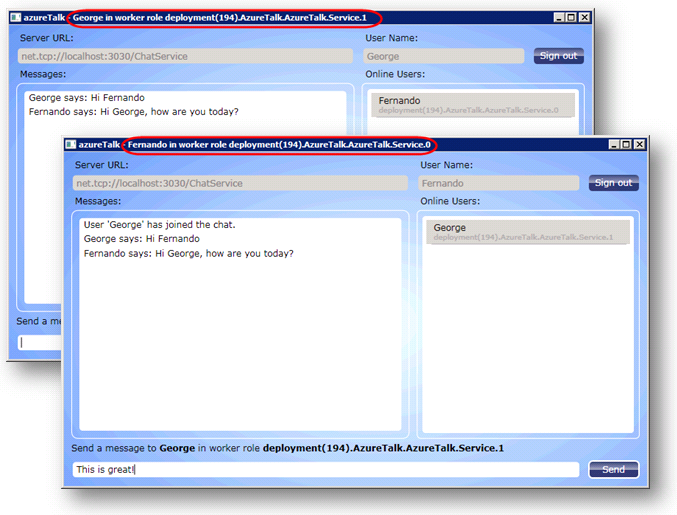
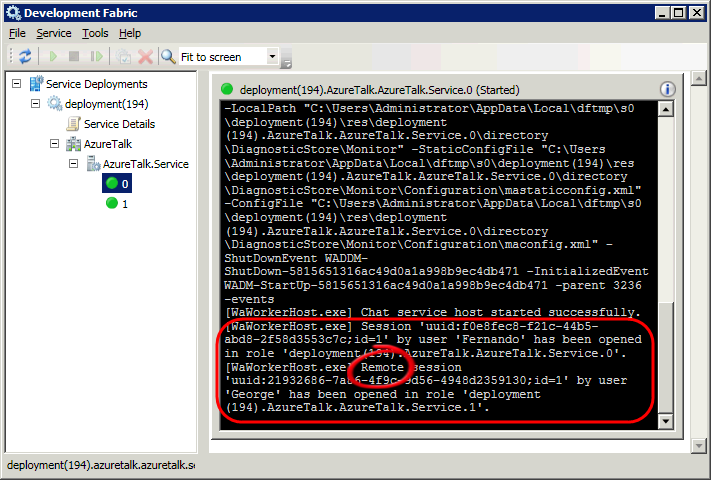
* 1. In this task, you update the service to send notifications to other worker roles whenever a new client connects or disconnects and when clients in the current role send messages to peers in different roles.
  2. Open the **ChatService.cs** file (for Visual C# projects) or **ChatService.vb** (for Visual Basic projects) in the **AzureTalk.Service** project.
  3. If you implement the code in C#, locate the **Register** method and insert a call to **NotifyAllNodes** inside the embedded handler for the **Closed** event of the channel and immediately following the call to **NotifyConnectedClients**. In addition, insert a second call to **NotifyAllNodes** after the call to **NotifyConnectedClients** in the main body of the method. This ensures that worker roles are alerted both when clients open and when they close a session.
     1. C#
     2. public ClientInformation Register(string userName)
     3. {
     4. // retrieve session information
     5. string roleId = RoleEnvironment.CurrentRoleInstance.Id;
     6. string sessionId = OperationContext.Current.SessionId;
     7. IClientNotification callback = OperationContext.Current.GetCallbackChannel<IClientNotification>();
     8. SessionInformation session;
     9. if (SessionManager.CreateOrUpdateSession(sessionId, userName, roleId, callback, out session))
     10. {
     11. // ensure that the session is killed when channel is closed
     12. OperationContext.Current.Channel.Closed += (sender, e) =>
     13. {
     14. SessionManager.RemoveSession(sessionId);
     15. NotifyConnectedClients(session);
     16. **WorkerRole.NotifyAllNodes(session);**
     17. Trace.TraceInformation("Session '{0}' by user '{1}' has been closed in role '{2}'.", sessionId, userName, roleId);
     18. };
     19. Trace.TraceInformation("Session '{0}' by user '{1}' has been opened in role '{2}'.", sessionId, userName, roleId);
     20. }
     21. // Notify clients connected to this role
     22. NotifyConnectedClients(session);
     23. **// Notify other worker roles**
     24. **WorkerRole.NotifyAllNodes(session);**
     25. return new ClientInformation()
     26. {
     27. SessionId = sessionId,
     28. UserName = userName,
     29. RoleId = roleId
     30. };
     31. }
  4. If you implement the code in Visual Basic, locate the **Register** method and insert a call to **NotifyAllNodes** immediately after the call to **NotifyConnectedClients**. In addition, locate the **OnClosed** event handler that handles the **Closed** event of the channel and insert a call to **NotifyAllNodes** after the call to **NotifyConnectedClients**. This ensures that worker roles are alerted both when clients open and when they close a session.
     1. Visual Basic
     2. ' <returns>The ClientInformation object for the new session.</returns>
     3. Public Function Register(ByVal userName As String) As ClientInformation Implements IChatService.Register
     4. ' retrieve session information
     5. Dim roleId As String = RoleEnvironment.CurrentRoleInstance.Id
     6. Dim sessionId As String = OperationContext.Current.SessionId
     7. Dim callback As IClientNotification = OperationContext.Current.GetCallbackChannel(Of IClientNotification)()
     8. Dim session As SessionInformation
     9. If SessionManager.CreateOrUpdateSession(sessionId, userName, roleId, callback, session) Then
     10. ' ensure that the session is killed when channel is closed
     11. AddHandler OperationContext.Current.Channel.Closed, Function(sender, e) OnClosed(sender, e, sessionId, session, userName, roleId)
     12. Trace.TraceInformation("Session '{0}' by user '{1}' has been opened in role '{2}'.", sessionId, userName, roleId)
     13. End If
     14. ' Notify clients connected to this role
     15. NotifyConnectedClients(session)
     16. **' Notify other worker roles**
     17. **WorkerRole.NotifyAllNodes(session)**
     18. Return New ClientInformation() With {.SessionId = sessionId, .UserName = userName, .RoleId = roleId}
     19. End Function
     20. Private Function OnClosed(ByVal sender As Object, ByVal e As Object, ByVal sessionId As String, ByVal session As SessionInformation, ByVal userName As String, ByVal roleId As String) As Object
     21. SessionManager.RemoveSession(sessionId)
     22. NotifyConnectedClients(session)
     23. **WorkerRole.NotifyAllNodes(session)**
     24. Trace.TraceInformation("Session '{0}' by user '{1}' has been closed in role '{2}'.", sessionId, userName, roleId)
     25. Return Nothing
     26. End Function
  5. Next, replace the body of the **SendMessage** method with the code shown (highlighted) below. The changed code determines whether a client is local or remote and if necessary, forwards the message to the target worker role.
     1. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 Multi-Role SendMessage - CS*)
     2. C#
     3. public void SendMessage(string message, string sessionId)
     4. {
     5. **string fromSessionId = OperationContext.Current.SessionId;**
     6. **SessionInformation toSession = SessionManager.GetSession(sessionId);**
     7. **// if recipient is connected to this role, deliver the message to the**
     8. **// recipient; otherwise, forward the message to the recipient's role**
     9. **if (toSession != null)**
     10. **{**
     11. **if (toSession.RoleId == RoleEnvironment.CurrentRoleInstance.Id)**
     12. **{**
     13. **this.DeliverMessage(message, fromSessionId, sessionId);**
     14. **}**
     15. **else**
     16. **{**
     17. **WorkerRole.ForwardMessage(message, fromSessionId, sessionId);**
     18. **}**
     19. **}**
     20. }
     21. (Code Snippet – *Windows Azure Worker Role Communication* – *Ex02 Multi-Role SendMessage - VB*)
     22. Visual Basic
     23. Public Sub SendMessage(ByVal message As String, ByVal sessionId As String) Implements IChatService.SendMessage
     24. **Dim fromSessionId As String = OperationContext.Current.SessionId**
     25. **Dim toSession As SessionInformation = SessionManager.GetSession(sessionId)**
     26. **' if recipient is connected to this role, deliver the message to the**
     27. **' recipient; otherwise, forward the message to the recipient's role**
     28. **If toSession IsNot Nothing Then**
     29. **If toSession.RoleId = RoleEnvironment.CurrentRoleInstance.Id Then**
     30. **Me.DeliverMessage(message, fromSessionId, sessionId)**
     31. **Else**
     32. **WorkerRole.ForwardMessage(message, fromSessionId, sessionId)**
     33. **End If**
     34. **End If**

End Sub

* + 1. **Note:** The **SendMessage** operation retrieves the target session, obtains its role ID and compares it with the ID of the current worker role. If the IDs match, the message is sent directly to the client using **DeliverMessage**; otherwise, the message is forwarded to the target role using **WorkerRole.ForwardMessage**.

Verification

You will now test the updated solution using two instances of the worker role. To determine whether clients connected to different worker roles are able to communicate with each other, you will start two instances of the client application and connect each one to a different role. Finally, you will exchange messages between these two instances to establish that messages and notifications flow between worker roles.

* 1. Press **F5** to launch the cloud project in the development fabric.
  2. Switch to the development fabric UI and ensure that the service has started successfully. Notice that two instances of the worker role are currently active.
  3. In **Solution Explorer**, right-click the **AzureTalk.Client** project, point to **Debug** and select **Start new instance**.
  4. In the main window of the application, enter a user name for the first client and click **Sign In**.
  5. Start a second instance of the client application and sign in using a different user name. Ensure that the second client connects to a different worker role than the first client. You can verify the connected role by checking the title bar of the application that shows the name of the current user and the ID of the worker role where the client is connected. Each client should connect to a different worker role.
     1. **Note:** The load balancer determines which instance of the worker role responds to a client's request. The development fabric typically assigns connections in round robin fashion, so the second client should normally start a session in a different worker role when it connects. If necessary, restart one of the client instances until both clients have sessions in different worker roles.
     2. 
     3. Figure
     4. Verifying the connected worker role ID of the client
  6. In the second client, notice how the online users list includes the first client. Switch to the first instance of the client application to make sure that the service notified it about the new active user. This shows that notifications flow from the role where the second user connected to the first role and then back to the client application. Contrast this with the result obtained after you completed the first exercise, where clients in different roles were unaware of each other.
  7. Switch to the development fabric UI and examine the logs for each of the worker roles. View the entries for the sessions that just started and see how the log registers local and remote sessions.
     1. 
     2. Figure
     3. Local and remote sessions logged in the worker role log
  8. In the first instance of the client application, select the other active user from the list, type a message and click **Send**. Switch to the second instance of the application to verify that the service delivered the message.
  9. In the second instance, type a reply and click **Send** once again. Notice that the service delivers messages across worker roles in both directions.
  10. Finally, click **Sign out** in one of the instances of the client application. Switch to the other instance and notice that the server immediately notifies the client attached to the role about the session ending in the other role.

Summary

* 1. By completing this hands-on lab, you have learnt how to communicate externally from worker roles using non-HTTP application protocols. You have seen how to configure external endpoints and how to host a WCF service at these endpoints.
  2. Finally, you have explored internal endpoints and seen how they provide an alternative to queues for inter-role communication.