Introducción a Azure Service Bus

* 1. 
  2. <http://www.dnmplus.net>
  3. Quique Martínez ([Pasiona](http://www.pasiona.com/))

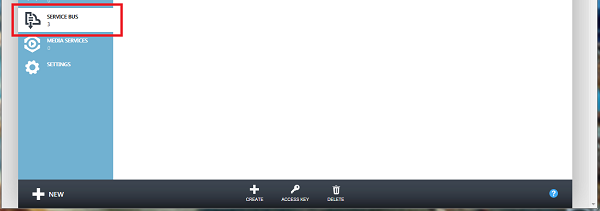
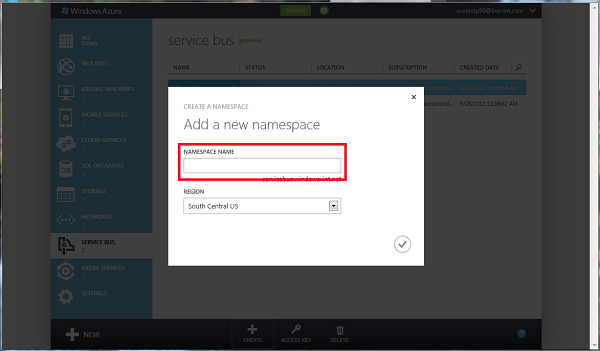
En este artículo presentamos Azure Service Bus, otro de los servicios que nos proporciona la plataforma de *Cloud Computing* de Microsoft.

Azure Service Bus (SB) proporciona una infraestructura en modo SaaS, protegida y ampliamente disponible para una comunicación extendida, una distribución de eventos a gran escala, asignación de nombres y publicación de servicios. SB nos proporciona opciones de conectividad entre los diferentes sistemas/servicios de una arquitectura distribuida, así como capacidades de mensajería tanto *relayed* (retransmitida) como *brokered* (con confianza establecida). Con el patrón de mensajería retransmitida, SB nos permite la mensajería unidireccional directa, mensajería tipo solicitud/respuesta y mensajería punto a punto. Por otra parte, la mensajería con confianza establecida proporciona componentes de mensajería duraderos y asíncronos, tales como colas y suscripciones; es decir, con características que admiten publicación-suscripción y desacoplamiento.

## Configuración de Azure Service Bus

Lo primero que vamos a hacer es configurar el servicio dentro del portal de Azure. Para ello, vamos al portal (**https://manage.windowsazure.com**), y una vez allí seleccionamos la opción de SB (figura 1). Pulsando el botón de crear, nos aparecerá una ventana emergente como la que se muestra en la figura 2. En ella solo tenemos que introducir un nombre y seleccionar en que región queremos que nuestro bus se publique.

Una vez tenemos configurado nuestro SB, en el portal sólo nos queda una cosa que hacer: guardar las dos claves secretas que nos permitirán utilizar el servicio desde nuestras *apps*.

* 1. Figura 1
  2. 
  3. Figura 2
  4. 

## Programando un ejemplo

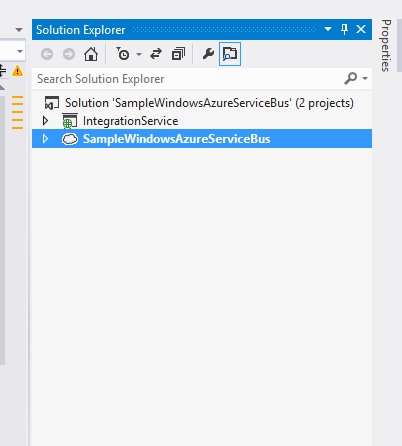
Con el servicio configurado, vamos ahora a realizar una pequeña prueba de concepto, que nos ayudará a entender cómo funciona SB y cómo tenemos que utilizarlo.

Para nuestro ejemplo, imaginemos que estamos desarrollando una capa de servicios que integra muchos desarrollos que hemos ido realizando con nuestro ERP. Tras crear un servicio WCF que expone una serie de métodos que integran nuestras aplicaciones con el ERP, nos hemos dado cuenta de que el sistema tarda mucho en responder cuando llega el final del mes y los *n* usuarios lanzan los procesos de integración. Además, en algún caso hemos perdido información, ya que los procesos de integración pueden hacerse de forma masiva.

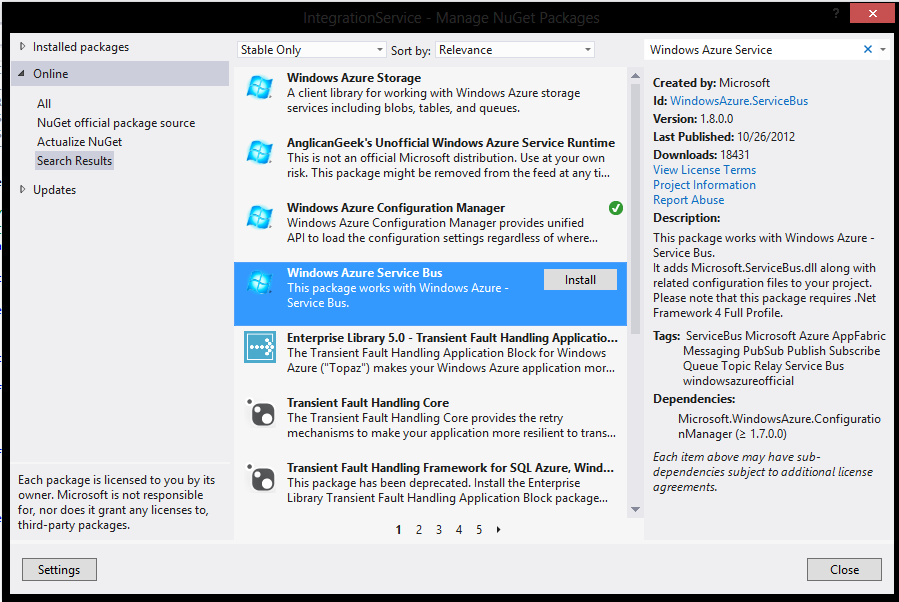
Los cambios que vamos a realizar son para no tener que procesar las nóminas. En lugar de procesarlas directamente, vamos a encolar las peticiones y hacer que un Walker Role las vaya procesando en otro servidor.

## El servicio WCF

La estructura de la solución se muestra en la figura 3, y un fragmento del código del servicio WCF que vamos a utilizar para nuestro ejemplo puede verse en el listado 1 (el código completo puede descargarse desde el sitio web de **dNM**). Como podéis ver, ya tenemos nuestro servicio WCF desarrollado para Azure, y ahora solo tenemos que mejorar su “arquitectura”.

* 1. Figura 3
  2. 
  3. C# Listado 1
  4. public string ContabilizarNominas(IEnumerable<Nomina> nominaList)
  5. {
  6. // Lógica...
  8. return "Se ha procesado todo!";
  9. }

Lo primero que vamos a hacer es instalar el paquete de SB desde NuGet (figura 4). Una vez hecho esto, podremos ver que se nos ha agregado una nueva opción de configuración a los ficheros de configuración del servicio de Azure (listado 2). Aquí es donde deberemos poner en uso las dos claves secretas que hemos comentado al principio.

* 1. Listado 2
  2. <Setting name="Microsoft.ServiceBus.ConnectionString"
  3. value="Endpoint=sb://[your namespace].servicebus.windows.net;
  4. SharedSecretIssuer=owner;SharedSecretValue=[your secret]" />
  5. Figura 4
  6. 

## Añadiendo los datos al Service Bus

El listado 3 muestra los cambios necesarios a nivel de código para hacer uso de SB. Como podemos ver, los cambios en nuestro método ContabilizarNominas han sido pocos. Lo único que hemos hecho ha sido coger todas las nóminas a contabilizar y añadirlas a nuestra cola de SB. Además, hemos ido trazando por donde pasamos para poder tener un poco más de información en caso de error (este código se puede optimizar, se ha dejado así para el ejemplo☺).

* 1. C# Listado 3
  2. public class IntegrationService : IServiceIntegration
  3. {
  4. const string NominaQueueName = "NominasQueueName";
  5. public string ContabilizarNominas(IEnumerable<Nomina> nominaList)
  6. {
  7. // Trazamos que vamos a guardar en cola las nóminas
  8. System.Diagnostics.Trace.TraceInformation("Added nominaList to IntegrationService");
  9. // Creamos la conexión en base los datos de configuración
  10. var connectionString =
  11. CloudConfigurationManager.GetSetting("Microsoft.ServiceBus.ConnectionString");
  12. var namespaceManager = NamespaceManager.CreateFromConnectionString(connectionString);
  13. // Comprobamos si existe la cola y si no, la creamos
  14. if (!namespaceManager.QueueExists(NominaQueueName))
  15. {
  16. namespaceManager.CreateQueue(NominaQueueName);
  17. }
  18. var client = QueueClient.CreateFromConnectionString(connectionString, NominaQueueName);
  19. // Creamos un BrokeredMessage por cada elemento
  20. foreach (var nomina in nominaList)
  21. {
  22. client.Send(new BrokeredMessage(nomina));
  23. System.Diagnostics.Trace.TraceInformation(string.Format("{0}{1}",
  24. "Queued message to process ", Nomina.NIF));
  25. }
  27. return "Las nóminas se han añadido a la cola...";
  28. }
  29. }

## Procesando los datos del Service Bus

Ahora vamos a procesar los datos que hemos ido añadiendo a nuestro SB. Para ello, lo primero que vamos a hacer es agregar un nuevo proyecto a la solución. En esta ocasión es un Worker Role de Azure, pero uno especial (figura 5). Se trata de un **Worker Role with Service Bus Queue**, y al agregarlo podremos ver que nos ofrece parte de la lógica en la gestión de las colas. Por defecto, el Worker Role with Service Bus Queue nos da los tres métodos que se presentan en el listado 4.

Los cambios requeridos para poder procesar los mensajes que hemos añadido desde nuestro servicio WCF pueden verse en el listado 5. Además de cambiar el nombre de la cola, que tiene que ser el mismo que le hemos dado en nuestro servicio WCF, los cambios se concentran en el método Run, y tienen lugar una vez recibimos el mensaje, donde lo recogemos y lo procesamos con la lógica que ya teníamos programada anteriormente:

* 1. Recogemos el mensaje:

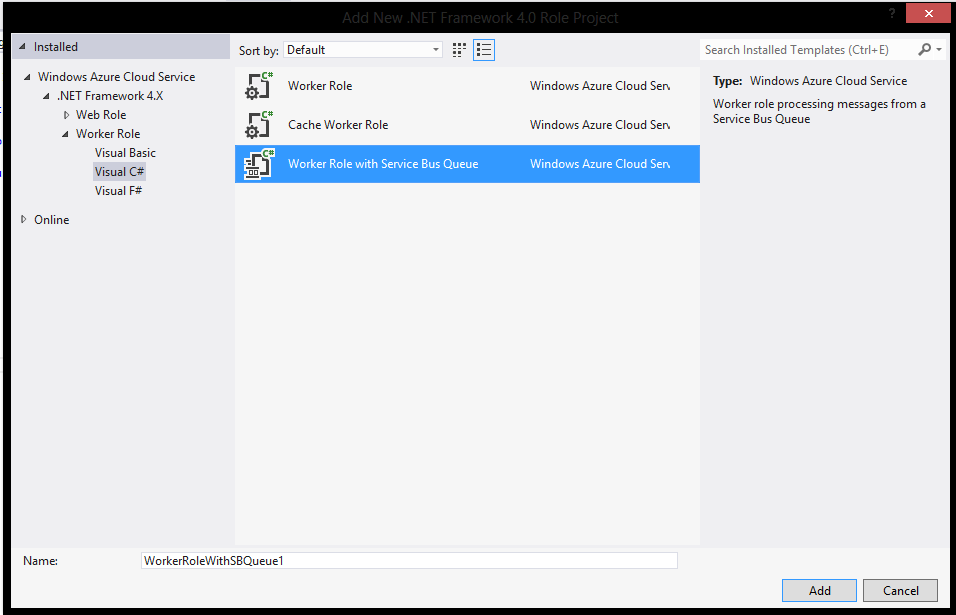
BrokeredMessage receivedMessage = Client.Receive();

* 1. En caso de no ser nulo, recuperamos el cuerpo del mensaje:

var message = receivedMessage.GetBody<Nomina>();

* 1. Procesamos el mensaje:

var result = ContabilizarNomina(message);

* 1. También deberemos añadir el código de gestión de errores que estimemos necesario.
  2. Figura 5
  3. 
  4. C# Listado 4
  5. public class WorkerRole : RoleEntryPoint
  6. {
  7. // The name of your queue
  8. const string QueueName = "ProcessingQueue";
  9. // QueueClient is thread-safe. Recommended that you cache
  10. // rather than recreating it on every request
  11. QueueClient Client;
  12. bool IsStopped;
  13. public override void Run()
  14. {
  15. while (!IsStopped)
  16. {
  17. try
  18. {
  19. // Receive the message
  20. BrokeredMessage receivedMessage = null;
  21. receivedMessage = Client.Receive();
  22. if (receivedMessage != null)
  23. {
  24. // Process the message
  25. Trace.WriteLine("Processing", receivedMessage.SequenceNumber.ToString());
  26. receivedMessage.Complete();
  27. }
  28. }
  29. catch (MessagingException e)
  30. {
  31. if (!e.IsTransient)
  32. {
  33. Trace.WriteLine(e.Message);
  34. throw;
  35. }
  36. Thread.Sleep(10000);
  37. }
  38. catch (OperationCanceledException e)
  39. {
  40. if (!IsStopped)
  41. {
  42. Trace.WriteLine(e.Message);
  43. throw;
  44. }
  45. }
  46. }
  47. }
  48. public override bool OnStart()
  49. {
  50. // Set the maximum number of concurrent connections
  51. ServicePointManager.DefaultConnectionLimit = 12;
  52. // Create the queue if it does not exist already
  53. string connectionString =
  54. CloudConfigurationManager.GetSetting("Microsoft.ServiceBus.ConnectionString");
  55. var namespaceManager = NamespaceManager.CreateFromConnectionString(connectionString);
  56. if (!namespaceManager.QueueExists(QueueName))
  57. {
  58. namespaceManager.CreateQueue(QueueName);
  59. }
  60. // Initialize the connection to Service Bus Queue
  61. Client = QueueClient.CreateFromConnectionString(connectionString, QueueName);
  62. IsStopped = false;
  63. return base.OnStart();
  64. }
  65. public override void OnStop()
  66. {
  67. // Close the connection to Service Bus Queue
  68. IsStopped = true;
  69. Client.Close();
  70. base.OnStop();
  71. }
  72. }
  73. C# Listado 5
  74. public class WorkerRole : RoleEntryPoint
  75. {
  76. // The name of your queue
  77. const string QueueName = "ProcessingQueue";
  78. // ...
  79. public override void Run()
  80. {
  81. while (!IsStopped)
  82. {
  83. try
  84. {
  85. // Receive the message
  86. BrokeredMessage receivedMessage = Client.Receive();
  87. if (receivedMessage != null)
  88. {
  89. // Process the message
  90. Trace.WriteLine("Processing", receivedMessage.SequenceNumber.ToString());
  91. var message = receivedMessage.GetBody<Nomina>();
  92. var result = ContabilizarNomina(message);
  93. if (result)
  94. receivedMessage.Complete();
  95. else
  96. {
  97. // Gestión de errores
  98. }
  99. }
  100. }
  101. catch (MessagingException e)
  102. {
  103. if (!e.IsTransient)
  104. {
  105. Trace.WriteLine(e.Message);
  106. throw;
  107. }
  108. Thread.Sleep(10000);
  109. }
  110. catch (OperationCanceledException e)
  111. {
  112. if (!IsStopped)
  113. {
  114. Trace.WriteLine(e.Message);
  115. throw;
  116. }
  117. }
  118. }
  119. }
  120. }

## Conclusiones

Desde mi punto de vista, utilizar Azure Service Bus no presenta ningún aspecto negativo. Todo es positivo: nos ayuda a gestionar los sistemas distribuidos, no perdemos datos y optimiza nuestros sistemas delegando responsabilidades. Todo ello utilizando este servicio sin tener que preocuparnos del hardware, red y demás aspectos que nos ahorramos utilizando SaaS.

## Referencias

[1] Portal de Microsoft Azure: [**http://www.windowsazure.com**](http://www.windowsazure.com)