#03

súbete a la nube de Microsoft

Windows Azure AppFabric

Ibón Landa Martín
Unai Zorrilla Castro

campus MVP .com

Plain Concepts
The Microsoft Technologies Company

Súbete a la nube de Microsoft

Parte 3: Windows AppFabric

Ibón Landa Martín
Unai Zorrilla Castro

SÚBETE A LA NUBE DE MICROSOFT PARTE 3: WINDOWS APPFABRIC

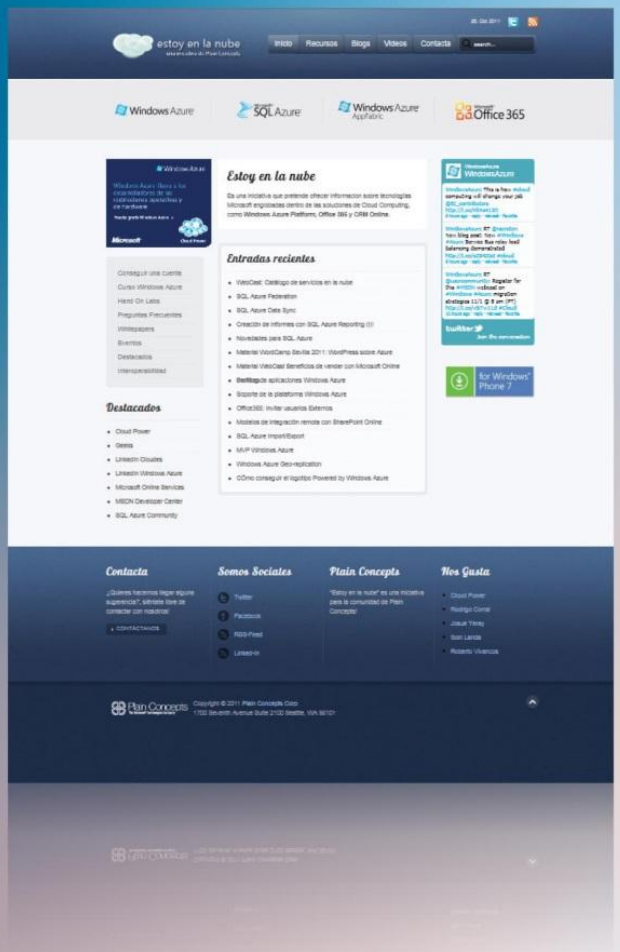
Noviembre de 2011

Contenido

estoy en la nube

INICIATIVA DE PLAIN CONCEPTS

www.estoyenlanube.com



Windows Azure

www.plainconcepts.com

Plain Concepts is a company specialized in Microsoft technologies, agile methodologies, Application Lifecycle Management, performance tuning, advanced debugging, software architecture and User Experience.

Plain Concepts focuses on delivering high quality consulting, mentoring and training as well as in being an effective and reliable team resolving all type of software development issues.

Plain Concepts

¿Aún quieres más?

campus MVP

Formación online especializada en tecnologías Microsoft.

krasis PRESS

Los libros que lo saben todo sobre tecnologías Microsoft.

Síguenos y descubrirás los mejores trucos y recursos:

facebook.com/campusmvp twitter.com/campusmvp

feed your brain®

- Sin tener que desplazarse
- Sin romper el ritmo de trabajo
- Preguntándole a los que más saben

infórmate ya:
902 876 475
www.campusmvp.com

http://www.krasis.com

krasis

Microsoft Partner
Silver Learning
Silver Software Development

CAPÍTULO

I

Windows Azure AppFabric

Windows Azure Platform AppFabric proporciona un bus de servicios empresarial y un servicio de control de acceso que permite integrar servicios y aplicaciones que se ejecutan en la nube, en proveedores de alojamiento tradicionales y en la propia empresa basándose en estándares de interoperabilidad.

1.- ¿QUÉ ES WINDOWS AZURE APPFABRIC?

Windows Azure Platform AppFabric proporciona un bus de servicios empresarial y un servicio de control de acceso que permite integrar servicios y aplicaciones que se ejecutan en la nube, en proveedores de alojamiento tradicionales y en la propia empresa basándose en estándares de interoperabilidad.

1.1.- AppFabric Service Bus

Un bus de servicios empresarial (AppFabric Service Bus) permite orquestar la conectividad segura entre diferentes servicios y aplicaciones a través de cortafuegos y redes utilizando numerosos patrones de comunicación.

Los diferentes servicios se registran en el bus de servicios de manera que pueden ser fácilmente accedidos a través de las más variadas tipologías de red.

Si una aplicación tiene que consumir e interactuar con una gran cantidad de servicios, algunos de ellos controlados por terceros, utilizar un bus de servicios permite "olvidarse" de detalles como la autenticación y autorización, los protocolos de comunicación, los cortafuegos y otras cuestiones técnicas, delegándolos en el bus de servicios. De esta manera, los desarrolladores pueden centrarse en solucionar escenarios de negocio y no perderse en los detalles de implementación de los servicios.

El bus de servicios de la plataforma Azure facilita la labor de conectar aplicaciones que se ejecutan sobre Windows Azure o contra SQL Azure con aplicaciones que corren en una infraestructura propia y contra servidores de bases de datos convencionales.

Otro escenario en el que el bus de servicios ayuda enormemente es en la creación de aplicaciones compuestas mediante la integración de diferentes servicios ya existentes y nuevos servicios que se ejecutan en la plataforma Azure.

A continuación puede verse el esquema de funcionamiento del bus de servicios de la plataforma Azure.

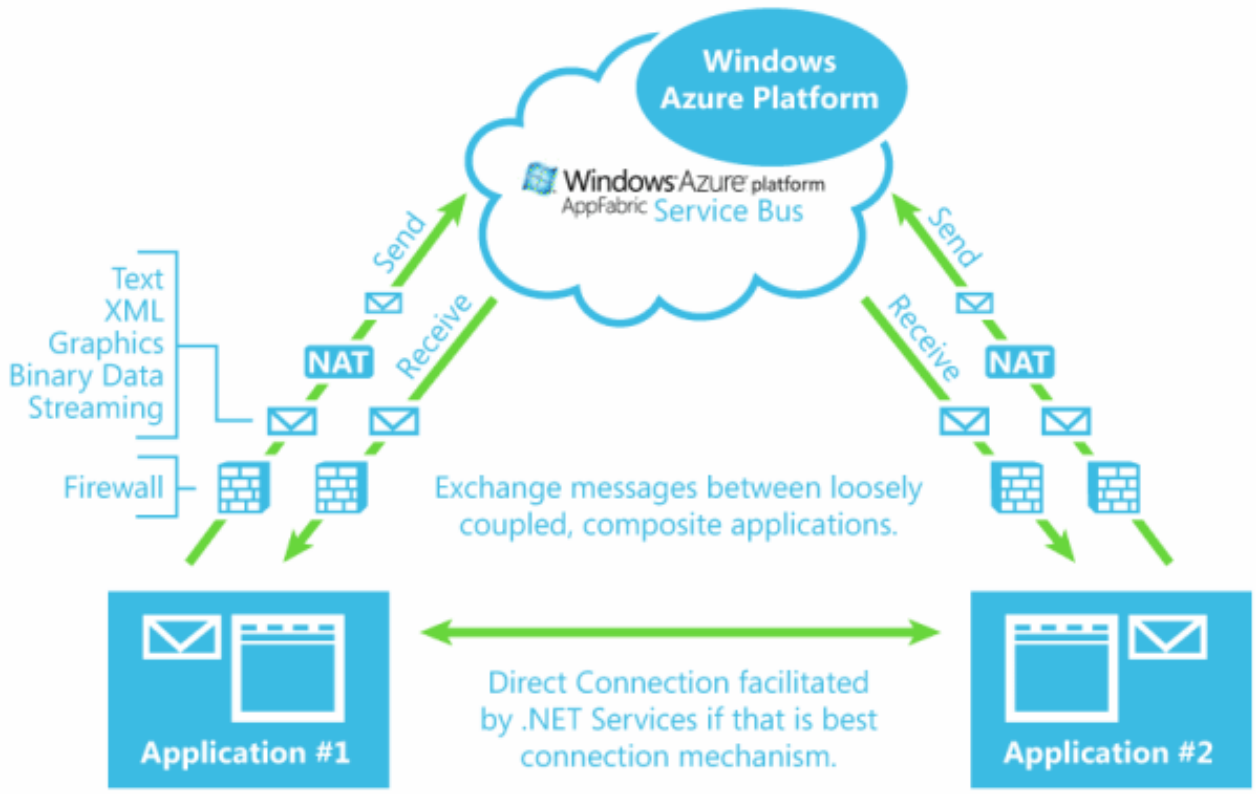


Figura 1.1.- AppFabric Service Bus

Del esquema anterior los puntos más destacables son:

- Que el mecanismo preferido de comunicación entre aplicaciones en la nube es utilizar la infraestructura que el framework de .NET proporciona para tal fin, que es Windows Communication Foundation.
- Que cuando la comunicación directa no es posible por cuestiones relacionada con la topología de la red, o por el hecho de que no se controlen todos los servicios y que se necesite integración con otros servicios de terceros, el bus de servicios de .NET puede ahorrar mucho trabajo proporcionando: mecanismos de comunicación amigables con los cortafuegos, comunicaciones basadas en mensajes entre servicios, direccionamiento entre redes heterogéneas (NAT) y servicios de orquestación.

1.2.- AppFabric Access Control

El servicio de Control de acceso (AppFabric Access Control) permite generar una autorización federada entre aplicaciones y servicios, sin la programación complicada que, por lo general, se requiere para proteger aplicaciones que atraviesan los límites de la organización.

Al admitir un sencillo modelo declarativo de reglas y claims, las reglas del Control de acceso pueden configurarse con facilidad y flexibilidad para cubrir varias necesidades de seguridad y distintas infraestructuras de administración de identidades.

A continuación puede verse el esquema de funcionamiento del Control de acceso de la plataforma Azure.

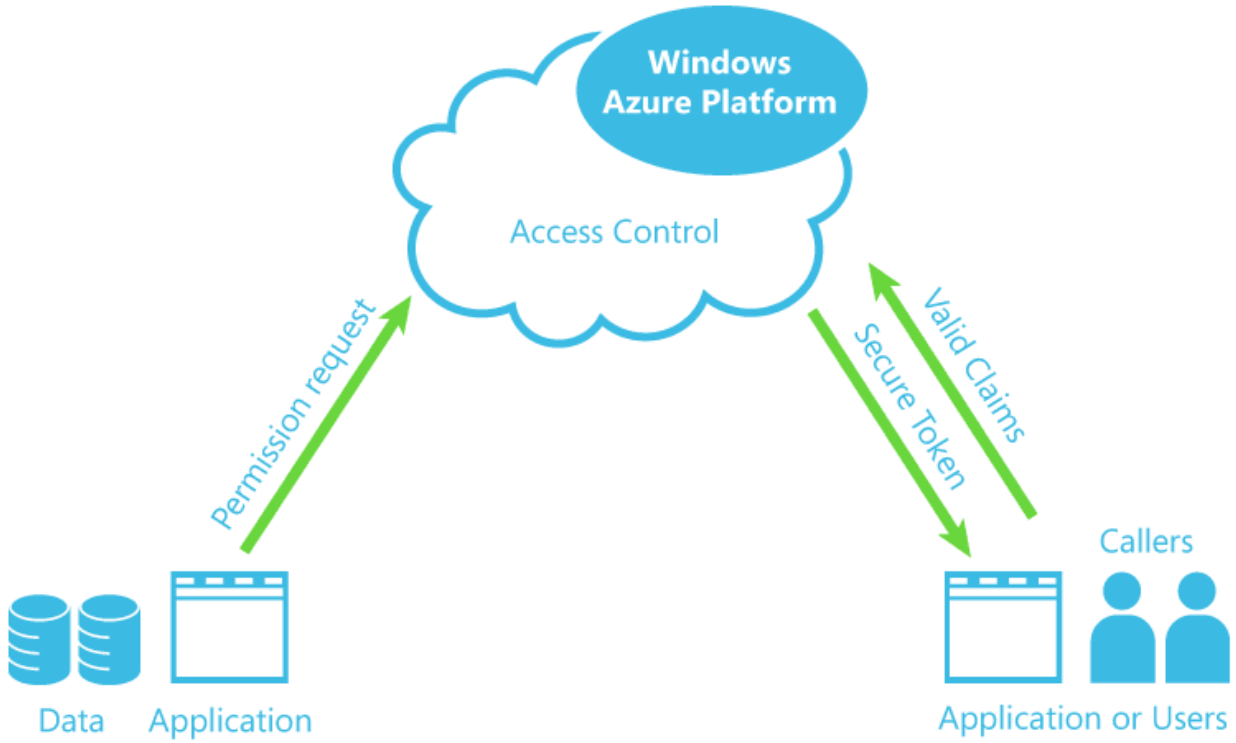


Figura 1.2.- AppFabric Access Control

No se puede decir que este componente de la plataforma Azure sea de uso común en la gran mayoría de las aplicaciones Azure, pero sí que es de gran utilidad en las ocasiones en las que orquestar servicios y comunicarlos entre sí es la principal labor que tiene que realizar la aplicación.

1.3.- AppFabric Cache

Windows Azure AppFabric Caching es un sistema de caché distribuida, en memoria, que se ofrece como un servicio en la nube.

Un servicio similar ya existía para soluciones on-premise, integrado dentro de Windows Server AppFabric. Ahora tenemos las mismas capacidades y características para aplicaciones que estén en la nube. Se ofrece como un servicio en la nube, por lo que como veremos nos vamos a tener que hacer nada relacionado con las tareas de instalación, configuración o administración…simplemente hacer uso del servicio.

2.- AUTENTICACIÓN FEDERADA DE APLICACIONES WEB EN AZURE CON WINDOWS IDENTITY FOUNDATION

La gestión de la identidad sigue siendo unos los problemas habituales con los que se encuentran los desarrolladores a la hora de desarrollar una aplicación; evitar el acceso no autorizado de terceros, controlar la autorización de acceso a los datos de los usuarios autenticados y revisar determinada información o atributos asociados a ellos son siempre muchos de los esfuerzos en los que gastamos nuestras horas.

Los desarrolladores se enfrentan de forma habitual a los mismos problemas; cómo proveer de un sistema de autenticación y autorización a las aplicaciones que desarrollan, qué tecnología elegir, que sistema de almacenamiento emplear etc...

El modelo de seguridad basado en claims es una estrategia de seguridad permite disponer de las funcionalidades habituales de autenticación y autorización en una aplicación, centralizando todo el proceso en servicios externos a la propia aplicación, servicios desarrollados y mantenidos por expertos en seguridad.

Access Control es un servicio ofrecido por la plataforma Windows Azure que provee de la funcionalidad que se acaba de mencionar. En lugar de tener que escribir en cada aplicación que se desarrolle la lógica de

autenticación y autorización, se puede usar Access Control para delegar estas dos funcionalidades en este servicio, pudiendo hacer además, de una forma sencilla la compartición de estos almacenes entre distintas aplicaciones, facilitar el Single Sign On y otras funcionalidades.

A continuación se describen algunas de las principales tecnologías de identidad que puede emplearse junto con Windows Azure para proteger las aplicaciones.

- Windows Identity Foundation
- Active Directory Federation Services 2.0
- Windows Azure AppFabric Access Control Service

2.1.- Windows Identity Foundation

Windows Identity Foundation, más conocido por sus siglas, WIF permite a los desarrolladores .NET descargar a las aplicaciones de la responsabilidad de contener la lógica de gestión de la identidad de sus usuarios, proporcionando un modelo de programación basado en la separación de responsabilidades.

Los desarrolladores menos expertos pueden asegurar sus aplicaciones sin estar expuestos a la complejidad subyacente de la criptografía y protocolos, aprovechando las características de integración de Visual Studio, las cuales permiten implementar aplicaciones seguras empleando estándares como WS-Federation y WS-Trust.

Además de ofrecer un modelo de programación sencillo, que unifica las aplicaciones ASP.NET y los servicios WCF bajo un mismo modelo de objetos, Windows Identity Foundation dispone de una amplia gama de funcionalidades ofrecidas por WS-Security, el formato de token SAML y muchos otros estándares del sector industrial.

Empleando Windows Identity Foundation el sistema de autenticación lo proporcionan servicios externos a la aplicación y empleando protocolos estándares para la comunicación. La aplicación que hace uso de este framework recibe la información de los usuarios autenticados como Claims, que pueden ser usados por la aplicación para tomar decisiones.

Aunque los servicios de autenticación pueden utilizar diferentes proveedores de identidad, la mejor forma de aprovechar su funcionalidad es utilizar la autenticación proporcionada por Active Directory Federation Services 2.0, por lo menos en redes empresariales dónde ya disponen de un directorio activo con la información de sus usuarios.

2.2.- Active Directory Federation Services 2.0

Aunque Active Directory Federation Services 2.0 (AD FS 2.0) es una tecnología que puede ser empleada en soluciones que no residan en la nube, juega un papel muy importante en la autenticación de aplicaciones que funcionan en Windows Azure.

AD FS 2.0 extiende la funcionalidad de Active Directory dotando a éste de un sistema de identidad basado en claims. AD FS 2.0 provee de un Security Token Service (STS), cuya información reside en AD, que permite ofrece una interfaz para que las aplicaciones puedan disponer de un sistema de autenticación, independientemente de que la aplicación se encuentre desplegada en la nube o en una solución a medida (on-premise)

Los usuarios ya no están limitados por las fronteras de su red local: si una aplicación alojada en Windows Azure se ha desarrollado utilizando Windows Identity Foundation, AD FS 2.0 permite al instante el acceso a cualquiera cuenta válida en el directorio local para esta aplicación. Todo ello sin requerir ningún tipo de sincronización, duplicación y sin tener que crear una nueva cuenta de aprovisionamiento.

AD FS 2.0 facilita la creación y el mantenimiento de relaciones de confianza con partners federados, lo que simplifica el acceso a los recursos y la posibilidad de disponer de un "single sing-on". Todo estos trabajos, se basan en estándares de la industria y aceptados por la W3C como WS-Trust y WS-Federation, además del protocolo SAML. Una buena prueba de la interoperabilidad de ADFS es que ha superado con éxito las últimas pruebas públicas Liberty Alliance SAML 2.0 que proveen de interoperabilidad con productos de IBM, Novell, Ping Identity, SAP, Siemens y muchos otros.

2.3.- AppFabric Access Control

AppFabric Access Control permite generar una autorización federada entre aplicaciones y servicios, sin la programación complicada que, por lo general, se requiere para proteger aplicaciones que atraviesan los límites de la organización.

Las aplicaciones puede delegar en Access Control el sistema de autorización permitiendo escenarios simples basados en validación con un usuario o contraseña o escenarios empresariales más complejos dónde se emplea Active Directory Federation Services 2.0.

Desde el portal de Windows Azure, dentro de la administración de Access Control es posible incluir a AD FS 2.0 como proveedor de identidad de Access Control.

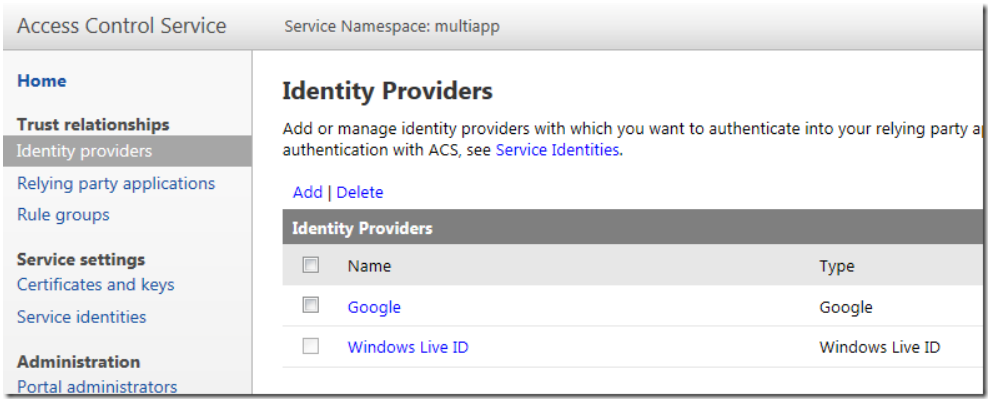


Figura 1.3.- Añadir proveedores de identidad

3.- APPFABRIC ACCESS CONTROL

3.1.- Terminología

A modo de resumen, se comentan algunos de los términos que se emplean cuando se habla de Access Control.

- **Identidad:** El término identidad suele emplearse para referirse al conjunto de claims que dispone un usuario y que han sido devueltos por una entidad certificada, Access Control en el caso de Windows Azure. Los claims devueltos por Access Control componen la identidad del usuario que realiza la conexión.
- **Claim:** Es la unidad mínima con la que se describe la identidad de un usuario o sistema, por ejemplo, su nombre, su edad, su dirección de correo, una hash, un thumbprint etc...Podría describirse como un atributo nombre-valor. Los claims sirve para componer la identidad de un usuario o sistema.
- **Token de Seguridad:** Un token de seguridad no es más que un conjunto de claims que definen la identidad de un usuario. Cuando un usuario intenta realizar una comunicación con un cliente debe indican en la petición el token de seguridad, su conjunto de claims, para que el servicio que recibe la petición pueda conocer la identidad del usuario que realiza la llamada y poder actuar en consecuencia.
- **Secure Token Service**: Es la entidad proveedora de claims, Access Control en el caso de Windows Azure. Un STS debe ser capaz de recibir peticiones de los clientes a través de protocolos interoperables, validar al usuario y devolver los claims asociados a él.
- **Proveedor de identidad:** Es la entidad que realiza la autenticación de usuarios, aquella que se encarga de comprobar que el usuario dice quién dice ser. El Secure Token Service hará uso de uno o más proveedores de identidad para comprobar la validez del usuario. Un proveedor de identidad puede ser Active Directory, Windows Live ID o Facebook.

Relaying Party: Es el servicio o aplicación que requiere de los servicios de autenticación y autorización y que implementa un mecanismo de seguridad basado en claims.

4.- SEGURIDAD BASADA EN CLAIMS

Hasta ahora ya hemos visto que es la seguridad basada en claims así como la definición de los componentes más importantes. A mayores, también hemos visto como Access Control Service y ADFS son dos implementaciones de este modelo de seguridad. A continuación se hablará sobre el uso de este modelo de seguridad en nuestros servicios/aplicaciones.

Cuando se utiliza Access Control dentro de un servicio, el usuario que quiera conectarse al servidor debe obtener primero un token de seguridad del Access Control para poder conectarse. Un token de seguridad está compuesto por un conjunto de claims. Un claim puede definirse como un atributo nombre-valor, que aporta información sobre la identidad del usuario que intenta realizar la conexión.

Figura 1.4.- Ejemplo de Claims

Para obtener el token de seguridad mencionado anteriormente Access Control, deberá comprobar previamente la identidad del usuario que los solicita. El usuario debe poder demostrar su identidad, de alguna forma para que Access Control pueda validarla contra el proveedor que tenga configurado; Active Directory, Windows Live ID, etc.

Una vez validado el usuario, Access Control devolverá el token de seguridad al usuario. El usuario obtendrá este token y realizará la llamada al servicio, pasando el mismo en la petición.

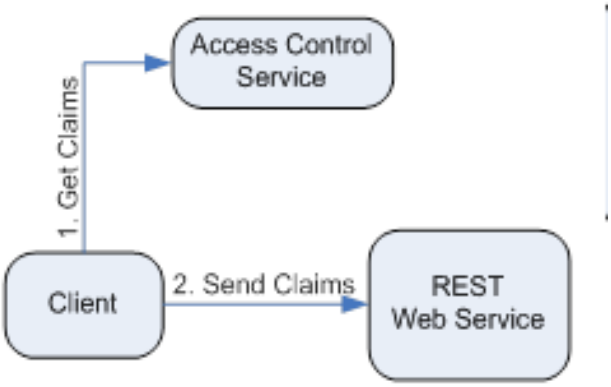


Figura 1.5.- Proceso de conexión

Access Control permite un sencillo modelo declarativo de reglas por el cual pueden configurarse el token a devolver, el conjunto de claims que contenga, en función del usuario que realizar la solicitud. Los beneficios de este modelo son claros, de entrada es más fácil de conseguir un escenario de single sing-on y los servicios que se implementan se pueden olvidar de las siguientes responsabilidades:

- Autenticación de usuarios.
- Guardar usuarios y contraseñas.
- Llamar a entidades externas, como Active Directory, para validar las identidades.
- Integración con proveedores de identidades externos.

4.1.- Descripción del proceso

En la Figura 1.5 puede verse el diagrama general que describe el proceso a seguir por una aplicación cliente que desea conectarse a un servicio REST.El primer paso que debe realizar siempre el cliente es conectarse con el Access Control, para obtener un token de seguridad que le permita conectarse al servicio.

Para poder obtener el token el cliente realizar una petición por HTTP POST al Access Control incluyendo el nombre de usuario y contraseña y la URI a la que desea conectarse, la URI dónde se encuentra el servicio.

Access Control deberá validar la identidad del usuario. En primera instancia validará el usuario y la contraseña con el proveedor que tenga configurado. Si la validación es correcta, devolverá el token de seguridad, que está compuesto por un conjunto de claims. Los claims que se incluyen dentro del token de seguridad dependen de las reglas configuradas en el Access Control, reglas que pueden personalizarse.

Una vez que el cliente dispone del token de seguridad deberá incluirlo dentro de las cabeceras HTTP de la petición. El servicio web leerá las cabeceras HTTP, validará el contenido de la cabecera y obtendrá los claims que en ésta se incluyen. La aplicación podrá disponer de la lógica necesaria para actuar en función de los claims recibidos.

A continuación, en el siguiente fragmento de código se muestra un ejemplo de cómo interactuar con Access Control para solicitar un token de seguridad. Access Control permite recibir solicitudes a través de un protocolo llamado WRAP (Web Resource Authorization Protocol), basado en REST.

```
private static string GetTokenFromACS(string issuerKeySuppliedByCaller)
{
    // request a token from ACS
    WebClient client = new WebClient();
    client.BaseAddress = string.Format("https://{0}.{1}",
      serviceNamespace, acsHostName);

    NameValueCollection values = new NameValueCollection();
    values.Add("wrap_name", "my-issuer");
    values.Add("wrap password", issuerKeySuppliedByCaller);
    values.Add("wrap_scope", "http://localhost/ACSGettingStarted");

    byte[] responseBytes = client.UploadValues("WRAPv0.9", "POST", values);

    string response = Encoding.UTF8.GetString(responseBytes);

    return response
        .Split('&')
        .Single(value => value.StartsWith("wrap access token=",
           StringComparison.OrdinalIgnoreCase))
        .Split('=')[1];
}
```

Una vez recuperado el token, éste debe ser incluido en las cabeceras HTTP.

```
private static string SendMessageToService(string token, string valueToReverse)
{
    WebClient client = new WebClient();
    client.BaseAddress = "http://localhost/ACSGettingStarted/Default.aspx";

    string headerValue = string.Format("WRAP access_token=\"{0}\"",
       HttpUtility.UrlDecode(token));


    client.Headers.Add("Authorization", headerValue);

    NameValueCollection values = new NameValueCollection();
    values = new NameValueCollection();
    values.Add("string_to_reverse", valueToReverse);

    byte[] serviceResponseBytes = client.UploadValues(string.Empty, values);

    return Encoding.UTF8.GetString(serviceResponseBytes);
}
```

5.- MI PRIMERA APLICACIÓN CON ACCESS CONTROL

A continuación podrá verse de forma detallada un ejemplo sencillo que muestra cómo crear una aplicación web que hace uso de Access Control y que emplea Windows Live ID para la autenticación de usuarios.

En el ejemplo se ha empleado Visual Studio 2010, Windows Identity Foundation y Windows Identity Foundation SDK. El primer paso a realizar debe ser crear un namespace de AppFabric en la cuenta de Windows Azure, para disponer de un servicio de Access Control.

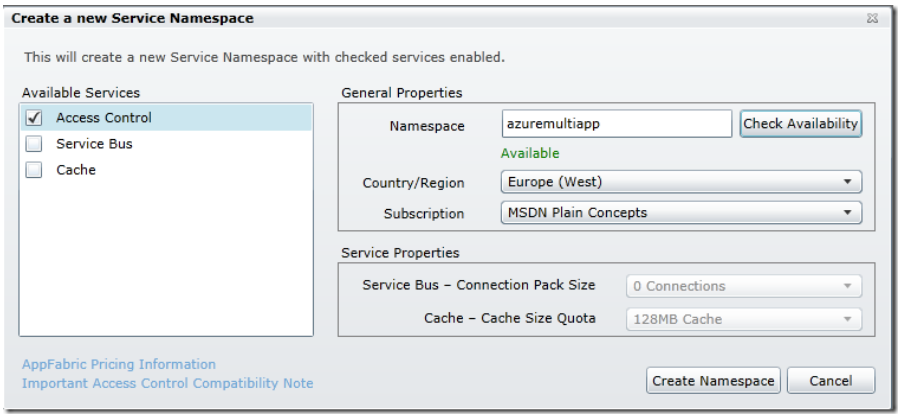


Figura 1.6.- Crear namespace

Una que vez está creado el namespace, es posible entrar en la administración de "Access Control" y seleccionar la opción "Administrar".

A través de esta pantalla se podrá configurar todos los aspectos que se necesitan para securizar la comunicación. En el ejemplo se realizan los siguientes pasos:

- Configurar los proveedores de identidad; Windows Live, Google, Facebook, Active Directory…
- Dar de alta la aplicación a securizar (Relaying Party).
- Definir las reglas de AppFabric.
- Añadir los certificados necesarios para la encriptación y firmado.

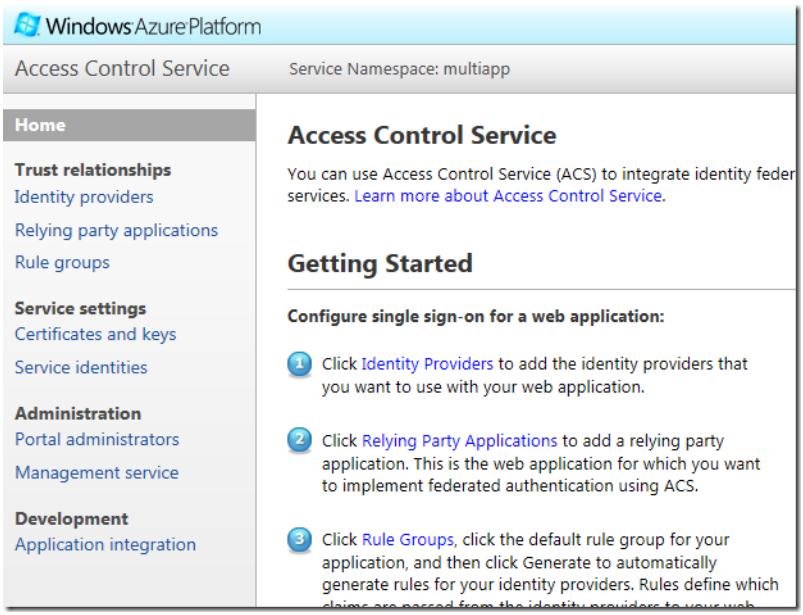


Figura 1.7.- Configuración de Access Control

Seleccionar los proveedores de identidad. En este ejemplo se empleará Windows Live ID y Google.

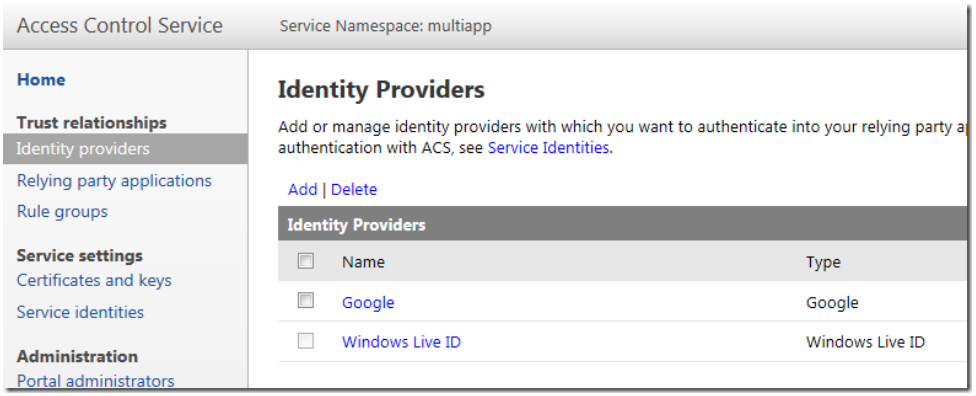


Figura 1.8.- Añadir proveedores de identidad

Dar de alta la aplicación dentro de Access Control. En el campo Realm y Return URL debe indicarse la URL dónde está la aplicación Web. En un caso real nunca debería contener la "localhost" la URL. En este ejemplo, URI será la que genera Visual Studio al arrancar la aplicación en modo depuración (F5).

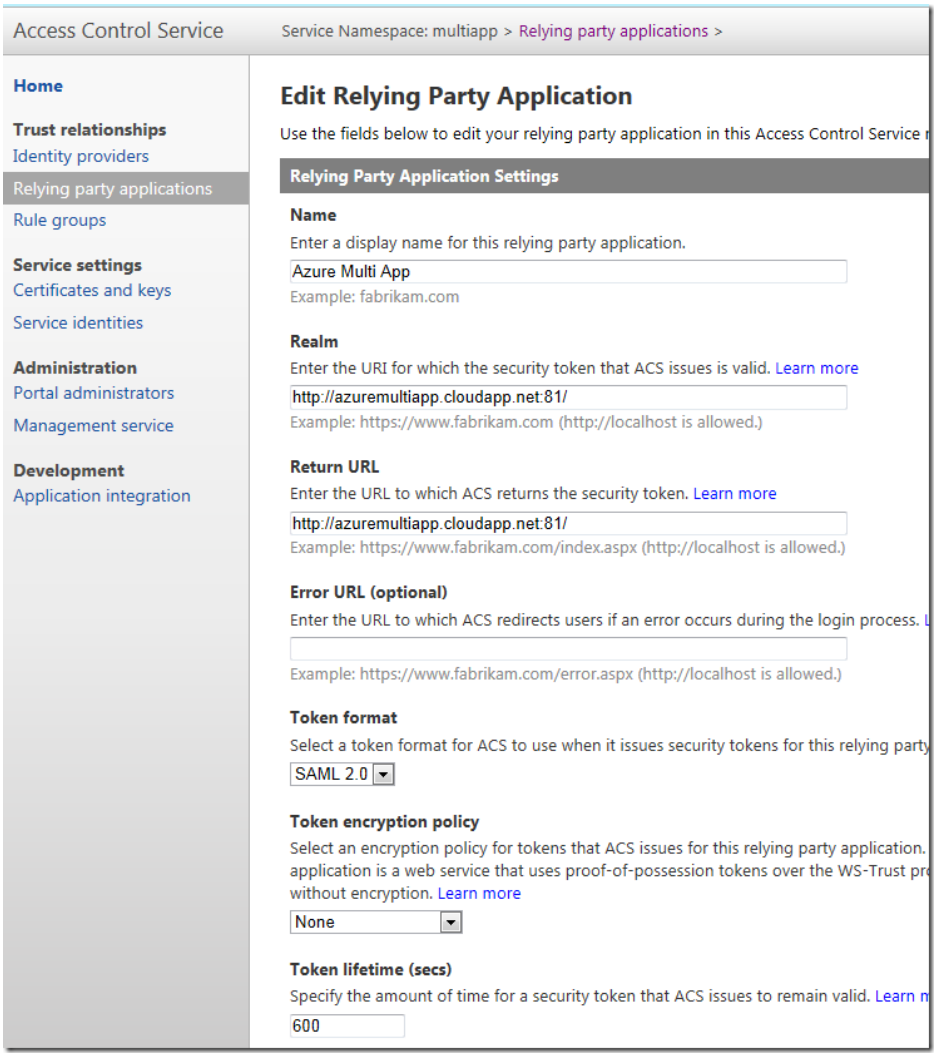


Figura 1.9.- Configurar Relaying Party.

Una vez dada de alta la aplicación, deben crearse las reglas. Por simplificar, en este punto no se ha incluida ninguna regla y se usan las que la propia aplicación genera para los proveedores de identidad seleccionados.

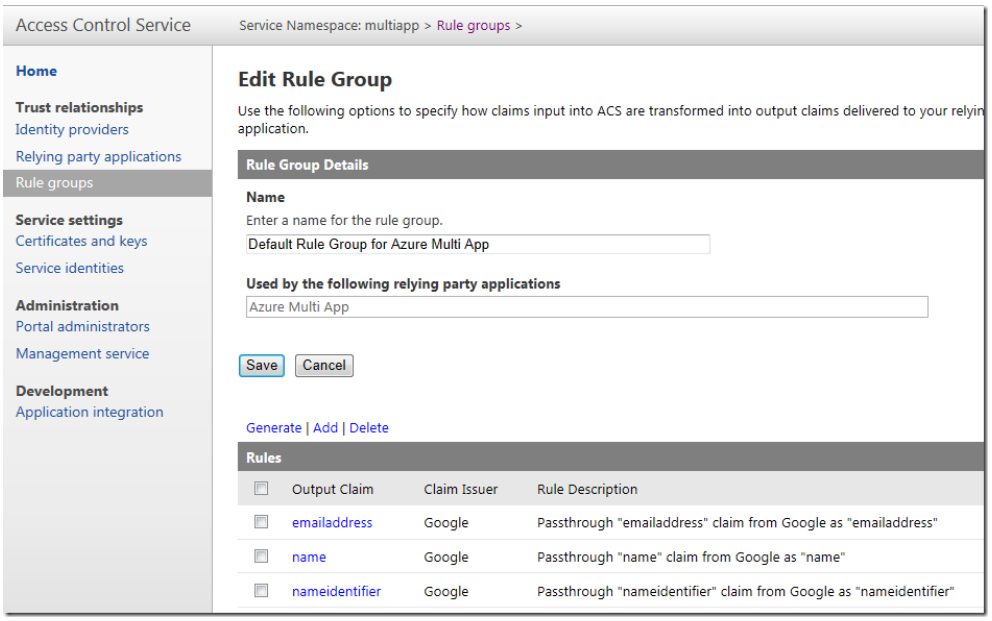


Figura 1.10.- Reglas de Access Control

Una vez se ha terminado los pasos anteriores, ya es posible crear una aplicación web y configurarla para que utilice Access Control.

Al crear el proyecto puede ser buena práctica establecer un puerto fijo en la configuración del proyecto, para que sea el mismo que se ha configurado al dar de alta la "Relaying Party".

Una vez esto, será necesario añadir la referencia al STS (Secure Token Service), que en este caso es Access Control.

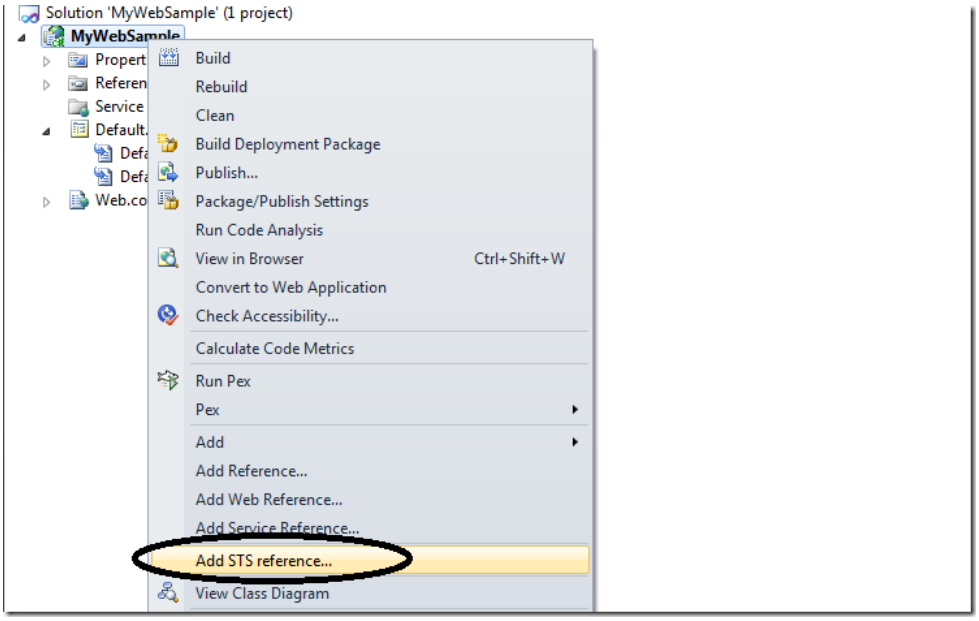


Figura 1.11.- Añadir la referencia a STS.

Se indica la ruta dónde se encuentra el fichero de la aplicación y se indica la URI de la aplicación, la misma que se configurado en el Relaying Party.

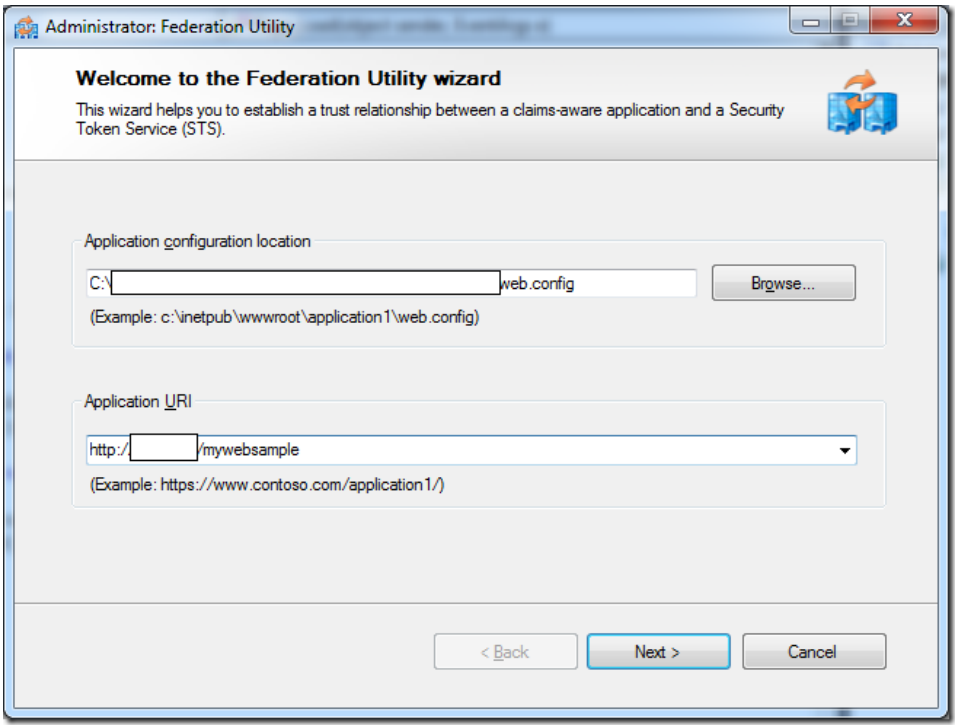


Figura 1.12.-Añadir la referencia a STS

Dentro del portal de Windows Azure se indica la información necesaria para conectarse a Access Control. Entre esa información se encuentra al URI que debemos emplear para acceder a la información de WS-Federation. Este URL es necesario para dar el alta el STS en la aplicación web.

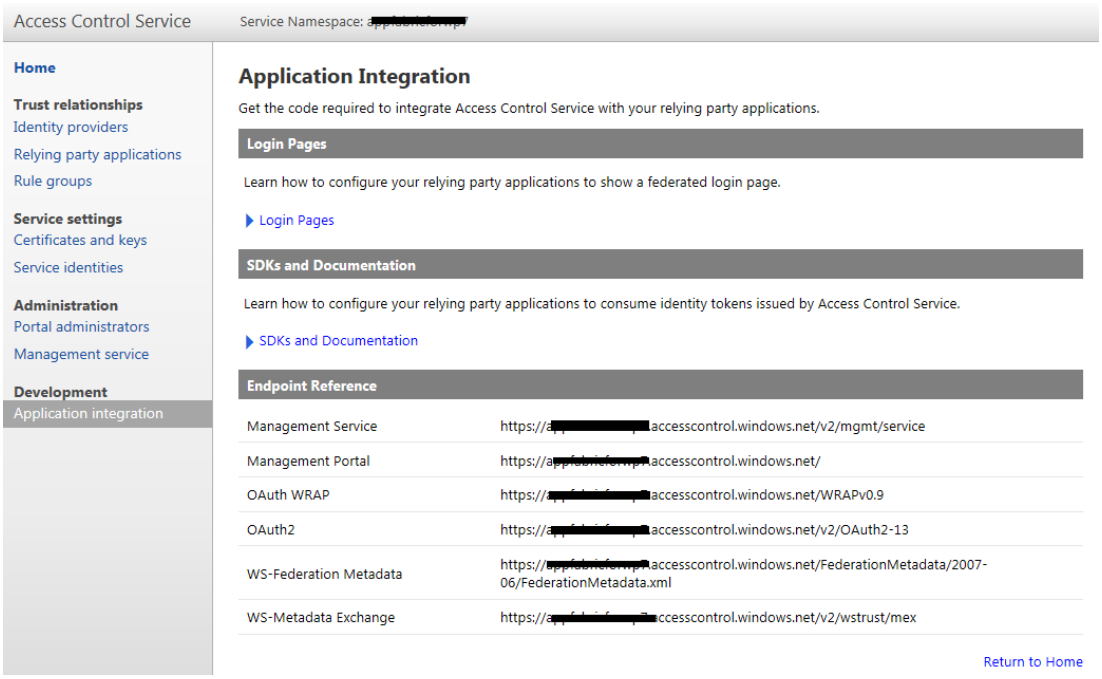


Figura 1.13.- Información de Access Control

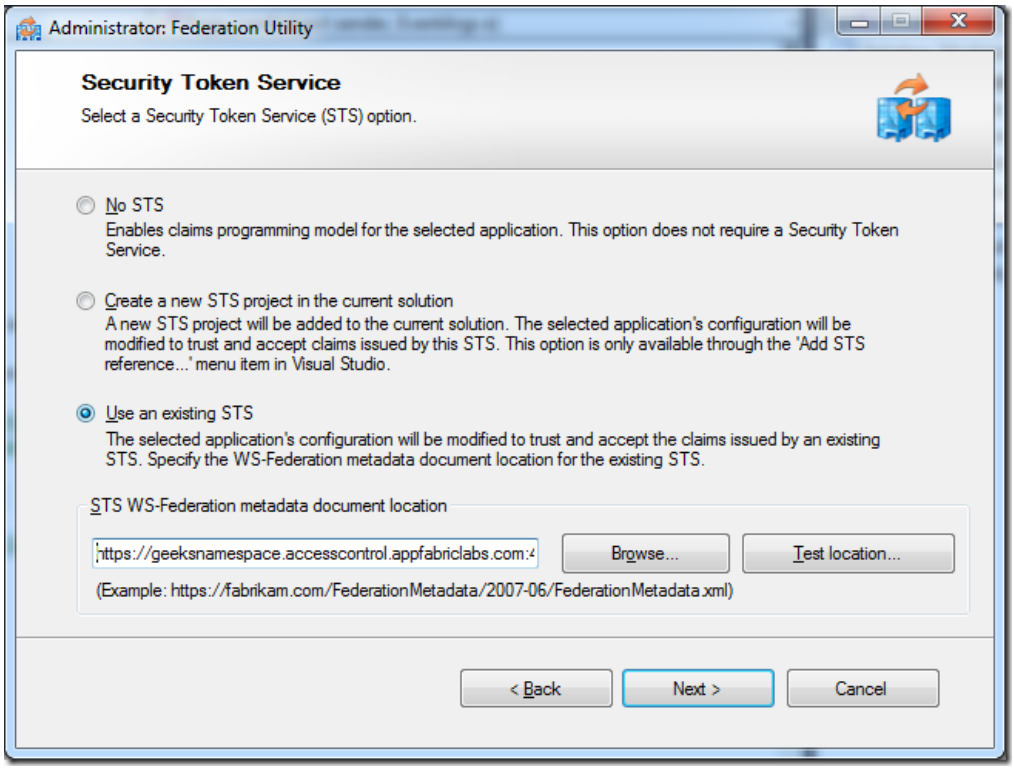


Figura 1.14.- Añadir la referencia a STS

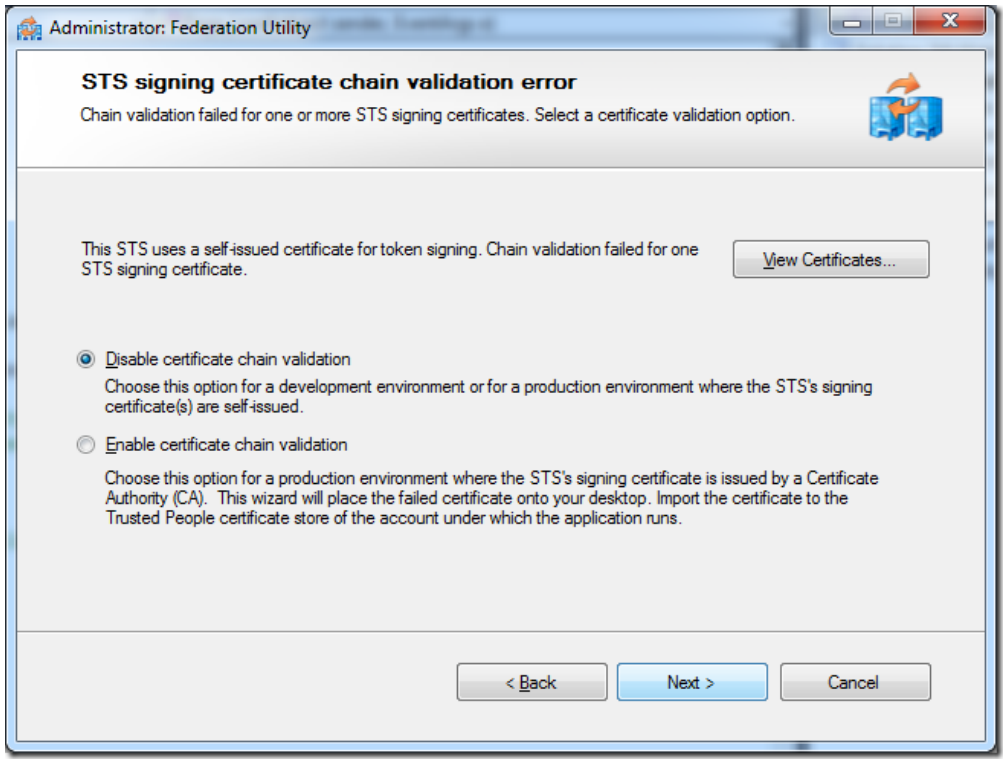


Figura 1.15.- Añadir la referencia a STS

Figura 1.16.- Añadir la referencia a STS

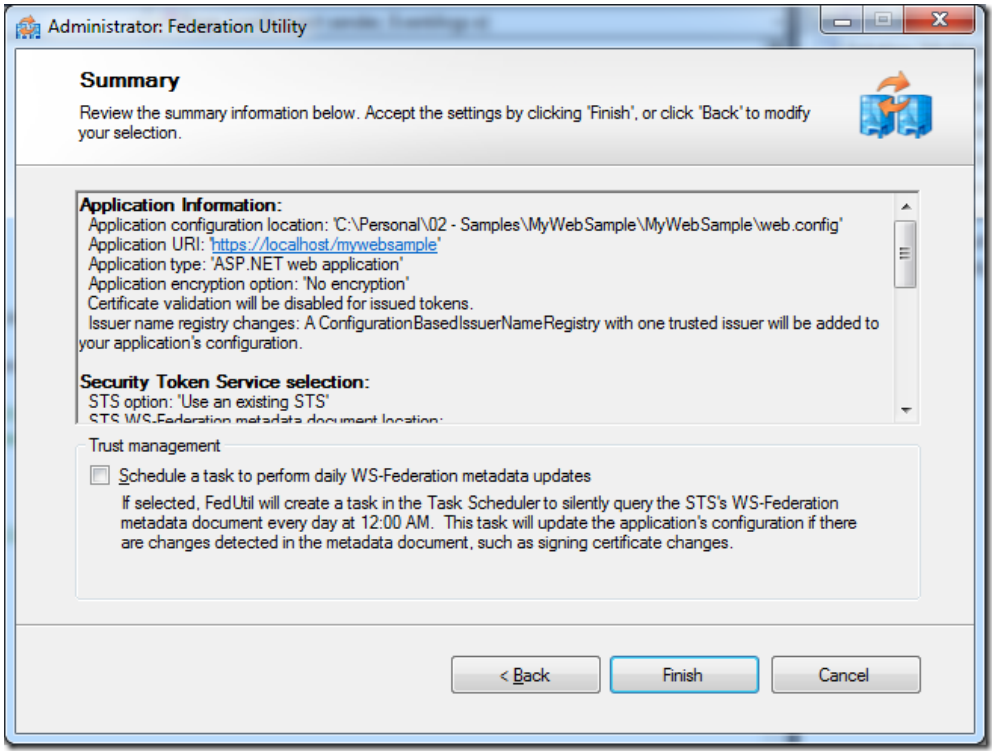


Figura 1.17.- Añadir la referencia a STS

Por último, antes de ejecutar la aplicación será necesario establecer el siguiente valor en el fichero de configuración de la aplicación Web:

```
<system.web>
    <httpRuntime requestValidationMode="2.0" />
    <authorization>
      <deny users="?" />
```

Al ejecutar la aplicación web, se mostrará una ventana como la siguiente, dónde se puede elegir el proveedor de identidad que quiere emplearse para la validación. Esta pantalla muestra los proveedores configurados anteriormente.

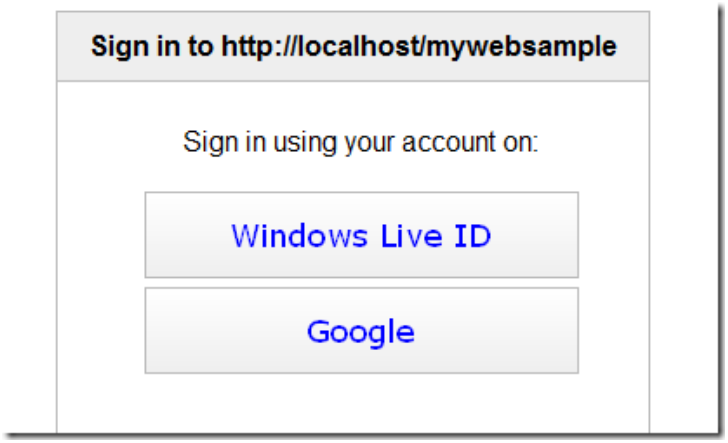


Figura 1.18.- Seleccionar proveedor de identidad

Si por ejemplo se selecciona Windows Live ID, se mostrará la pantalla de validación usuarios de Windows Live, dónde el habrá que autenticarse usando un usuario de Windows Live ID. Una vez validado con el usuario de Live ID, se podrá acceder a la aplicación.

6.- WINDOWS AZURE SERVICE BUS

Windows Azure Platform AppFabric proporciona un bus de servicios empresarial y un servicio de control de acceso que permite integrar servicios y aplicaciones que se ejecutan en la nube, en proveedores de alojamiento tradicionales y en cualquier otra ubicación basándose en estándares de interoperabilidad.

Un bus de servicios empresarial, conocido como AppFabric Service Bus, nos permite orquestar la conectividad segura entre diferentes servicios y aplicaciones a través de cortafuegos y redes utilizando numerosos y diferentes patrones de comunicación.

Los diferentes servicios se registran en el bus de servicios de manera que puedan ser fácilmente accedidos a través de las más variadas tipologías de red (como por ejemplo los dispositivos móviles bajo la red de un operador de telefonía). Si una aplicación tiene que consumir e interactuar con una gran cantidad de servicios, algunos de ellos controlados por terceros, utilizar un bus de servicios permite "olvidarse" de detalles como la autenticación y autorización, los protocolos de comunicación, los cortafuegos y otras cuestiones técnicas, delegándolos en el bus de servicios. De esta manera, los desarrolladores pueden centrarse en solucionar escenarios de negocio y no perderse en los detalles de implementación de los servicios.

Uno de los usos habituales del Service Bus de la plataforma Azure es facilitar la labor de conectar aplicaciones que se ejecutan sobre Windows Azure o contra SQL Azure con aplicaciones que corren en una infraestructura propia y contra servidores de bases de datos convencionales.

Otro escenario en el que el bus de servicios ayuda enormemente es en la creación de aplicaciones compuestas mediante la integración de diferentes servicios ya existentes y nuevos servicios que se ejecutan en la plataforma Azure.

A continuación puede verse el esquema de funcionamiento del bus de servicios de la Platform Azure.

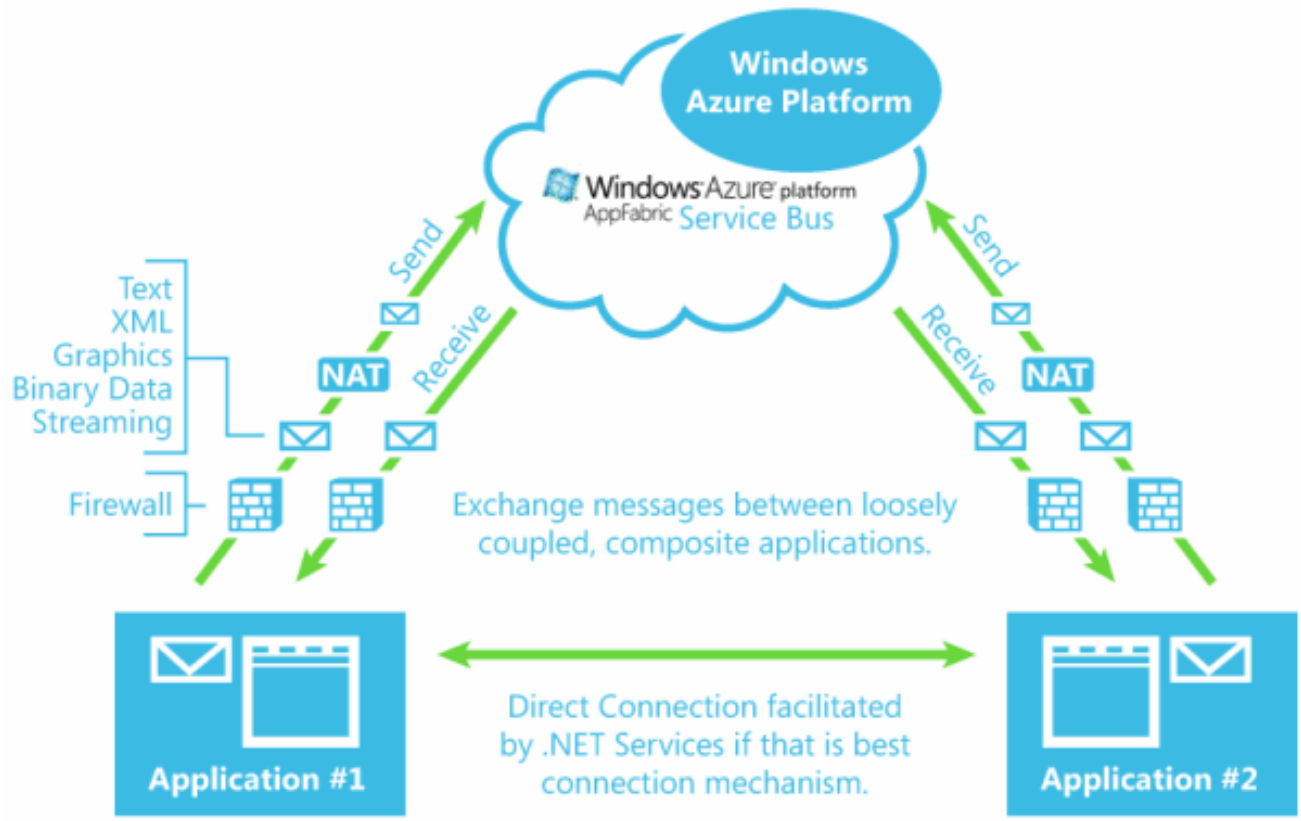


Figura 1.19.- Esquema de funcionamiento de Service Bus

6.1.- Nomenclatura y servicios de registro

Para poder empezar a trabajar con AppFabric Service el primer paso es disponer de una cuenta de Windows Azure. Una vez que ya disponemos de esta una cuenta se pueden asociar a ella tanto *namespaces* como se necesiten, siempre y cuando estos namespaces sean únicos entre todas las cuentas existentes de AppFabric. Un *namespace* es el espacio de nombres que se utiliza para administrar los tokens de seguridad en el servicio de control y acceso de AppFabric, y es, además, un espacio de nombres bajo los cuales pueden registrarse cualquier número de servicios.

Hay que tener en cuenta que se aunque pueden declararse varios espacios de nombres dentro de una cuenta de Azure, cada uno estará completamente aislado de los demás.

Una vez creada la cuenta de Windows Azure, desde la pestaña de AppFabric puede añadirse tantos espacios de nombres como se necesiten.

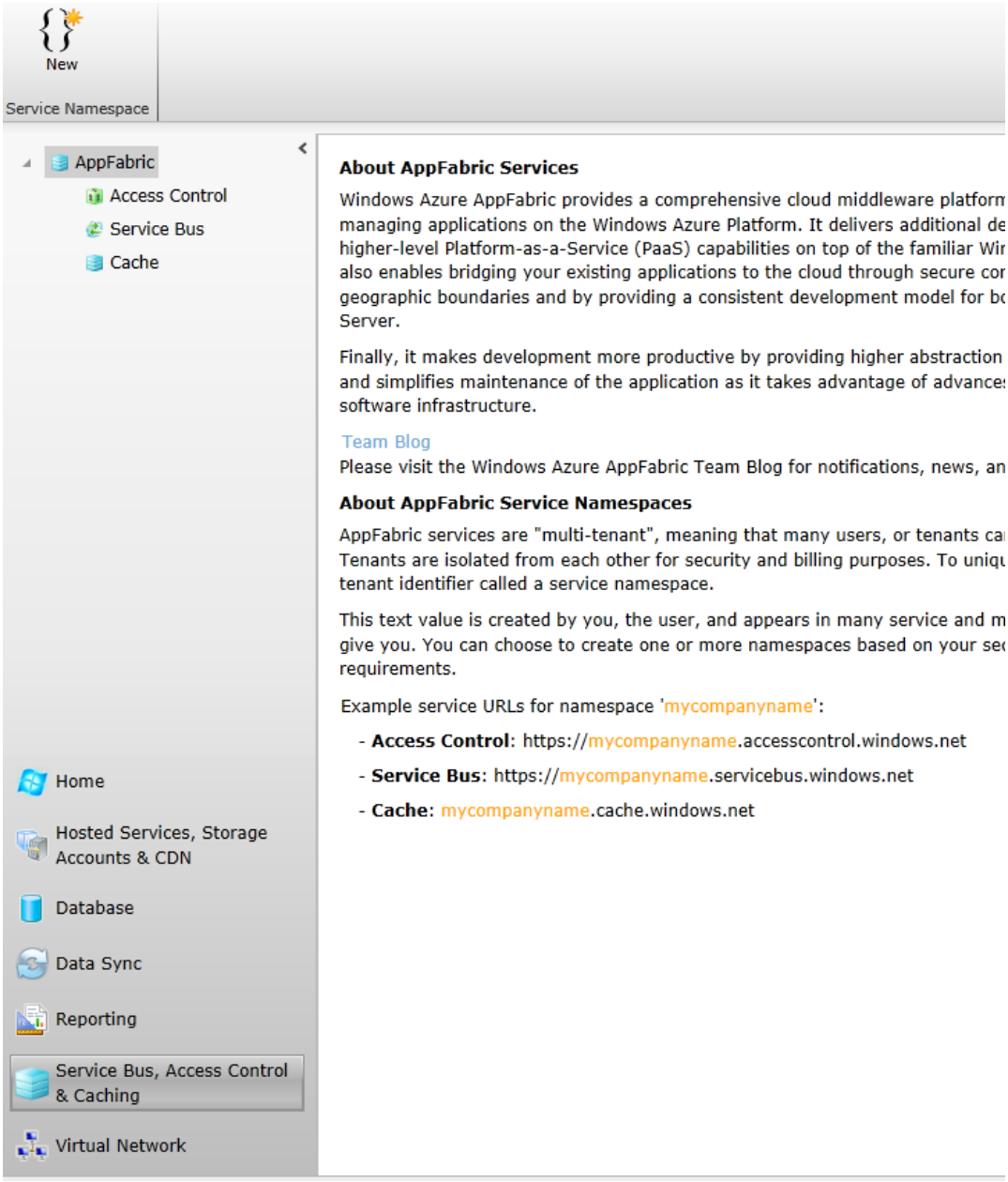


Figura 1.20.- Página principal de AppFabric

Un punto a tener en cuenta al definir un namespace es que el totalmente independiente de la ubicación y del tipo de binding que se usará en las comunicaciones. Los servicios que hagan uso de este componente podrán utilizar diferente formas de comunicación y podrán residir en diferentes ubicaciones.

En la creación del namespace será necesario indicar, como ya se comentó anteriormente, el nombre único del mismo y el data center dónde se quiere que se ubique. Lógicamente, es importante establecer la ubicación lo más cercana posible a los servicios que usarán esta funcionalidad.

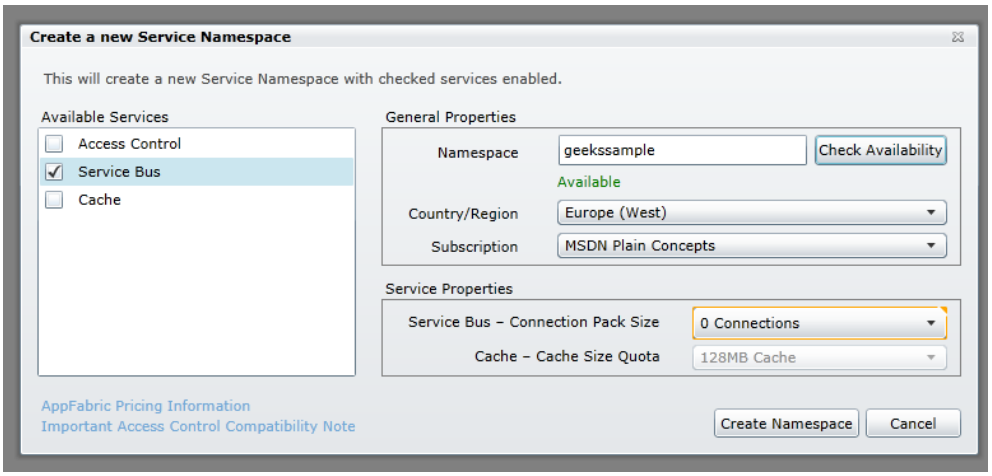


Figura 1.21.- Creación de un namespace

Una vez creado el namespace, el portal de Windows Azure ofrecerá todos los datos necesarios para poder emplear la funcionalidad de Service Bus y Acces Control.

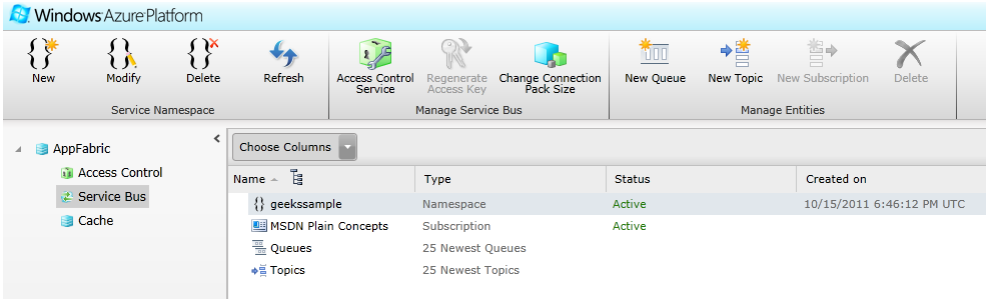


Figura 1.22.- Creación de un namespace

6.1.1.- Registro

Por defecto, los servicios registrados en la AppFabric Service Bus son privados. Sin embargo, es posible configurar el Service Bus para hacer los endpoints públicos cuando se realiza el registro.

Service Bus expone los endpoints de los servicios públicos a través de un feed ATOM 1.0, para que puedan ser descubiertos por cualquier aplicación que conozca el URI base del namespace.

A través de la configuración del behavior del endpoint puede establecerse que un determinado extremo sea público y por lo tanto, que sea visible a través del feed ATOM. En las siguientes líneas se puede ver cómo podemos agregar un nuevo comportamiento, incluido en el SDK de App Fabric Service Bus, a un servicio WCF con el fin de hacer este servicio descubrible.

```
ServiceRegistrySettings serviceRegistrySettings = newServiceRegistrySettings(DiscoveryType.Public);
serviceRegistrySettings.DisplayName = "MyService";

foreach (ServiceEndpoint subscriberEndpoint in subscriberHost.Description.Endpoints)
{
    subscriberEndpoint.Behaviors.Add(serviceRegistrySettings);
}
```

Una vez publicado, a través de la URI http://<namespace>.servicebus.windows.net/ se podría acceder el feed que permite descubrir los servicios públicos.

6.2.- Mensajería

El service bus de AppFabric tiene, entre otros, como objetivo resolver el desafío de conectividad de Internet, que en muchas ocasiones resultan tremendamente difícil. A menudo, el servicio con el que conectarse se encuentra detrás de los servidores de seguridad (firewalls software y/o hardware,) detrás de un balanceador de carga, su dirección es dinámica o sólo se puede resolver su dirección si está la red local, etc.

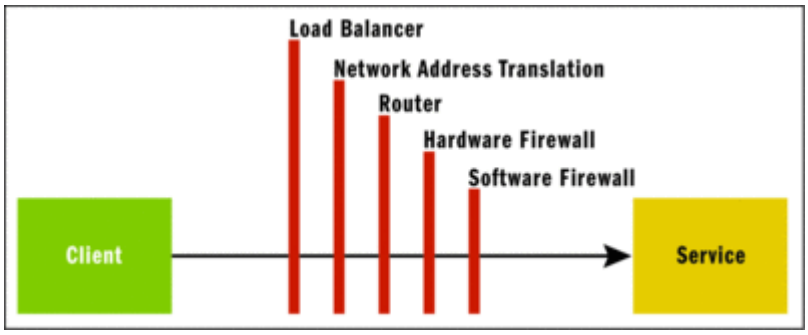


Figura 1.23.- Retos de la comunicación

6.2.1.- EL servicio de relay

La solución de conectividad a través de Internet puede ser muy sencilla. Si es tan difícil conectar al cliente con el servicio directamente, pues habrá que evitar conectarnos directamente y utiliza en su lugar un servicio de relay, que se encargue de retransmitir los mensajes que van desde el cliente al servidor. El servicio de relay, es un servicio que está en la nube y tiene como objetivo, ayudar a comunicar aplicaciones entre sí, evitando los posibles problemas que puede haber es una comunicación directa.

El Service Bus de AppFabric actúa de relay y es capaz de retransmitir los mensajes provenientes de un cliente al destino del mismo. Lógicamente, ambos puntos de la comunicación debe tener acceso al componente que actúa de relay, situación que habitualmente ocurre ya que el servicio de relay se encuentra situado en un lugar "neutral" y, por lo tanto, las aplicaciones son capaces de realizar comunicaciones de salida hacia el relay.

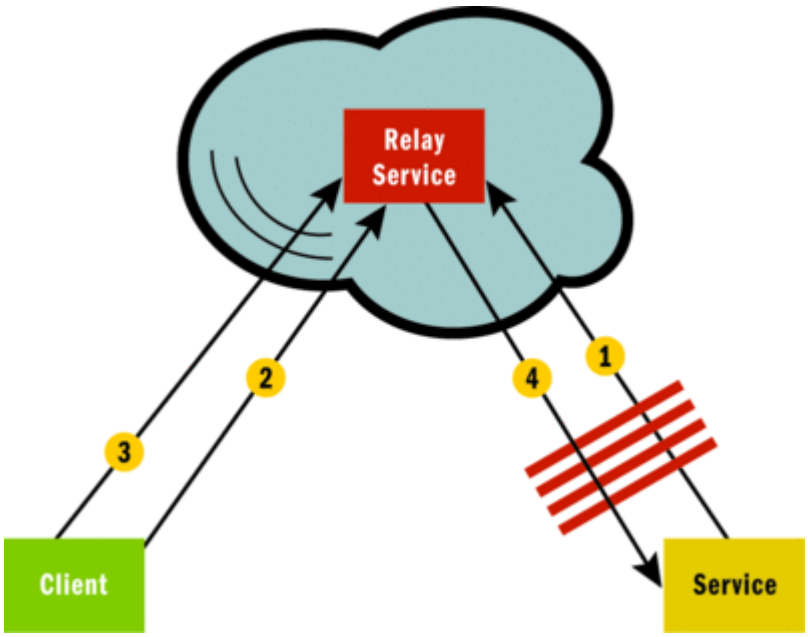


Figura 1.24.- Comunicación a través del servicio de relay

Para poder realizar la comunicación entre el cliente y el servicio es necesario que ambos componentes se registre en el servicio de relay. El registro requiere de un paso de autenticación, para evitar que cualquiera pudiera hacer uso de este servicio.

Para el caso de plataforma .NET podremos interactuar con el servicio de Relay de Windows Azure utilizando el mismo API que utilizamos para otro tipo de comunicaciones, Windows Communication Foundation. Es importante destacar que en este caso, a diferencia de una comunicación directa empleando WCF, el servicio siempre tiene que iniciar al menos una comunicación con el servicio de relay para poder recibir peticiones de los clientes. Una vez registrado el servicio, éste quedará a la espera de que el servicio de relay le retransmita los mensajes provenientes del o de los clientes. Trabajar con el service bus no es muy diferente a desarrollar aplicaciones con WCF que no hagan uso de este servicio de relay. El API de Service Bus simplemente añade nuevo bindings a los ya existentes para WCF, bindings que permiten la comunicación entre un cliente y un servicio a través de un servicio de relay por ejemplo.

6.3.- Relayed Messaging vs Brokered Messaging

Una de las novedades que ha aportado el SDK 1.5 de Windows Azure ha sido la aparición de un conjunto de mensajes en service bus.

Cuando se habla de Service Bus es importante distinguir entre dos tipos de comunicaciones o sistema de mensajería; Relayed Messagins y Brokered Messaging.

6.3.1.- Relayed Messaging

Este tipo de mensajes existen desde la aparición de la primera versión de Service Bus, cuya primera versión reléase es de enero de 2010.

En este escenario un servicio hosteado en Windows Azure hace de relay entre los dos puntos de la conectividad cliente-servidor, haciendo posible la comunicación entre ambos aunque haya elementos por medio que pudieran complicar dicha comunicación.

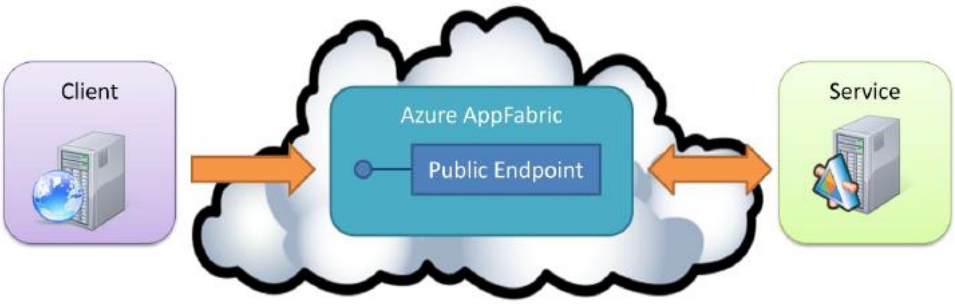


Figura 1.25.- Relayed Messaging

Este tipo de comunicación ofrece ventajas claras en situaciones dónde la conectividad es un "problema", pero también, desde el punto de vista de arquitectura, hay que tener en cuenta otras características del sistema si se opta por este tipo de mensajes.

En este tipo de escenario tanto cliente como servidor tiene que estar conectado al servicio de AppFabric para que la comunicación sea posible.

Otra aspecto a tener en cuenta es que en situaciones de carga, dónde muchos clientes quisieran conectarse con el servidor, el rendimiento del sistema podría verse afectado y tener situaciones de "timeout"s en las comunicaciones. Es una solución que desde el punto de vista de balanceo de carga o escalabilidad no ofrece ninguna solución clara que cubra estos escenarios.

6.3.2.- Brokered Messaging

El segundo tipo de mensajes es una de las nuevas características del SDK 1.5, versión de septiembre de 2011.

Es este escenario el servicio de service bus hosteado en la nube hace de bróker de mensajes entre los clientes y servidores, ofreciendo un servicio de almacenamiento persistente de los mensajes "en tránsito".

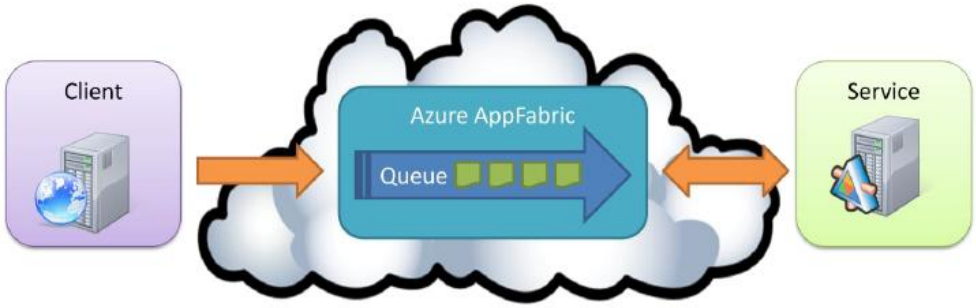


Figura 1.26.- Brokered Messaging

Con este tipo de mensajes se introducen las tecnologías de colas, topics y subscripciones, las cuáles se utilizan para diferentes tipos comunicación entre cliente y servidor. Este tipo de mensajes cubre perfectamente el típico patrón publicador-subscriptor de muchas aplicaciones.

En este tipo de escenarios la comunicación no ocurre de forma síncrona, no siendo un requisito necesario que ambas partes de la comunicación se encuentren siempre disponibles.

Desde el punto de vista de cliente el servidor siempre es capaz de atender las peticiones, ya que éstas se almacenan en el servicio que hace de relay. Cuando el servidor se encuentre online ya se encargará de recibir las peticiones y procesarlas.

Este modelo, a diferencia del anterior, es completamente válido para escenarios de alta disponibilidad, dando a su vez opción para implementar soluciones de balanceo de carga.

Aunque las **colas de Service Bus** son una versión mucho más potente que la que ofrece Windows Azure Storage, puede tomarse como referencia el comportamiento de éstas últimas para entender la característica. Comparar este sistema con MSMQ podría ser una opción más acertada.

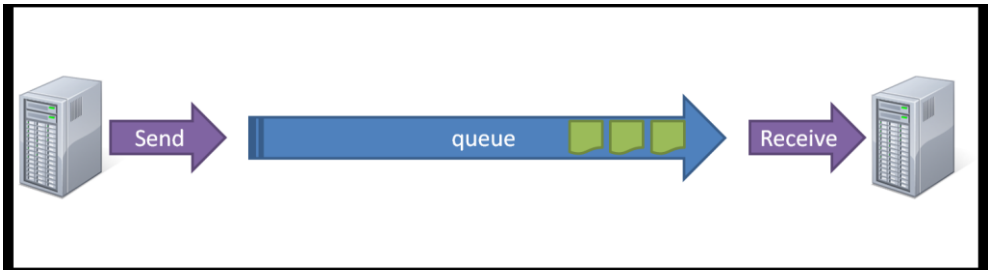


Figura 1.27.- Service Bus Queues

Los **topics y subscripciones** son una funcionalidad que extiende el concepto de cola, ya que permite que las aplicaciones sólo envíen o reciban mensajes basándose basado en ciertas condiciones, no teniendo que tratar todos los mensajes que pasen por las colas.

Una subscripción se crea a partir de un topic y un topic puede tener cero o más subscripciones asociadas.

La aplicación envía los mensajes a un determinado topic, siendo estos mensajes en rutados a las subscripciones existentes en base a las reglas que se hayan configurado.

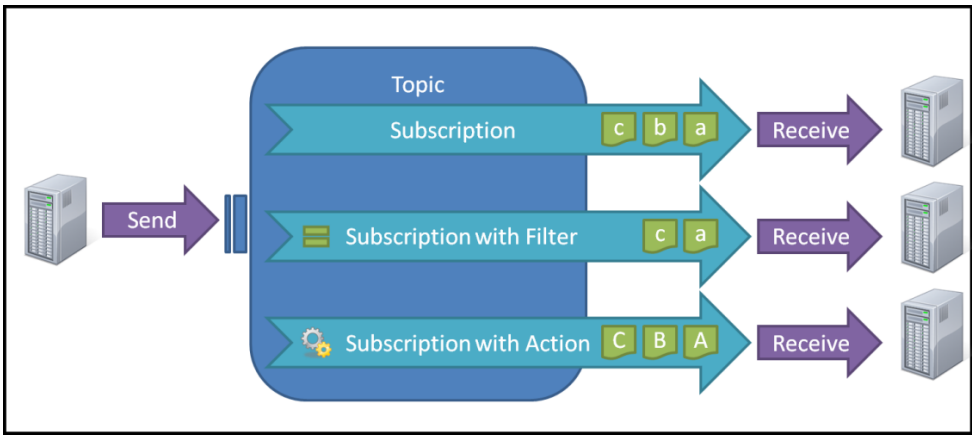


Figura 1.28.- Topics y subscripciones

6.4.- Modelo de programación

Como se puede ver a continuación el modelo de programación de los mensajes brokered ofrece varios modelos de programación.

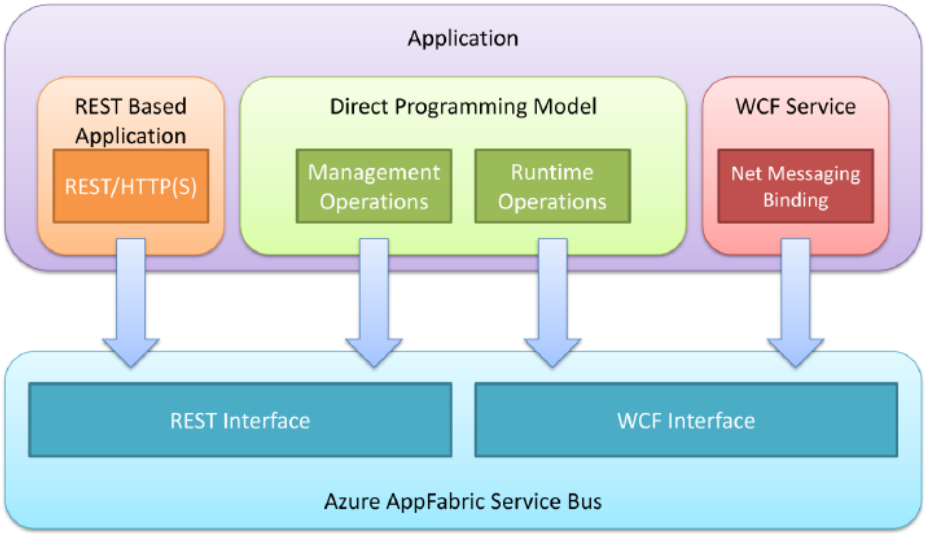


Figura 1.29.- Modelo de programación

6.4.1.- Interfaz REST

A través de REST es posible acceder a toda la funcionalidad que ofrece la runtime de Service Bus, opción que suele ser empleada cuando se quiere acceder desde tecnologías no .NET.

6.4.2.- Modelo directo

El modelo de trabajo directo ofrece una serie de librerías y clases de .NET que permite ser referenciadas desde proyectos .NET y que dan acceso de una forma clara y sencilla a toda la funcionalidad de Service Bus.

Por similitud, estas clases son similares a las que existen para trabajar con Windows Azure Storage.

En este caso la librería es Microsoft.ServiceBus.dll.

```
public bool SendMessageToQueue(string message, string queueName, MessageSettings settings)
{
 MessagingFactory factory = MessagingFactory.Create(serviceUri, credentials);
 QueueClient queueClient = factory.CreateQueueClient(queueName);
BrokeredMessage payLoad = GetPayload(message, settings);
queueClient.Send(payLoad);
return true;
}
```

6.4.3.- Modelo WCF

Del mismo modo que se puede hacer con los mensajes de tipo relayed, existe la posibilidad de trabajar con WCF. La clase NetMessagingBinding nos ofrece la posibilidad de desarrollar aplicaciones WCF que hagan uso de las nuevas capacidades de Service Bus, pudiendo manejar las colas, topics y subscripciones.

7.- DIFERENCIAS ENTRE WINDOWS AZURE STORAGE QUEUES Y SERVICE BUS QUEUES

En este apartado se muestra una gráfica resumen que muestra las diferencias entre el sistema de colas de Windows Azure Storage y AppFabric Service Bus, comparativa obtenida de **http://preps2.wordpress.com**

Aunque como se verá a continuación el sistema de colas de service bus es mucho más completo que el existente en el storage, no se debe caer en la tentación de emplear siempre el más completo, sino en aquel que sea mejor para la aplicación que se esté desarrollando.

Tabla 1.1.- Diferencias entre Windows Azure Storage Queues y Service Bus Queues

Feature	Windows Azure Storage Queues	Service Bus Queues	Comments
Programming Models			
Raw REST/HTTP	Yes	Yes	
.NET API	Yes (Windows Azure Managed Library)	Yes(AppFabric SDK)	
Windows Communication Foundation (WCF) binding	No	Yes	
Windows Workflow Foundation (WF) integration	No	Yes	

Protocols			
Runtime	REST over HTTP	REST over HTTPBi-directional TCP	The Service Bus managed API leverages the bi-directional TCP protocol for improved performance over REST/HTTP.
Management	REST over HTTP	REST over HTTP	
Messaging Fundamentals			
Ordering Guarantees	No	First-In-First-Out (FIFO)	Note: guaranteed FIFO requires the use of sessions.
Message processing guarantees	At-Least-Once (ALO)	At Least-Once (ALO)Exactly-Once (EO)	The Service Bus generally supports the ALO guarantee; however EO can be supported by using SessionState to store application state and using transactions to atomically receive messages and update the SessionState. The AppFabric workflow uses this technique to provide EO processing guarantees.
Peek Lock	YesVisibility timeout: default=30s; max=2h	YesLock timeout: default=30s; max=5m	Windows Azure queues offer a visibility timeout to be set on each receive operation, while Service Bus lock timeouts are set per entity.
Duplicate Detection	No	Yes, send-side duplicate detection	The Service Bus will remove duplicate messages sent to a queue/topic (based on MessageId).
Transactions	No	Partial	The Service Bus supports local transactions involving a single entity (and its children). Transactions can also include updates to SessionState.
Receive Behavior	Non-blocking, i.e., return immediately if no messages	REST/HTTP: long poll based on user-provided timeout.NET API: 3 options: blocking, blocking with timeout, non-blocking.	
Batch Receive	Yes(explicit)	Yes. Either (a) Implicitly using prefetch, or (b) explicitly using transactions.	
Batch Send	No	Yes (using transactions)	

Receive and Delete	No	Yes	Ability to reduce operation count (and associated cost) in exchange for lowered delivery assurance.
Advanced Features			
Dead lettering	No	Yes	Windows Azure queues offer a 'dequeue count' on each message, so applications can choose to delete troublesome messages themselves.
Session Support	No	Yes	Ability to have logical subgroups within a queue or topic.
Session State	No	Yes	Ability to store arbitrary metadata with sessions. Required for integration with Workflow.
Message Deferral	No	Yes	Ability for a receiver to defer a message until they are prepared to process it. Required for integration with Workflow.
Scheduled Delivery	No	Yes	Allows a message to be scheduled for delivery at some future time.
Security			
Authentication	Windows Azure credentials	ACS roles	ACS allows for three distinct roles: admin, sender and receiver. Windows Azure has a single role with total access, and no ability for delegation.
Management Features			
Get Message Count	Approximate	No	Service Bus queues offer no operational insight at this point, but plan to in the future.
Clear Queue	Yes	No	Convenience functions to clear queue efficiently.
Peek / Browse	Yes	No	Windows Azure queues offer the ability to peek a message without locking it, which can be used to implement browse functionality.
Arbitrary Metadata	Yes	No	Windows Azure queues allow an arbitrary set of <key, value> pairs on queue metadata.
Quotas/Limits			
Maximum message size	8KB	256KB	

Maximum queue size	Unlimited	5GB	Specified at queue creation, with specific values of 1,2,3,4 or 5 GB.
Maximum number of entities per service namespace	n/a	10,000	

8.- APPFABRIC SERVICE BUS TEST CLIENT

En **http://appfabricservicebus.codeplex.com/** podemos encontrar un proyecto, código fuente incluido, que puede ser de gran utilidad para poder entender y probar las colas y topics de Service Bus.

En este apartado se verá un ejemplo de cómo crear un servicio de colas y topics y cómo es posible emplear esta herramienta para poder probar dicha funcionalidad. Al venir con código fuente, podemos emplear esta aplicación para poder entender la forma de utiliza las librerías (Microsoft.ServiceBus.dll) que ofrece el Sdk de Windows Azure para poder trabajar con colas y topics.

```
public List<Message> ReceiveQueueMessages(string queue, ReceiveMode mode)
{
    QueueDescription queueDescription = namespaceManager.GetQueue(queue);
    if (queueDescription != null)
    {
        MessagingFactory factory = MessagingFactory.Create(serviceUri, credentials);
        QueueClient client = factory.CreateQueueClient(queue, mode);
        List<Message> messageList = new List<Message>();

        for (int count = 0; count < queueDescription.MessageCount; count++)
        {
            BrokeredMessage message = client.Receive(TimeSpan.FromSeconds(5));
            if (message != null)
            {
                if (mode == ReceiveMode.PeekLock)
                    message.Complete();
                messageList.Add(GetMessage(message));
            }
        }

        return messageList;
    }
    return null;
}
```

El primer paso es crear un namespace de Windows Azure AppFabric desde el portal de Windows Azure.

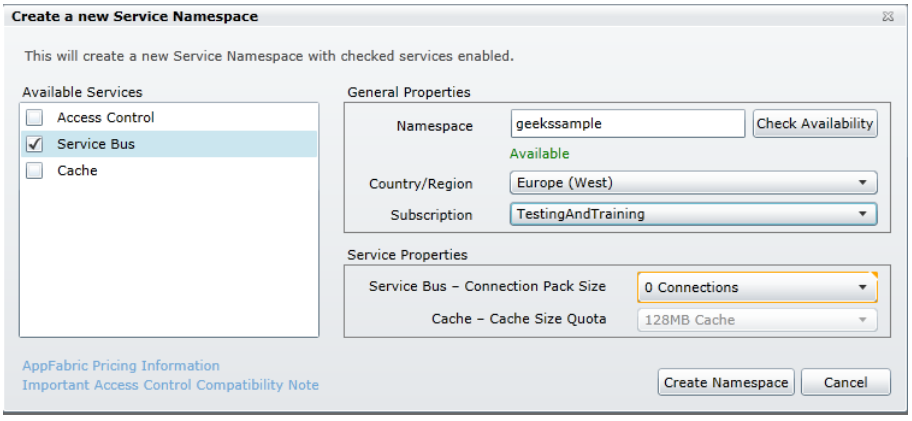


Figura 1.30.- Crear namespace

Una vez creado el namespace se podrá ver desde el portal el listado de todos los namespace disponibles y cómo service bus dispone de las características de colas y topics.

Desde el portal se puede crear las colas, topics y subscripciones, que posteriormente serán las que utilice la aplicación de ejemplo.

Lógicamente, las mismas tareas de aprovisionamiento que se hacen desde el portal podrían hacerse por código empleando los diferentes modelos de programación que permite service bus.

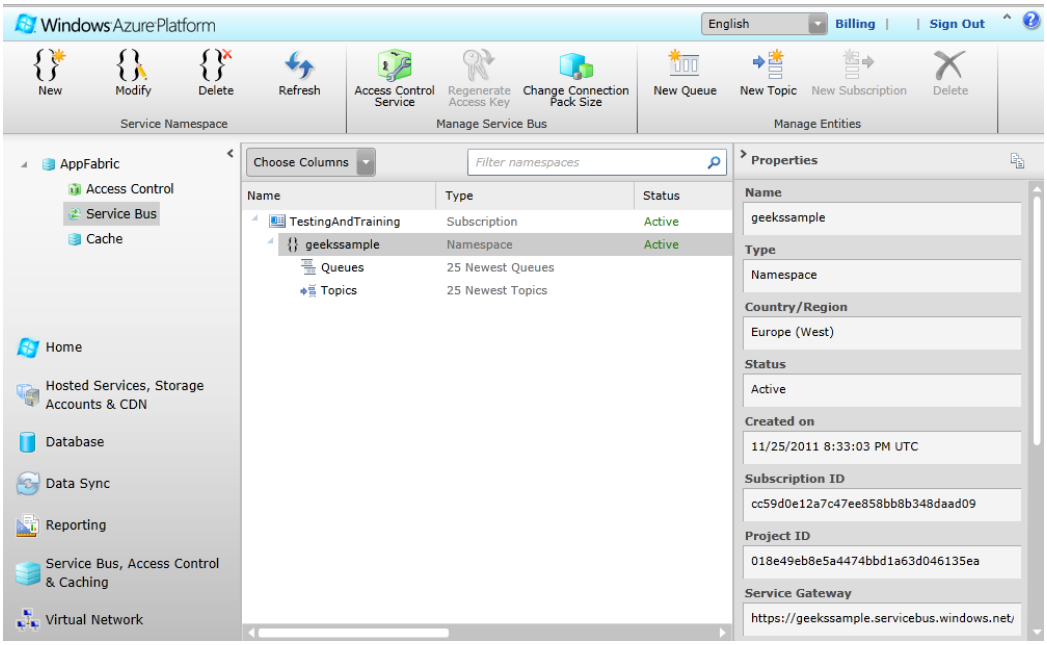


Figura 1.31.- Listado de namespaces

A continuación se muestran las ventanas a través de las cuáles es posible crear colas, topics y subscripciones. Se puede ver en dichas ventanas las opciones de configuración que ofrece cada servicio.

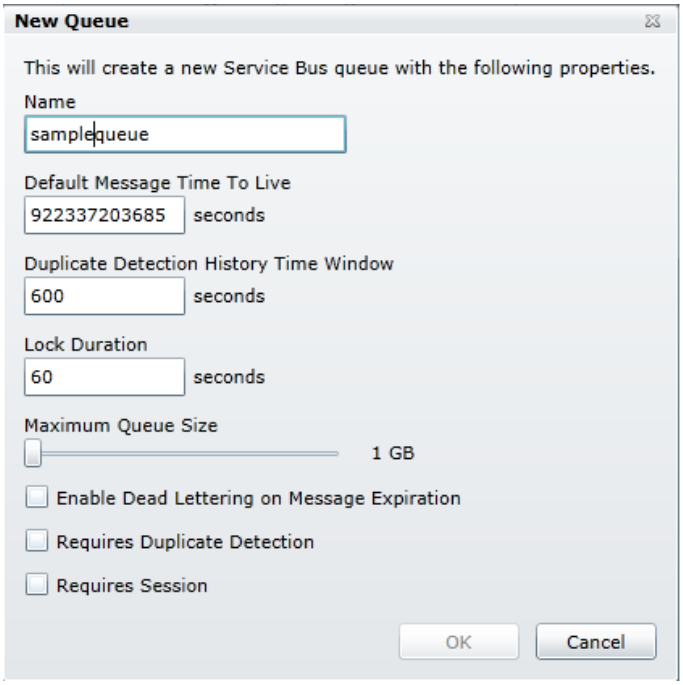


Figura 1.32.- Crear una cola

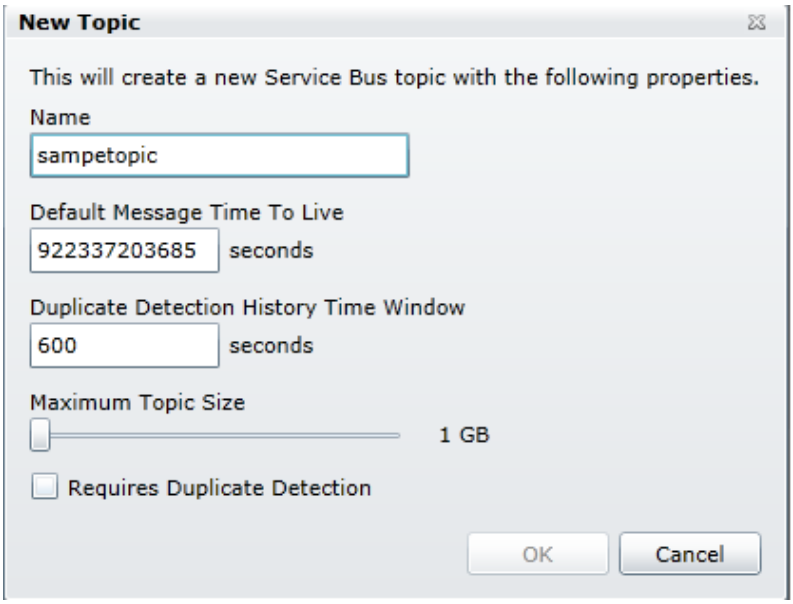


Figura 1.33.- Crear un topic

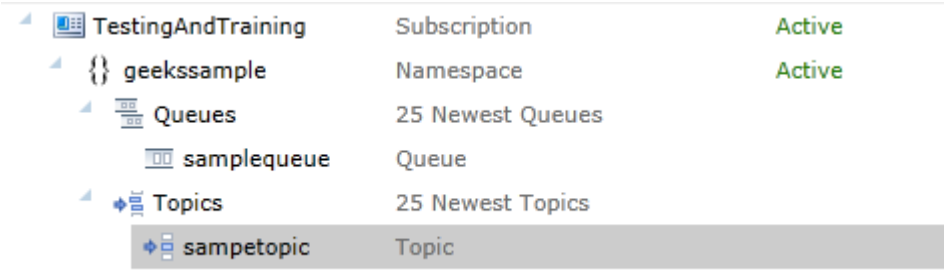


Figura 1.34.- Vista del namespace

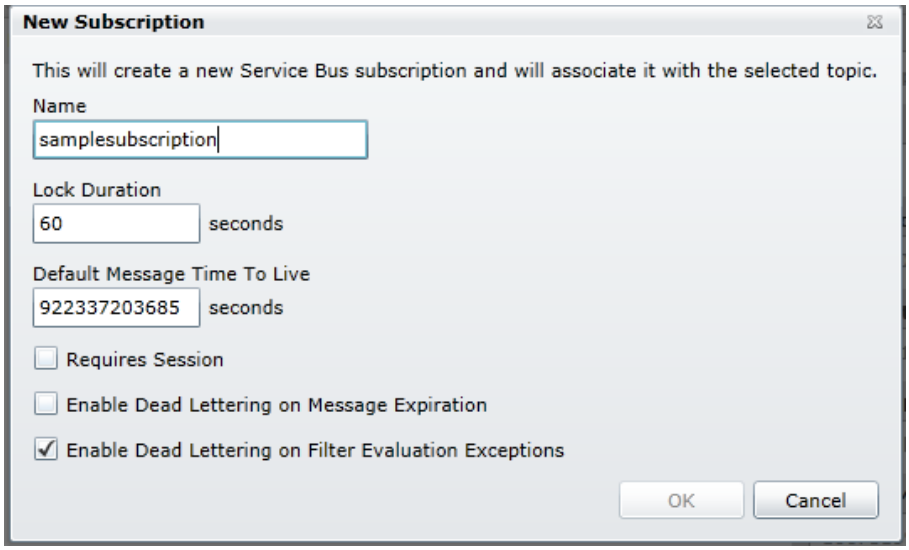


Figura 1.35.- Crear una subscripción

Una vez creado el servicio, éste ya puede ser consumido desde una aplicación cliente. En este caso se usará la herramienta cliente que se ha mencionado anteriormente.

Una vez descargado el código fuente se podrán ver que existe dos soluciones de Visual Studio, una para la aplicación cliente y otra para la aplicación servidora, las cuáles pueden ser usadas para probar toda la funcionalidad comentada anteriormente.

Desde la aplicación cliente tendremos que realizar una conexión al servicio de Service Bus, para el cuál será necesario indicar el namespace del servicio y las credenciales de acceso; el nombre y la clave.

En este caso cabe recordar que toda la autenticación se realiza a través del servicio de Access Control.

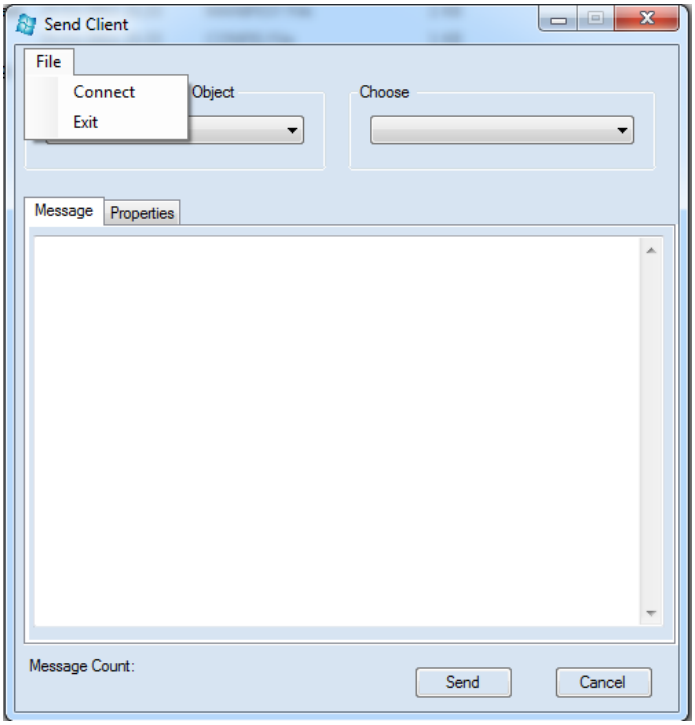


Figura 1.36.- Conexión al service Bus

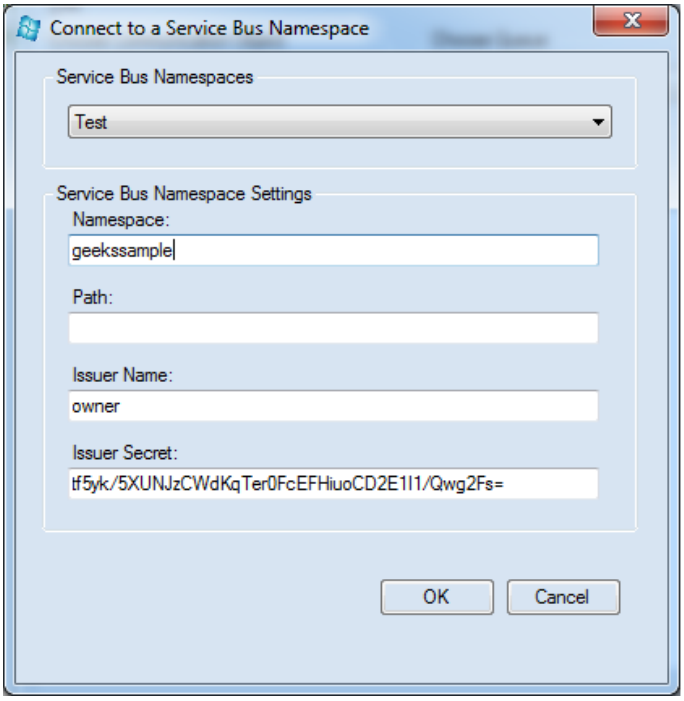


Figura 1.37.- Conexión al Service Bus

Una vez conectados al servicio se podrán ver todas las colas, topics y subscripciones creados anteriormente y simular el envío de uno o varios mensajes.

Hay que destacar que para probar este servicio no es necesario disponer del servidor arrancado, ya que la comunicación ocurre de manera asíncrona y que todos los mensajes son persistidos en el Service Bus hasta que el servidor los lea y elimine.

Para el servidor habría que realizar el mismo proceso; abrir la aplicación, ejecutarla, conectarse al service bus e indicar de qué cola o subscripción se desea hacer la lectura.

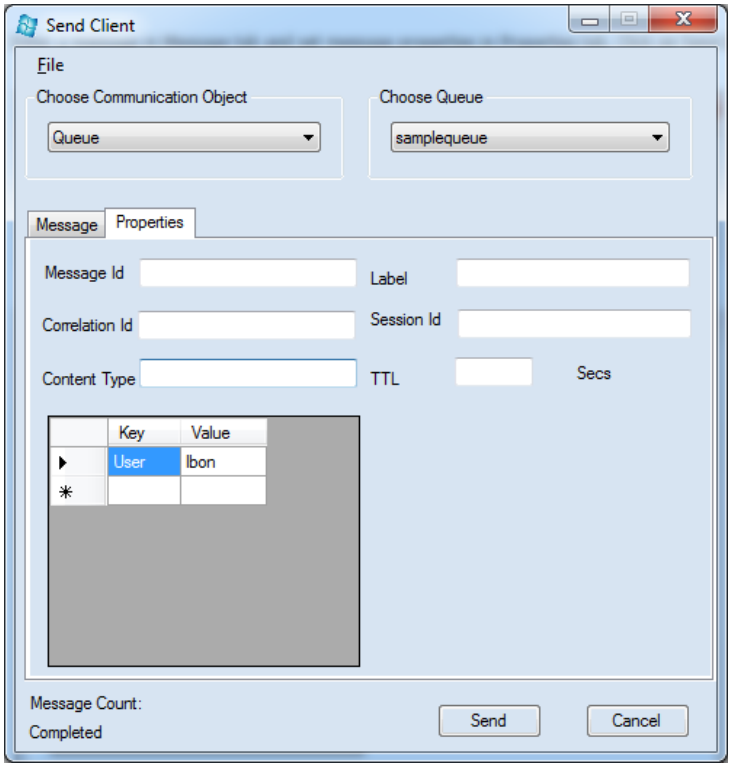


Figura 1.38.- Envío de mensajes

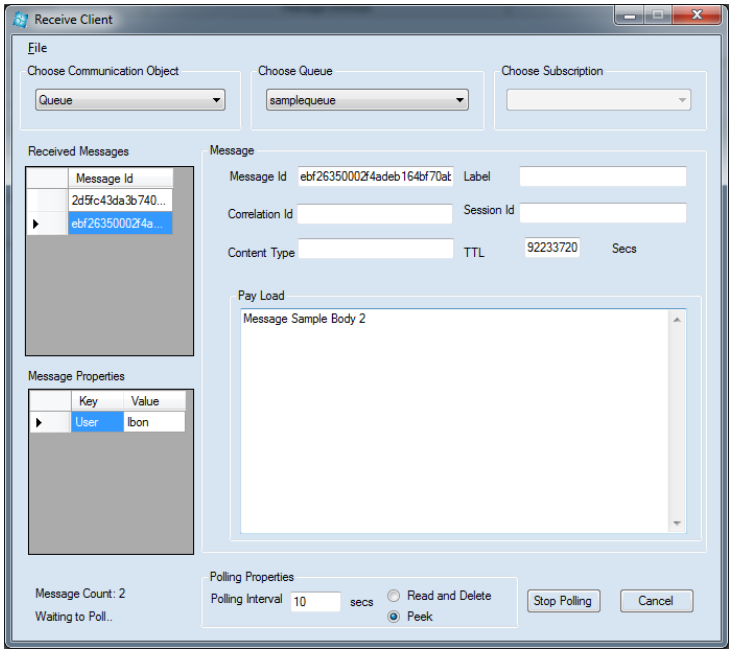


Figura 1.39.- Modelo de programación

9.- TIPOS DE BINDINGS WCF EN SERVICE BUS

AppFabric Service Bus ofrece múltiples bindings que pueden emplearse para utilizar el servicio de relay, aunque los principales son los bindings de relay para realizar las comunicaciones por TCP (NetTcpRelayBinding), los bindings WS (WSHttpRelayBinding), los bindings unidireccionales (NetOnewayRelayBinding) y los bindings de eventos (NetEventRelayBinding).

A continuación se muestra el fichero de configuración de un servicio construido con WCF y que emplea NetTcpRelayBinding. Es importante destacar el contenido de la propiedad address del endpoint, puesto que este, como observará, es el que contiene la dirección del servicio de relay. Fíjese además como el prefijo de la dirección no es net.tcp sino sb.

```
<system.serviceModel>
    <services>
      <service name="Service.MyService" >
        <endpoint address="sb://mynamespace.servicebus.windows.net/MyService"
                  behaviorConfiguration ="sharedSecretClientCredentials"
                  binding="netTcpRelayBinding"
                  contract="Service.IContract" />
      </service>
    </services>
    <behaviors>
      <endpointBehaviors >
        <behavior name ="sharedSecretClientCredentials">
          <transportClientEndpointBehavior credentialType="SharedSecret">
            <clientCredentials>
              <sharedSecret issuerName="owner"
                            issuerSecret="Hzw55XAm/jhTiLwHs8dwukK7LFBEmEqDuGJQ/tL+I84="/>
            </clientCredentials>
          </transportClientEndpointBehavior>
        </behavior>
      </endpointBehaviors>
    </behaviors>
</system.serviceModel>
```

El binding TCP permite configurar la comunicación de tres maneras diferente; utilizando el servicio de relay, utilizando la comunicación directa o el modo híbrido. En el primer caso todas las comunicaciones se realizan a través del sistema de relay. En el segundo caso las comunicación se realizan de forma directa (sólo la primera comunicación se realiza por el servicio de relay), mientras que la tercera opción es una mezcla de las dos anteriores; siempre que se puede se realizará la conexión de forma directa y cuándo no se puede se realizará a través del servicio de relay.

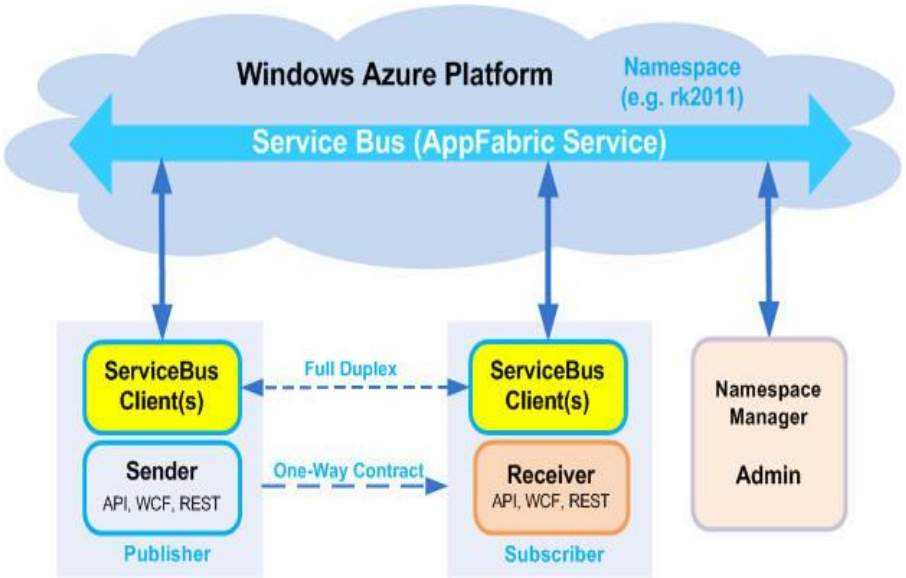


Figura 1.40.- Comunicación WCF

A la configuración anterior se le puede añadir la siguiente sección con la propiedad connectionMode indicando el modo de funcionamiento deseado.

```
<bindings>
      <!-- Application Binding -->
      <netTcpRelayBinding>
        <binding name="lab" connectionMode="Hybrid">
          <security mode="None" />
        </binding>
      </netTcpRelayBinding>
</bindings>
```

El binding WSHttpRelayBinding permite enviar y recibir mensajes interoperables a través del protocolo HTTP p HTTPS, es decir, mensajes utilizando los estándares de WS-*. Este tipo de binding sólo soporta la comunicación a través de servicio de relay no pudiendo hacer una comunicación directa o hibrida.

Con NetOnewayRelayBinding el funcionamiento es algo diferente. El cliente envía los mensajes al servidor, en lugar de al servicio, y éste los almacena en un buffer de datos. El servicio de relay, posteriormente, encargará de intentar hacer llegar los datos al servicio. Todos los mensajes son siempre unidireccionales y para poder utilizar este tipo de binding se comprueba que todas las operaciones del servicio estén marcadas como "One-Way". Esto, tiene sus implicaciones, porque en realidad nos aportará un grado de escalabilidad al no tener que esperar a que las operaciones se realicen, estamos incluyendo el soporte para asincronía en nuestros comandos.

NetEventRelayBinding es una especialización clara pero fundamental del binding unidireccional NetOnewayRelayBinding. En esencia el funcionamiento es el mismo, pero permite que haya varios servicios que puedan estar escuchando a la vez en la misma URI, seguramente a muchos les sonará esto a algo como Pub/Sub, de esta manera, un cliente puede enviar un mensaje al servicio de relay y ser posteriormente múltiples servicios los que reciban el mensaje.

9.1.- ¿Qué binding debo elegir?

AppFabric Service Bus, como acabamos de ver, ofrece múltiples bindings que pueden emplearse para utilizar el servicio de relay, aunque los principales son los bindings de relay para realizar las comunicaciones por TCP (NetTcpRelayBinding), los bindings WS (WSHttpRelayBinding), los bindings unidireccionales (NetOnewayRelayBinding) y los bindings de eventos (NetEventRelayBinding).

9.1.1.- NetTcpRelayBinding

NetTcpRelayBinding es seguramente la opción más habitual que se empleará para realizar la comunicación a través del servicio de relay. Es la alternativa que mejor rendimiento ofrece.

Este tipo de binding permite comunicaciones bidireccionales, unidireccionales e incluso permite comunicaciones duplex con callbacks, todo ello a través del servicio de relay.

Para poder utilizar este bindings es necesario poder realizar la comunicación a través del puerto 808 TCP (o 828 si se emplea seguridad de transporte). NetTcpRelayBinding no tiene un límite máximo en el tamaño de los mensajes y es capaz de mantener la sesión, asegurando que las comunicaciones que se realizan a través del mismo proxy se realizan contra el mismo servicio.

El binding TCP permite además, configurar la comunicación de tres maneras diferente; utilizando el servicio de relay, utilizando la comunicación directa o el modo híbrido. En el primer caso todas las comunicaciones se realizan a través del sistema de relay. En el segundo casi las comunicación se realizan de forma directa (sólo la primera comunicación se realiza por el servicio de relay), mientras que la tercera opción es una mezcla de las dos anteriores; siempre que se puede se realizará la conexión de forma directa y cuándo no se puede se realizará a través del servicio de relay.

Como aspecto negativo, decir que este tipo de binding no es interoperable.

9.1.2.- WSHttpRelayBinding

WSHttpRelayBinding permite enviar y recibir mensajes interoperables a través del protocolo HTTP p HTTPS.

Este tipo de binding es similar a NetTcpRelayBinding salvo por el hecho de que sólo soporta la comunicación a través de servicio de relay.

Este tipo de binding puede ser una buena alternativa si existe un requisito de interoperabilidad o si el puerto TCP que necesita el binding TCP no se encuentra disponible.

9.1.3.- NetOneWayRelayBinding

Con NetOnewayRelayBinding el cliente envía los mensajes al servidor, en lugar de al servicio, y éste los almacena en un buffer de datos. El servicio de relay, posteriormente, se encarga de intentar hacer llegar los datos al servicio. El servicio nunca puede responder.

Todos los mensajes son siempre unidireccionales y para poder utilizar este tipo de binding se comprueba que todas las operaciones del servicio estén marcadas como "One-Way".

Existe un máximo para del 64 kb para el tamaño de los mensajes que se envían al servicio de relay.

En este caso, el cliente nunca tiene la seguridad de que el mensaje ha llegado al destinatario, incluso puede darse el caso de que el servicio envíe mensajes a un servicio que no existe.

Este tipo de binding podría ser útiles cuando se desea realizan una comunicación entre un cliente y un servicio y no depender del estado del servicio, por ejemplo, en entornos desconectados. El servicio no tiene por qué estar disponible pero en el momento en el que lo esté recibirá los mensajes almacenados en el buffer.

9.1.4.- NetEventRelayBinding

NetEventRelayBinding es una especialización clara pero fundamental del binding unidireccional NetOnewayRelayBinding.

En esencia el funcionamiento es el mismo, pero permite que haya varios servicios que puedan estar escuchando a la vez en la misma URI. De esta manera, un cliente puede enviar un mensaje al servicio de relay y ser posteriormente múltiples servicios los que reciban el mensaje. El sistema de relay no asegura el orden de los mensajes.

Este tipo de binding puede ser útil para implementar un mecanismo de comunicación basado en el patrón publicador-subscriptor.

9.2.- Autenticación y autorización con Access Control

Tal y como ya comentamos anteriormente, el servicio de Relay requiere tener a los dos extremos autenticados y autorizados, este proceso, como cualquiera dentro de Azure pasa por el servicio de control de acceso, App Fabric Access Control Service.

Existen cuatro tipos de autenticación disponibles en la actualidad:

- **SharedSecret**, basado en un sistema de usuario/contraseña.
- **Saml**, que permite interactuar con los sistemas de autenticación SAML 2.0.
- **SimpleWebToken**, sistema basado en el protocolo WRAP (Web Resource Application Protocol) y Simple Web Tokens (SWT).
- **No autenticado**.

Este podría ser el fichero de configuración de un servicio que emplea Service Bus y SharedSecret como sistema de seguridad:

```
<system.serviceModel>
  <services>
    <service name="Service.MyService" >
      <endpoint address="sb://mynamespace.servicebus.windows.net/MyService"
                behaviorConfiguration ="sharedSecretClientCredentials"
                binding="netTcpRelayBinding"
                contract="Service.IContract" />
    </service>
  </services>
  <behaviors>
    <endpointBehaviors >
      <behavior name ="sharedSecretClientCredentials">
        <transportClientEndpointBehavior credentialType="SharedSecret">
          <clientCredentials>
            <sharedSecret issuerName="owner"
                          issuerSecret="Hzw55XAm/jhTiLwHs8dwukK7LFBEmEqDuGJQ/tL+I84="/>
          </clientCredentials>
        </transportClientEndpointBehavior>
      </behavior>
    </endpointBehaviors>
  </behaviors>
</system.serviceModel>
```

Adicionalmente a la seguridad ofrecida por Access Control, la propia aplicación servidora puede realizar e implementar su propio sistema de seguridad, del mismo modo que se puede llegar a securizar una aplicación WCF; seguridad de transporte, seguridad de mensaje etc...

Todos los bindings disponibles para utilizarse con el servicio de relay disponen de una opción de seguridad dónde pueden configurarse las diferentes opciones disponibles.

```
<bindings>
      <ws2007HttpRelayBinding>
        <binding name="lab">
          <security mode="Transport" />
        </binding>
      </ws2007HttpRelayBinding>
</bindings>
```

10.- BUFFERS DE MENSAJES

Un buffer de mensajes puede describirse como una caché temporal dónde los mensajes pueden almacenarse por un periodo corto de tiempo, hasta que éstos son leídos.

Los buffers de mensajes son especialmente útiles cuando no es posible utilizar algunos de los endpoints disponibles en Service Bus para comunicar un cliente con un servidor; por ejemplo, cuando una de las partes de la comunicación no se encuentra un sistema operativo Windows o está implementado en otra tecnología como puede ser Java.

El buffer de mensajes puede ser accedido desde cualquier tecnología empleando el protocolo HTTP y no requiere disponer del SDK de Windows Azure. El protocolo de comunicación es REST y por lo tanto, utilizando los distintos verbos HTTP podremos de una forma sencilla tratar los mensajes, crearlos, enviarlos etc…utilizando cualquier tecnología y dispositivo capaz de hacer una petición HTTP.

El protocolo de comunicación con el buffer de mensajes también emplea Access Control para realizar el proceso de autenticación y autorización, usando un Simple Web Token (SWT) como mecanismo de seguridad. El protocolo permite out of box los verbos POST/PUT, PUT, DELETE y GET.

A continuación se muestra un sencillo ejemplo de cómo emplear el protocolo HTTP REST para crear un buffer de mensajes y enviar y recibir mensajes de él.

```
// Prompt the user for the service namespace and issuer key.
Console.Write("Please enter your Service Namespace: ");
string serviceNamespace = Console.ReadLine();
Console.Write("Please enter the key for the 'owner' issuer: ");
```

```
   string ownerKey = Console.ReadLine();
   // Create a GUID for the buffer name.
   string bufferName = Guid.NewGuid().ToString("N");

   // Construct the message buffer URI.
   string messageBufferLocation =
            string.Format("http://{0}.servicebus.windows.net/{1}", serviceNamespace,
bufferName);

   // Get the AC token
   WebClient client = new WebClient();
   client.BaseAddress = string.Format("https://{0}-sb.accesscontrol.windows.net/",
serviceNamespace);
   NameValueCollection values = new NameValueCollection();
   values.Add("wrap_name", "owner");
   values.Add("wrap_password", ownerKey);
   values.Add("wrap_scope", messageBufferLocation);
   byte[] responseBytes = client.UploadValues("WRAPv0.9", "POST", values);
   string response = Encoding.UTF8.GetString(responseBytes);

   string token = Uri.UnescapeDataString(response.Split('&').Single(value =>
value.StartsWith("wrap_access_token=",
       StringComparison.OrdinalIgnoreCase)).Split('=')[1]);

   // Create the auth header from the token
   string authHeaderValue = string.Format("WRAP access_token=\"{0}\"", token);

   // Create the message buffer policy.
   string policy =
       "<entry xmlns=\"http://www.w3.org/2005/Atom\">" +
       "<content type=\"text/xml\">" +
       "<MessageBufferPolicy
xmlns=\"http://schemas.microsoft.com/netservices/2009/05/servicebus/connect\"/>" +
       "</content>" +
       "</entry>";

   // Create a message buffer.
   client.BaseAddress = string.Format("https://{0}.servicebus.windows.net/{1}/",
serviceNamespace, bufferName);
   client.Headers[HttpRequestHeader.ContentType] =
"application/atom+xml;type=entry;charset=utf-8";
   client.Headers[HttpRequestHeader.Authorization] = authHeaderValue;
   client.UploadData(String.Empty, "PUT", Encoding.UTF8.GetBytes(policy));
   Console.WriteLine("Message buffer was created at '{0}'.", messageBufferLocation);

   // Send a message to the message buffer.
   client.Headers[HttpRequestHeader.ContentType] = "text/xml";
   client.Headers[HttpRequestHeader.Authorization] = authHeaderValue;
   client.UploadData("messages?timeout=20", "POST", Encoding.UTF8.GetBytes("<msg1>This is
message #1</msg1>"));
   Console.WriteLine("Message sent.");

   // Retrieve message.
   client.Headers[HttpRequestHeader.Authorization] = authHeaderValue;
   string payload = Encoding.UTF8.GetString(client.UploadData("messages/head?timeout=20",
"DELETE", new byte[0]));
   Console.WriteLine("Retrieved the message '{0}'.", payload);

   // Delete the message buffer.
   client.Headers[HttpRequestHeader.Authorization] = authHeaderValue;
   client.UploadData(String.Empty, "DELETE", new byte[0]);
   Console.WriteLine("Message buffer at '{0}' was deleted.", messageBufferLocation);
```

Aunque no es necesario disponer del SDK de Windows Azure, éste también provee una serie de funcionalidades que pueden ayudar a simplificar el trabajo con los buffers de mensajes, si estamos en plataforma .NET. Por lo que, si usted está en este caso no dude en descargar, instalar y probar este SDK.

En el siguiente ejemplo se muestra como configurar y crear un buffer de mensajes y cómo poder enviar y recibir mensaje de él, todo empleando el SDK de Windows Azure.

```
    string serviceNamespace = "...";
    MessageVersion messageVersion = MessageVersion.Soap12WSAddressing10;
    string messageAction = "urn:Message";

    // Configure credentials.
    TransportClientEndpointBehavior behavior = new TransportClientEndpointBehavior();
    behavior.CredentialType = TransportClientCredentialType.SharedSecret;
    behavior.Credentials.SharedSecret.IssuerName = "...";
    behavior.Credentials.SharedSecret.IssuerSecret = "...";

    // Configure buffer policy.
    MessageBufferPolicy policy = new MessageBufferPolicy
    {
        ExpiresAfter = TimeSpan.FromMinutes(2.0d),
        MaxMessageCount = 100
    };

    // Create message buffer.
    string bufferName = "MyBuffer";
    Uri bufferLocation =
            new Uri("https://" + serviceNamespace + ".servicebus.windows.net/services/" +
bufferName);
    MessageBufferClient client = MessageBufferClient.
                                 CreateMessageBuffer(behavior, bufferLocation, policy,
messageVersion);

    // Send 10 messages.
    for (int i = 0; i < 10; ++i)
    {
        client.Send(Message.CreateMessage(messageVersion, messageAction, "Message #" + i));
    }

    Message message;
    string content;

    // Retrieve a message (destructive read).
    message = client.Retrieve();
    content = message.GetBody<string>();
    message.Close();

    Console.WriteLine("Retrieve message content: {0}", content);

    // Retrieve a message (peek/lock).
    message = client.PeekLock();
    content = message.GetBody<string>();

    Console.WriteLine("PeekLock message content: {0}", content);

    // Delete previously locked message.
    client.DeleteLockedMessage(message);
    message.Close();

    // If no more messages are retrieved within the ExpiresAfter time span,
    // the buffer will automatically be deleted...
```

11.- APPFABRIC CACHING

Windows Azure AppFabric Caching es un sistema de caché distribuida en memoria que se ofrece como un servicio en la nube.

Un servicio similar ya existía previamente para soluciones on-premise, integrado dentro de Windows Server AppFabric.

Ahora se disponen de las mismas capacidades y características para aplicaciones que estén en la nube. Se ofrece como un servicio en la nube, por lo que como se verá a continuación nos es necesario tener que hacer nada relacionado con las tareas de instalación, configuración o administración. Simplemente hacer uso del servicio.

Para crear un nuevo servicio de cache es necesario entrar en el portal de administración de Windows Azure y seleccionar la opción de crear un nuevo namespace con esta característica activada.

En el momento de la creación se debe elegir la ubicación del sistema y el tamaño que tendrá.

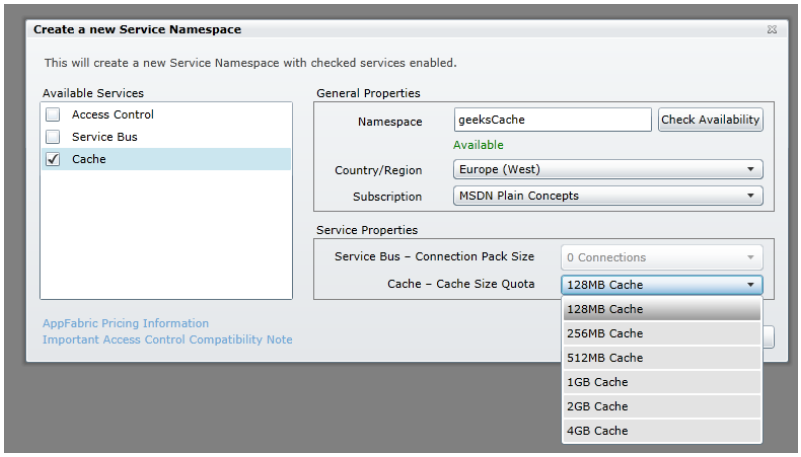


Figura 1.41.- Crear un servicio de cache

Seleccionando el servicio de caché se mostrarán las propiedades dónde se encuentra toda la información que se necesita para poder hacer uso de este servicio desde las aplicaciones; URL de la caché, puerto, token de seguridad (es posible integrar la seguridad con Access Control) y la información que se necesita tener para poder integrar la aplicación.

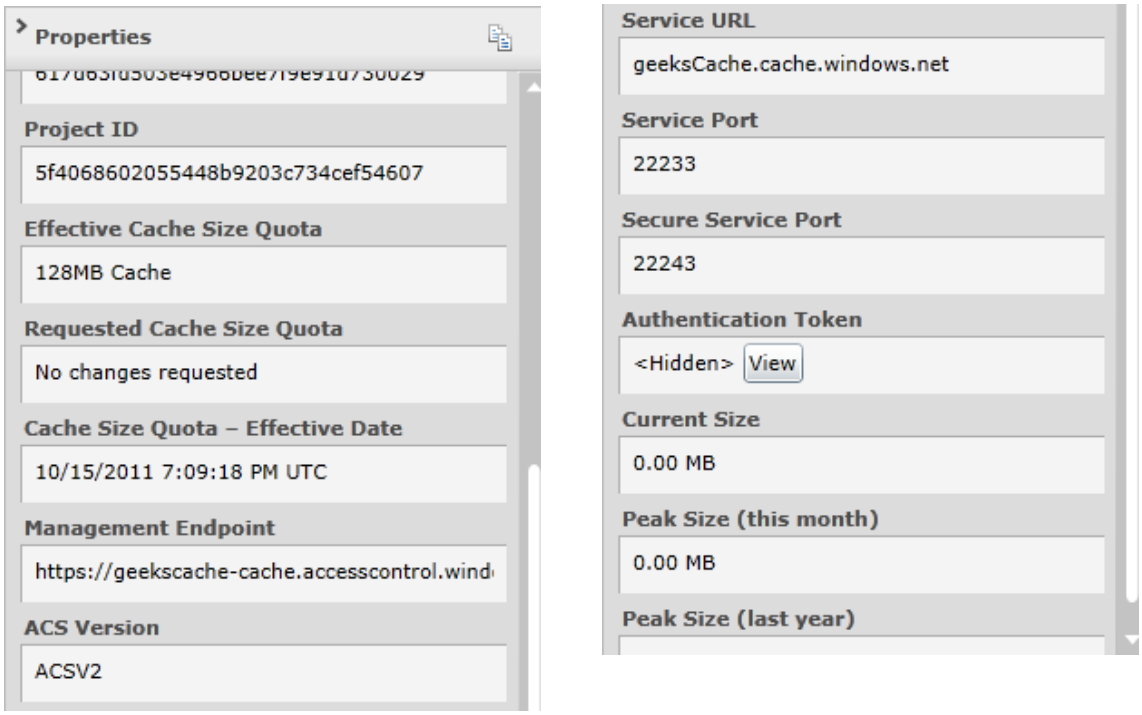


Figura 1.42.- Información del sistema de caché

En la sección de integración se ofrece información muy interesante para poder hacer que las aplicaciones hagan uso de este caché, para que prácticamente sea copiar ciertos valores en el fichero de configuración de la aplicación.

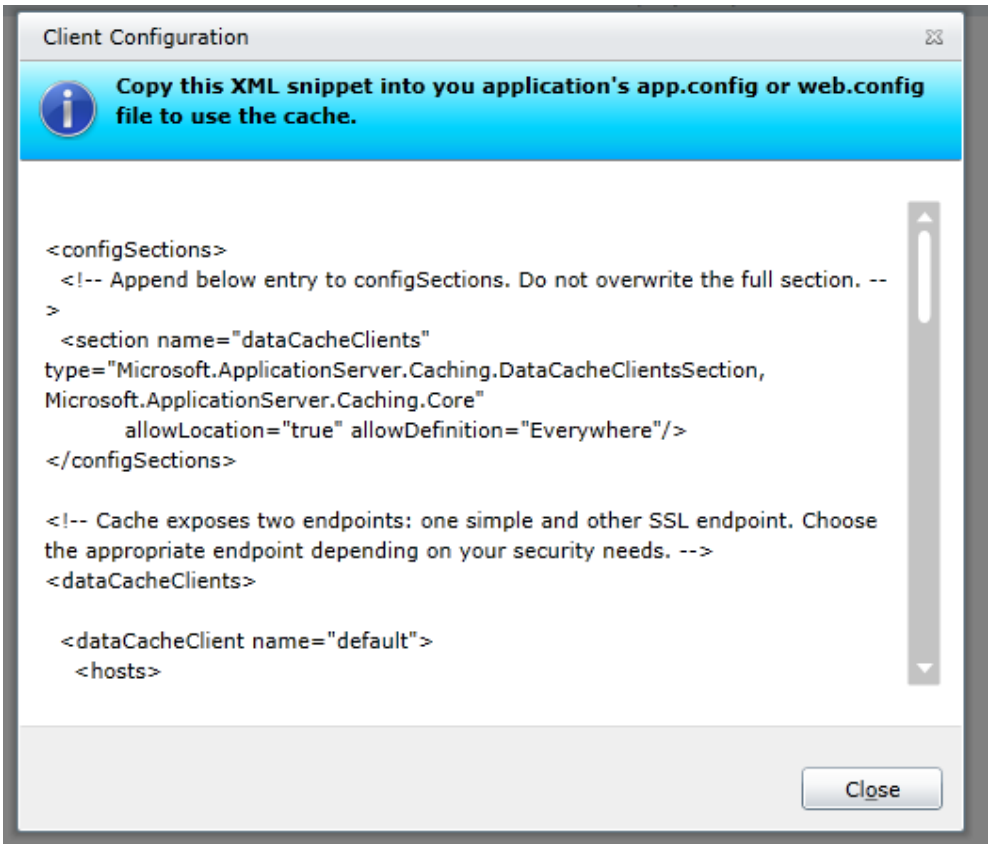


Figura 1.43.- Información de integración

```
   <configSections>

     <!-- Append below entry to configSections. Do not overwrite the full section. -->

     <section name="dataCacheClient"
type="Microsoft.ApplicationServer.Caching.DataCacheClientSection,
Microsoft.ApplicationServer.Caching.Core"

          allowLocation="true" allowDefinition="Everywhere"/>

   </configSections>

   <dataCacheClient deployment="Simple">

     <hosts>

       <host name="ibongeeks.cache.appfabriclabs.com" cachePort="22233" />

     </hosts>

     <securityProperties mode="Message">

       <messageSecurity

        authorizationInfo="XXX"
       </messageSecurity>

     </securityProperties>

   </dataCacheClient>

   <!-- If session state needs to be saved in AppFabric Caching service add the following to
web.config
   inside system.web -->
```

```
     <sessionState mode="Custom" customProvider="AppFabricCacheSessionStoreProvider">

       <providers>

           <add name="AppFabricCacheSessionStoreProvider"


type="Microsoft.Web.DistributedCache.DistributedCacheSessionStateStoreProvider,
Microsoft.Web.DistributedCache"

                cacheName="default"

                useBlobMode="false" />

        </providers>

     </sessionState>

   <!-- If output cache content needs to be saved in AppFabric Caching service add the
following to
   web.config inside system.web -->

     <caching>

       <outputCache defaultProvider="DistributedCache">

           <providers>

             <add name="DistributedCache"

               type="Microsoft.Web.DistributedCache.DistributedCacheOutputCacheProvider,
Microsoft.Web.DistributedCache"

               cacheName="default" />

           </providers>

         </outputCache>

       </caching>
```

Por ejemplo, hacer que una aplicación haga uso de este sistema para que la información de sesión de la aplicación se haga uso de esta caché será tan sencillo como hacer caso a la información de integración y hacer un "copy-paste" de la información que te dice que tienes que copiar en tu aplicación.

CAPÍTULO 2

Windows Identity Foundation

El objetivo de este capítulo es intentar mostrar las bondades de Windows Identity Foundation y cómo es posible su uso dentro de desarrollos.NET.

El mundo de la seguridad de aplicaciones en sin lugar a dudas un mundo complejo al que hay que dedicar tiempo, pero en muchas ocasiones ya sea por desconocimiento o por dejadez, los desarrolladores no se preocupan como deberían del sistema de seguridad o aportan soluciones rápidas e incorrectas para salir del paso.

A lo largo de este capítulo se intentará mostrar como realmente el trabajo con WIF es sencillo y que a los desarrolladores, lo único que les va a aportar es facilidad y sencillez de trabajo.

1.- ¿POR QUÉ WIF?

Una de los primeras cosas que es necesario preguntarse, la primera, es la motivación de WIF, ¿qué aporta a los desarrolladores? ¿Por qué se necesita esta tecnología? ¿Qué problemas resuelve?

Sin lugar a dudas, estas deberían ser las primeras preguntas que cualquier desarrollador debería hacerse, tanto para esta, como para cualquier otra tecnología.

Las respuestas pueden ser variadas, pero en los puntos siguientes se intentarán aportar algunas respuestas:

1.1.- Experiencia en seguridad.

La seguridad suele ser una tarea sobre la que se suele pasa de puntillas, incluyendo aquí problemas de criptografía, problemas con certificados, ataques etc…

Generalmente los problemas de seguridad que se tenían que resolver se podían atajar directamente con un usuario y contraseña, sin embargo, en este mundo cada vez más global, elementos como los tokens de seguridad, la autenticación en proveedores terceros (token facebook, twitter, partners, open id, oauth…) son cada vez más comunes. El manejo de estos elementos, la mezcla con los nuestros, y otros patrones de seguridad hacen que se necesiten en muchos casos muchos conocimientos de seguridad, cada vez más.

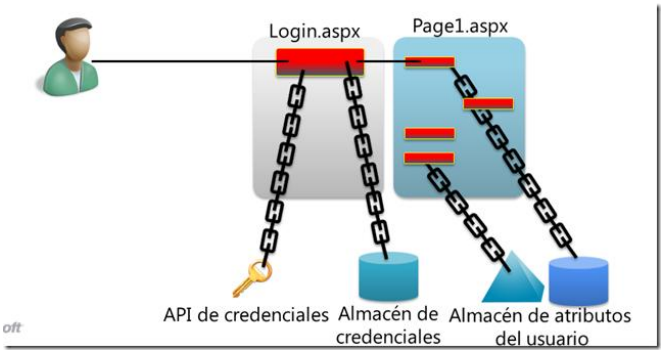


Figura 2.1.- Esquema de arquitectura WIF

1.2.- La identidad es ya un problema complejo

Como se acaba de comentar, los escenarios de seguridad son cada vez más complejos, las aplicaciones ya no se acceden solamente desde la intranet o internet, por lo tanto los repositorios de usuarios tienen que ser capaces de manejar diferentes tipos de proveedores (Active Directory, custom, Google, Yahoo, Facebook etc….), a mayores, escenarios como la federación de partners o la delegación vienen a completar el conjunto de escenarios de seguridad que es necesario que cubrir por los desarrolladores.

Además de esto, hoy en día la identidad ya no solamente viene representada por un par usuario/contraseña, la identidad, tanto de un usuario como de una máquina, puede venir representada por diferentes atribuciones.

1.3.- Interoperabilidad

Los mecanismos de seguridad (autenticación / autorización) deberían de ser interoperables con otras tecnologías, ya que generalmente no se puede dar por hecho que las aplicaciones vivirán siempre independientemente del resto de aplicaciones.

> **Nota**: Aunque la figura anterior hace referencia al trabajo en la web, realmente este mismo criterio se aplicaría a un servicio WCF..

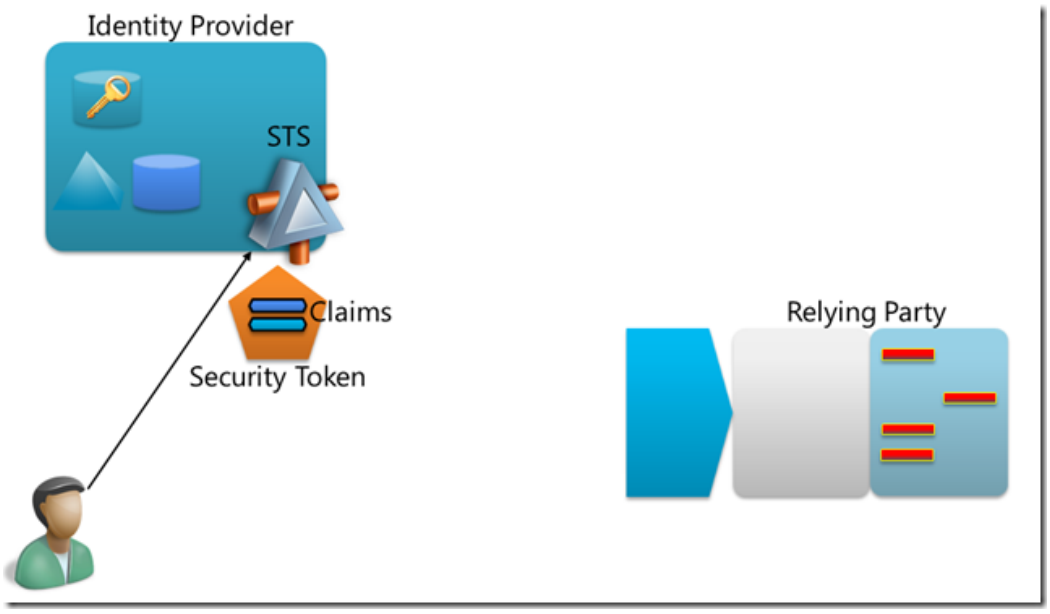


Figura 2.2.- WIF

Una posible solución a este problema es externalizar el proceso de seguridad de las aplicaciones, hacer un "outsourcing de la autenticación" delegando en un "experto" los distintos problemas y escenarios que se puedan plantear.

WIF y el modelo de seguridad basado en "claims" permiten realizar esta tarea de una forma realmente sencilla.

Lógicamente este modelo implica una serie de conceptos y definiciones se tratarán de explicar en los siguientes apartados.

2.- LOS CONCEPTOS BÁSICOS

A lo largo de esta segunda apartado, se tratará de poner de manifiesto algunos conceptos que es necesario entender y comprender, así como los elementos de trabajo que tendrá que conocer una aplicación que quiera hacer uso de WIF.

Dentro del escenario planteado en el punto anterior podemos observar conceptos como los siguientes:

2.1.- Security Token Service

Un Security Token Service, más comúnmente llamado STS, se encarga de entregar a la aplicación, una vez realizado el proceso de autenticación, un token d seguridad (security token).

Para realizar esta tarea, se dispone de un protocolo estándar que tanto .NET como otras tecnologías pueden implementar. Este protocolo, se conoce como **WS-Trust**.

Windows Identity Foundation implementa este protocolo por medio de un servicio WCF cuyo contrato, IWSTrust13SyncContract, está definido en Microsoft.IdentityModel.Protocols.WSTrust.

Realmente WS-Trust es un protocolo que funciona con paquetes SOAP por lo aquellos elementos que quieran interactuar con él para, emitir, renovar o cancelar un token tendrán que entender SOAP. A estos clientes, generalmente se les conoce como Clientes Activos.

Lógicamente, hay escenarios dónde los clientes que necesiten un token de seguridad no entienden SOAP, por ejemplo un navegador web y HTML. Para estos escenarios, hay un conjunto de recetas que están disponibles para solventar el problema. Este conjunto de recetas se conoce como **WS-Federation**.

2.2.- Security Token

Se acaba de hablar del encargado de emitir, renovar o cancelar tokens de seguridad así como de los protocolos que usa pero ¿qué es un token de seguridad?

Un token de seguridad no es más que un conjunto de claims (aserciones, afirmaciones) que representan atributos de una entidad, por ejemplo el nombre, la edad, el correo electrónico, cualquier cosa en definitiva.

Lógicamente existe un conjunto de claims comunes como name, role, action etc., pero el desarrollador puede definir nuestras propias claims.

Como es de esperar, este contenido de claims está securizado por medio de una firma digital que garantice que en el proceso de emisión y recepción nadie pueda tocar este elemento. La definición de este elemento se hace por medio de un lenguaje XML conocido como **SAML (Security Assertion Markup Language).**

2.3.- Relaying Party

Una Relying Party no es más que aquella aplicación o servicio que tiene externalizado el proceso de autenticación en un Security Token Service. Podría ser cualquier aplicación ASP.NET o Servicio WCF que delega su proceso de autenticación en un STS con un proveedor de identidad determinado.

3.- EJEMPLO DE USO

A lo largo de esta tercera apartado se intentarán mostrar ejemplos de uso que permitan ver cómo integrar WIF dentro de las aplicaciones y demostrar cómo este proceso no tiene por qué ser traumático y en cambio cómo el sistema ofrece muchas ventajas con respecto a sistemas más tradicionales.

3.1.- Cliente WCF Activo

En este primer ejemplo se intentará mostrar como incluir un STS dentro de un servicio expuesto por Windows Communication Foundation y cómo hacer uso de él. Posteriormente se intentará modificar la configuración/código para obtener nuevas funcionalidades o cambiar la parametrización de funcionamiento.

3.1.1.- Definición del servicio

El primer paso a realizar será el de disponer de un servicio de WCF, en este caso, usaremos un sencillo ejemplo de un servicio definido tal y como sigue en los siguientes fragmentos de código. Este servicio, por comodidad se ha expuesto en una aplicación de consola, pero lo mismo aplica si fuera un servicio de Windows o bien un sitio web.

```
[ServiceContract(Name="ICrmManager",Namespace="http://www.plainconcepts.com/wif/samples")]
    public interface ICRMManager
    {
        [OperationContract()]
        void Add(Customer customer);

        [OperationContract()]
        IList<Customer> Find();
    }
```

3.1.2.- Utilizando Federation Utility

Una vez que se tiene el contrato implementado y el servicio configurado (todo lo relativo a hosting y configuración de WCF es algo que se da por sentado), se puede hacer uso de las herramientas de Windows Identity Foundation de una forma muy sencilla.

En el menú contextual del proyecto, se puede ver una entrada con un texto similar a "Add STS Reference".

Esta entrada del menú contextual permitirá lanzar la herramienta "Federation Utility" (incluida con el SDK de WIF) gracias a la cual se puede configurar el servicio para trabajar con un nuevo STS o bien un STS existente.

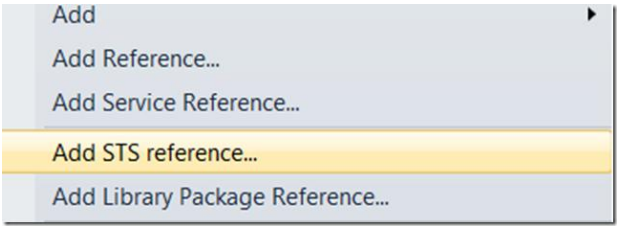


Figura 2.3.- Añadir la referencia a un STS

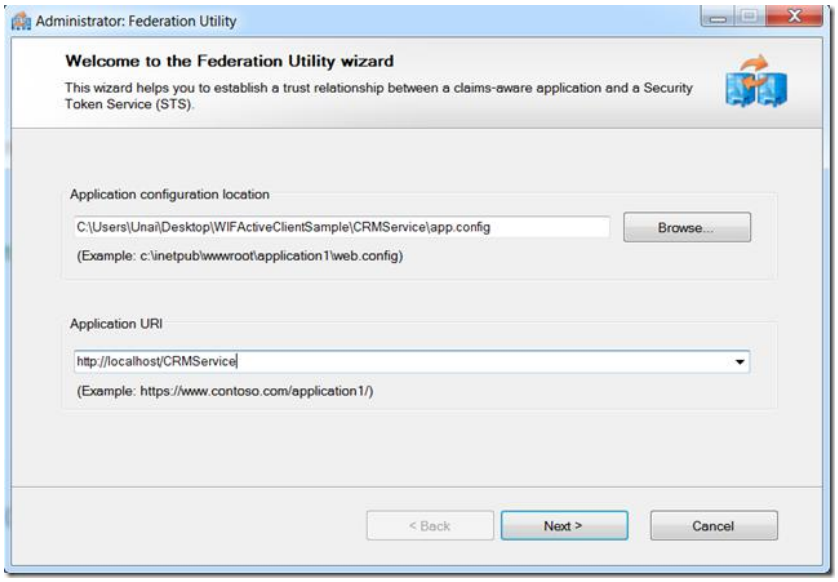


Figura 2.4.- Añadir la referencia a un STS

Este asistente pedirá los datos relativos al path del archivo de configuración del servicio WCF (un web.config si es un sitio web) y la URI del mismo.

Esta URI es importante, puesto que la misma servirá como valor de AudienceUri, es decir, como indicador de "para quien será el token generado por el STS".

Una vez configurado estos elementos el asistente preguntará si se quiere trabajar con un nuevo STS o un STS existente.

En este caso, se selecciona la opción crear un nuevo STS, tal y como se puede ver en la siguiente figura (en el caso de disponer ya de un STS se puede establecer la dirección de su metadata).

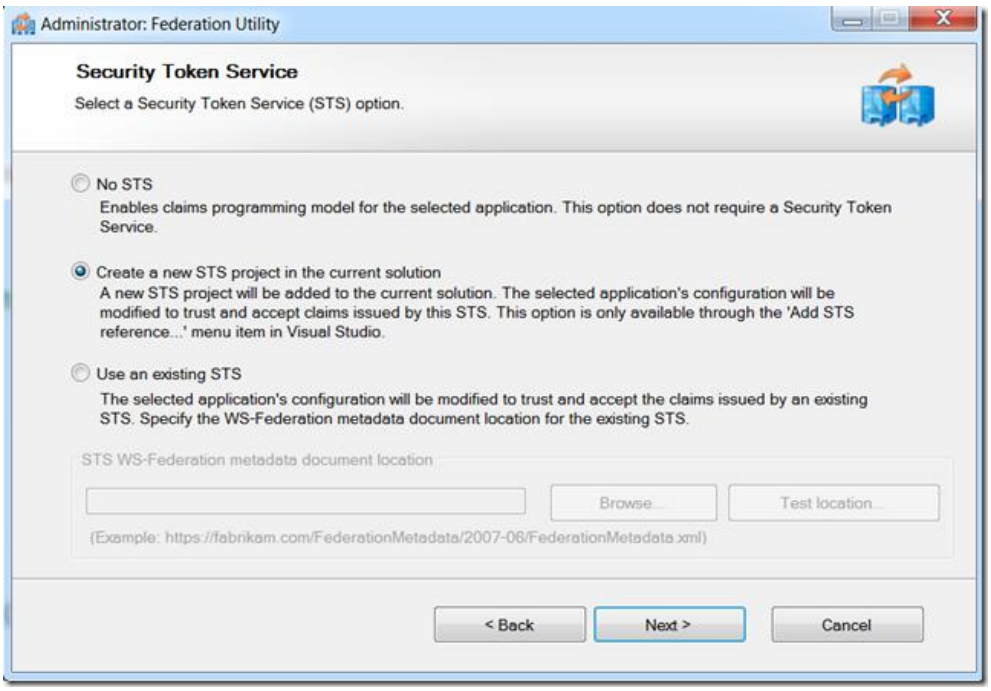


Figura 2.5.- Añadir la referencia a un STS

3.1.3.- Revisión de la configuración

El trabajo del asistente anteriormente ejecutado se traduce, como se puede ver en la última pantalla de resumen, en una serie de cambios a la configuración del servicio WCF, así como en la inclusión de una serie de certificados a nivel de máquina (estos certificados están pensados para el entorno de desarrollo y como se verá más adelante se usarán internamente por el STS para el cifrado y la encriptación de los tokens).

En cuanto a los cambios en la configuración, se puede ver la inclusión de un nuevo endpoint incluyendo un binding compatible con la federación de credenciales:

```
<endpoint address=""
                      binding="ws2007FederationHttpBinding"
                      contract="CRMService.ICRMManager"
                      bindingConfiguration="CRMService.ICRMManager_ws2007FederationHttpBinding"
/>
```

Dentro de la configuración también se incluye un comportamiento nuevo (no existente por defecto en WCF pero incluido con una extensión) llamada **FederatedServiceHostConfiguration**.

```
    <federatedServiceHostConfiguration name="CRMService.CRMManager" />
```

Este comportamiento tiene como objetivo permitir especificar la configuración de WIF que se va a utilizar. Esta configuración, se establece con una nueva sección personalizada por WIF llamada **microsoft.identitymodel**.

```
    <microsoft.identityModel>
        <service name="CRMService.CRMManager">
          <audienceUris>
            <add value="http://localhost/CRMService" />
          </audienceUris>
          <issuerNameRegistry
type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,
Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
            <trustedIssuers>
              <add thumbprint="6C6CE1DA53CE9F34F9DB4FFFAADDA8D84468C6BF"
name="http://localhost:2325/CRMService_STS/Service.svc" />
            </trustedIssuers>
          </issuerNameRegistry>
        </service>
      </microsoft.identityModel>
```

3.1.4.- Revisión del STS creado

En la aplicación en este momento se dispone de un nuevo proyecto, por defecto generado como un sitio web, que contiene el código de un STS personalizado, cuyo código está contenido dentro de la carpeta especial de ASP.NET App_Code.

Este código contiene un helper para trabajar con los certificados, por defecto con los generados con la herramienta, aunque los mismos pueden ser modificados cambiando en las appsettings del sitio web las claves:

```
     <appSettings>
        <add key="IssuerName" value="ActiveSTS"/>
        <add key="SigningCertificateName" value="CN=STSTestCert"/>
        <add key="EncryptingCertificateName" value="CN=DefaultApplicationCertificate"/>
      </appSettings>
```

Junto con este helper, también se tiene la definición del SecurityTokenService personalizado, en forma de la clase:

```
public class CustomSecurityTokenService
    : SecurityTokenService
{
}
```

Este STS personalizado dispone de un método especial **GetOutputClaimsIdentity** cuyo objetivo es la de ofrecer el conjunto de Claims que hay que otorgar al usuario que se ha autenticado en el STS.

El código por defecto de este método, que se puede ver en el siguiente fragmento de código, ofrece para cualquier usuario dos claims, una claim de tipo nombre y una claim de tipo rol.

```
    /// <summary>
        /// This method returns the claims to be issued in the token.
        /// </summary>
        /// <param name="principal">The caller's principal.</param>
        /// <param name="request">The incoming RST, can be used to obtain addtional
information.</param>
        /// <param name="scope">The scope information corresponding to this request.</param>///
        /// <exception cref="ArgumentNullException">If 'principal' parameter is null.</exception>
```

```
        /// <returns>The outgoing claimsIdentity to be included in the issued token.</returns>
        protected override IClaimsIdentity GetOutputClaimsIdentity( IClaimsPrincipal principal,
RequestSecurityToken request, Scope scope )
        {
            if ( null == principal )
            {
                throw new ArgumentNullException( "principal" );
            }

            ClaimsIdentity outputIdentity = new ClaimsIdentity();

            // Issue custom claims.
            // TODO: Change the claims below to issue custom claims required by your application.
            // Update the application's configuration file too to reflect new claims requirement.

            outputIdentity.Claims.Add( new Claim( System.IdentityModel.Claims.ClaimTypes.Name,
principal.Identity.Name ) );
            outputIdentity.Claims.Add( new Claim( ClaimTypes.Role, "Manager" ) );

            return outputIdentity;
        }
```

3.1.5.- Probando el escenario

Para probar este escenario, solamente es necesario crear un cliente cualquiera que haga uso del servicio,

Una vez hecho esto se podrá ver como la llamada al servicio funciona correctamente y la validación y obtención de claims se realiza correctamente.

Con el fin de mostrar en el servicio las claims de ejecución en cada uno de los métodos se utilizará en ellos una llamada a un método llamado **CheckSecurity**, que por ahora, solamente tendrá el propósito de enseñar por pantalla los claims del usuario autenticado.

```
            void CheckSecurity()
             {
                 //TODO:Check security here!

                 IClaimsPrincipal principal = Thread.CurrentPrincipal as IClaimsPrincipal;
                 if (principal != null)
                 {
                     Console.WriteLine("User claims collection");
                     foreach (var claim in ((IClaimsIdentity)principal.Identity).Claims)
                     {
                         Console.WriteLine("\t Name:{0} Value:{1}",claim.ClaimType,claim.Value);
                     }
                 }
             }
```

En el siguiente apartado se verá cómo modificar la parametrización del servicio para utilizar otro tipo de autenticación y alguna característica a mayores que pueden ser interesantes a los desarrolladores.

4.- CONFIGURACIÓN: AUTENTICACIÓN

En el sencillo ejemplo que se acaba de ver, no se ha comentado nada sobre algún tipo de configuración, dejando los parámetros por defecto, de los generados por el asistente FedUtil.

A lo largo de esta entrada se verá como modificar algunos elementos habituales de un Security Token Service.

4.1.- Modificación de la autenticación

Una de las cosas más habituales en cuanto a la configuración de un STS (lógicamente si este hace las tareas de un Identity Provider) es el cambio del mecanismo de autenticación.

Por defecto, cuando se crea un STS por medio del asistente este presenta la configuración por defecto, delegando la autenticación de los tokens en un manejador llamado **WindowsUserNameSecurityTokenHandler**.

Si no se quiere utilizar las credenciales Windows como mecanismo de validación de los usuarios se tendrá que modificar el "handler" asociado.

Esta tarea, requiere de varios pasos, en primer lugar, modificar el binding asociado al endpoint que por defecto presenta el siguiente aspecto.

```
<ws2007HttpBinding>
        <binding name="ws2007HttpBindingConfiguration">
          <security mode="Message">
            <message establishSecurityContext="false"  />
          </security>
        </binding>
</ws2007HttpBinding>
```

Aunque se pueda pensar que el trabajo consiste en delegar en WCF y sus enlaces todo el proceso, en realidad, WIF, puentea toda la infraestructura para delegar el trabajo en su propio stack.

Si se quiere establecer una autenticación en base a usuario y contraseña, por ejemplo, se tendría que modificar la configuración anterior incluyendo el atributo clientCredentialType="UserName".

```
<ws2007HttpBinding>
        <binding name="ws2007HttpBindingConfiguration">
          <security mode="Message">
            <message establishSecurityContext="false" clientCredentialType="UserName"  />
          </security>
        </binding>
</ws2007HttpBinding>
```

Una vez hecho esto, se procederá a indicarle a WIF cuál será el manejador de los tokens que lleguen a nuestro STS. Para ello, se incluirá la sección de configuración personalizada Microsoft.IdentityModel y se establecerá un "handler" concreto.

```
<microsoft.identityModel>
    <service>
      <securityTokenHandlers>
        <remove
type="Microsoft.IdentityModel.Tokens.WindowsUserNameSecurityTokenHandler,Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
        <add type="Helpers.CustomTokenHandler,Helpers"/>
      </securityTokenHandlers>
    </service>
</microsoft.identityModel>
```

Hay que tener en cuenta cómo en esta sección se ha quitado el manejador encargado de "manejar" identidades Windows por un nuevo manejador llamado CustomTokenHandler.

Todos estos "manejadores" están incluidos dentro de la jerarquía que impone SecurityTokenHandler y que por defecto, se pueden ver en la siguiente imagen extraída de reflector.

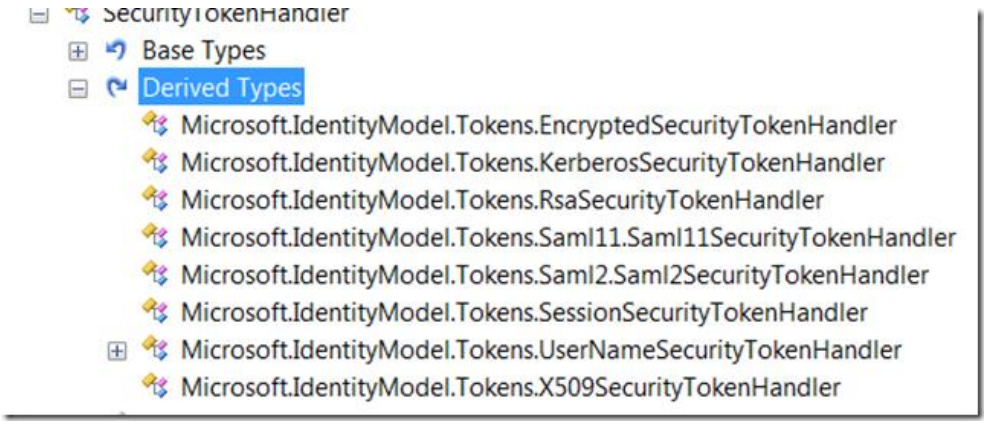


Figura 2.6.- Manejadores WIF

Cada uno de los handlers de esta jerarquía servirá como base para los distintos mecanismos de autenticación, usuario, kerberos, certificados, tokens saml11 o saml2 etc…

Como el propósito de este ejemplo es crear un manejador para credenciales de tipo usuario/contraseña se tendrá que hacer una implementación personalizada de la clase UserNameSecurityTokenHandler.

Crear un manejador de tokens de seguridad es tan sencillo como sobrescribir dos elementos, una propiedad llamada CanValidateToken y un método de validación de tokens ValidateToken.

El siguiente fragmento de código muestra una posible implementación de un token de seguridad, lógicamente, es un ejemplo sencillo y no se hace la validación de los tokens contra ningún almacén.

```
        public class CustomTokenHandler
        : Microsoft.IdentityModel.Tokens.UserNameSecurityTokenHandler
        {
            public override bool CanValidateToken
            {
                get
                {
                    return true;
                }
            }
            public override Microsoft.IdentityModel.Claims.ClaimsIdentityCollection
    ValidateToken(System.IdentityModel.Tokens.SecurityToken token)
            {
                //validate the token, the SecurityToken can be (
    RsaSecurityToken,WindowsSecurityToken,SessionSecurityToken ..... ) but
                //for UserNameSecurityTokenHandler the SecurityToken is a UserNameSecurityToken

                UserNameSecurityToken userNameToken = token as UserNameSecurityToken;
                if (userNameToken != null)
                {
                    //TODO:validate data into a specific store

                    if (userNameToken.UserName.Equals("unai",
    StringComparison.InvariantCultureIgnoreCase))
                    {
                        //Create the principal claims identity collection
                        var claims = new ClaimsIdentityCollection();
                        var claimIdentity = new ClaimsIdentity();

                        //assign identity ( this is the input collection of claims in
    GetOutputClaimsIdentity in SecurityTokenService)
                        claimIdentity.Claims.Add(new Claim(WSIdentityConstants.ClaimTypes.Name,
    "unai"));

                        //return claims
                        claims.Add(claimIdentity);
                        return claims;
                    }
                    else
                        throw new SecurityException("Invalid user or password");
                }
```

```
                else
                    throw new SecurityException("Invalid token for Custom Token Handler!");

            }
        }
```

Con estos pasos, ya se tiene modificado el sistema de autenticación, aunque falta un pequeño paso.

El pasos que falta consiste en incluir el certificado que es necesario usar para la encriptación de los datos incluidos en el mensaje de validación (user/pwd), del mismo modo que con cualquier servicio WCF. Para ello, simplemente se incluirá dentro del comportamiento una entrada **ServiceCredentials**.

```
    <serviceBehaviors>
       <behavior name="ServiceBehavior">
         <serviceMetadata httpGetEnabled="true" />
         <serviceDebug includeExceptionDetailInFaults="false" />
         <serviceCredentials>
           <serviceCertificate findValue="PORTBLACKCODE" x509FindType="FindByIssuerName" />
         </serviceCredentials>
       </behavior>
    </serviceBehaviors>
```

Y con esto ya se tiene todo el trabajo realizado, solamente quedará actualizar los clientes para actualizar la configuración de los enlaces hacia el nuevo STS.

5.- CONFIGURACIÓN

En el punto entrada se vio como modificar el manejador de tokens del STS, de forma, que se podría cambiar el sistema de autenticación de usuarios dentro de este. De hecho, se modificó el STS generado por defecto con FedUtil introduciendo un UsernameSecurityTokenHandler personalizado.

A lo largo de esta entrada, se verán algunos elementos particulares de la configuración que no han sido tenidos en cuenta pero que también es importante entenderlos.

5.1.- IssuerNameRegistry

Si se lee la documentación del SDK de WIF se puede ver a esta pieza como la responsable de rechazar los tokens de emisores inválidos, los STS en los que no confiamos.

Por defecto, las plantillas del asistente establecen un IssuerNameRegistry por defecto conocido como ConfigurationBasedIssuerNameRegistry, que hace el mapeo y rechazo de los emisores en función del thumbprint del token (certificado) presentado.

Como también se comenta en la documentación, este elemento debe de ser reemplazado en producción, creando un IssuerNameRegistry customizado

Crear un nuevo issuer name registry es “tan sencillo o complicado” como crear una subclase de IssuerNameRegistry y sobrescribir el método GetIssuerName.

```
        public class CustomIssuerNameRegistry
            :IssuerNameRegistry
        {
            public override string GetIssuerName(System.IdentityModel.Tokens.SecurityToken
securityToken)
            {

            }
        }
```

Si el valor devuelto por este método es nulo, entonces el sistema interpreta que este issuer no es admitido y por lo tanto no es posible aceptar tokens del mismo.

Lógicamente, este trabajo requeriría en la realidad disponer de un almacén configurable de issuers y propiedades de los mismos que pudiéramos configurar y administrar. Este trabajo, no lo haremos en apartado, solamente se incluirá una posible implementación "tonta" que devuelve el nombre del STS utilizado en los ejemplos.

```
    public class CustomIssuerNameRegistry
            :IssuerNameRegistry
        {
            public override string GetIssuerName(System.IdentityModel.Tokens.SecurityToken
securityToken)
            {
                if (securityToken == null)
                    throw new ArgumentNullException("securityToken");

                X509SecurityToken x509Token = securityToken as X509SecurityToken;
                if (x509Token != null)
                {
                    return "http://localhost:2325/CRMService_STS/Service.svc";
                }
                else
                    return null;
            }
        }
```

5.2.- AudienceUri

Si se revisa la configuración del Relaying Party se puede ver como la sección microsoft.identitymodel incluye dentro de ella un elemento llamado AudienceUris.

Este elemento permite establecer una colección de elementos validos a los que se les aplica un toquen. Es decir, cuando un token es emitido, éste se establece o aplica para alguien, el Relaying Party que lo ha solicitado.

Este token incluye una propiedad AppliesTo gracias a la cual se podrá validar que es el mismo se aplica al Relaying Party que se tiene.

Por ello esta colección ya contiene la dirección del Relaying Party con el que se esté trabajando. Lógicamente, si esta información no es igual que la que trae el token se producirá una excepción de seguridad.

```
    <microsoft.identityModel>
        <service name="CRMService.CRMManager"  >
          <audienceUris>
            <add value="http://localhost/CRMService" />
          </audienceUris>Federation Metadata
```

5.2.1.- Federation Metadata

Otro de los elementos importantes cuando se está trabajando con un Relaying Party y STS es la información de metadata.

El asistente de WIF, FedUtil genera un nuevo STS y crea en el proyecto de este un archivo llamado FederationMetadata.xml.

Este archivo, firmado por defecto con el asistente utilizando un certificado llamado STSCert, permite exponer toda la información del STS, incluyendo en ésta, por ejemplo, la información de los claims que éste conoce y que puede enviar en los tokens de autenticación.

Se podría decir que estos documentos son a la seguridad como el WSDL a los servicios.

Como es de imaginar, mantener estos documentos actualizados es una tarea vital, sobre todo si éstos son nuestro contrato de relación con los Relaying Party. Por defecto, este trabajo no lo se tiene disponible de una forma sencilla.

Cuando se crea un STS el asistente genera el archivo de metadata y las entradas necesarias en el web.config para dejarlo accesible, sin embargo, si se añaden nuevos claims a los tokens o bien se cambia alguna configuración, este archivo no se actualiza y por lo tanto podría dar problemas. Lógicamente, como está firmado, trabajar a mano con este documento no es posible.

Para la tarea de automatizar la generación dinámica de este archivo de metadata se tienen varias opciones, opciones que puede ver en este apartado.

La opción más habitual, de Vittorio Bertocci, es la de disponer de un servicio WCF REST que genere este metadata al vuelo, algo parecido al código siguiente.

Hay que tener en cuenta que este código es "código demo" aunque es una buena base para empezar a adaptarlo.

```
        public class FederationMetadata : IFederationMetadata
        {

            System.Xml.Linq.XElement IFederationMetadata.FederationMetadata()
            {
                // hostname
                string host = WebOperationContext.Current.IncomingRequest.Headers["Host"];
                EndpointAddress realm = new EndpointAddress(String.Format("https://{0}", host));
                EndpointAddress passiveEndpoint = new
    EndpointAddress(String.Format("https://{0}/Sts.aspx", host));
                // metadata document
                EntityDescriptor entity = new EntityDescriptor(new
    EntityId(realm.Uri.AbsoluteUri));
                SecurityTokenServiceDescriptor sts = new SecurityTokenServiceDescriptor();
                entity.RoleDescriptors.Add(sts);
                // signing key
                var signingCredentials = new
    X509SigningCredentials(CertificateUtil.GetCertificate(
                                                                    StoreName.My,
    StoreLocation.LocalMachine,
                                                                   "CN=STSTestCert"));

                KeyDescriptor signingKey = new
    KeyDescriptor(signingCredentials.SigningKeyIdentifier);
                signingKey.Use = KeyType.Signing;
                sts.Keys.Add(signingKey);

                // claim types
                sts.ClaimTypesOffered.Add(new DisplayClaim(ClaimTypes.Email, "email address",
    "The subject's email address."));
                // passive federation endpoint
                sts.PassiveRequestorEndpoints.Add(passiveEndpoint);
                // supported protocols
                sts.ProtocolsSupported.Add(new Uri(WSFederationConstants.Namespace));
                // add passive STS endpoint
                sts.SecurityTokenServiceEndpoints.Add(passiveEndpoint);
                // metadata signing
                entity.SigningCredentials = signingCredentials;
                // serialize
                MetadataSerializer serializer = new MetadataSerializer();
                MemoryStream stream = new MemoryStream();
                serializer.WriteMetadata(stream, entity);
                stream.Flush();
                stream.Seek(0, SeekOrigin.Begin);
                XmlReader xmlReader = XmlTextReader.Create(stream);
                return XElement.Load(xmlReader);
            }
```

Una alternativa a la creación de un servicio WCF REST es la de crear un IHttpHandler que responda a una dirección, generalmente a algo como 2007-07/FederationMetadata.xml que genere, con un código similar al anterior, la información de metadata.

6.- CUSTOM SECURITY TOKEN HANDLER

Trabajando con WIF es muy fácil encontrarse con el concepto de "**authentication outsourcing**", siendo el uso más habitual en los ejemplos más sencillos el uso de autenticación Windows.

Sin embargo, en la mayoría de las aplicaciones que se construyen los mecanismos de autenticación (porque se exponen por internet o bien porque no se dispone de un Active Directory con todo el repositorio de usuarios) suelen diferir de éstos.

En este apartado se intentará mostrar de la forma más detallada posible el uso de autenticación personalizada. Lógicamente, se asumirá que el lector ya tiene los conocimientos básicos sobre WIF y la situación de las distintas piezas en un escenario básico.

Aunque en un principio pudiera parecer que el objetivo a la hora de securizar un STS por medio de una autenticación personalizada pudiera pasar por la configuración del binding del mismo, la realidad es que la integración de WIF con WCF y los hooks de esta API sobre la extensibilidad permiten modificar la autenticación en esta pieza (STE), haciendo uso de los SecurityTokenHandlers y las subclases de ésta.

La configuración por defecto en la plantilla de STE es seguridad basada en el mensaje con credenciales Windows, por lo tanto, hace uso de **WindowsUserNameSecurityTokenHandler**.

Si se quiere cambiar este mecanismo, y por ejemplo, incluir una validación personalizada de usuario/contraseña se tendrían que realizar las siguientes tareas.

- **Modificar el binding del STE para establecer el tipo de credenciales a UserName.**

```
<binding name="ws2007HttpBindingConfiguration">
          <security mode="Message">
            <message establishSecurityContext="false" clientCredentialType="UserName" />
          </security>
</binding>
```

- **Crear un CustomUserNameSecurityTokenHandler como sigue:**

```
public class CustomUserNameSecurityTokenHandler
        :UserNameSecurityTokenHandler
    {
        public override bool CanValidateToken
        {
            get
            {
                return true;
            }
        }
        public override Microsoft.IdentityModel.Claims.ClaimsIdentityCollection
ValidateToken(System.IdentityModel.Tokens.SecurityToken token)
        {
            UserNameSecurityToken userToken = token as UserNameSecurityToken;
            if (userToken != null)
            {
                string user = userToken.UserName;
                string password = userToken.Password;

                //TODO: Esto es un mejor ejemplo para el blog,
                //por favor incluir aquí la lógica de validacion necesaria
                if (user.StartsWith("plainconcepts"))
                {
                    IClaimsIdentity identity = new ClaimsIdentity();
                    identity.Claims.Add(new Claim(WSIdentityConstants.ClaimTypes.Name,
user));

                    return new ClaimsIdentityCollection(new IClaimsIdentity[] { identity });
                }
```

```
                    else
                        throw new InvalidOperationException();
                }
                else
                    throw new InvalidOperationException();
            }
        }
```

- **Configurar dentro del STE la inclusión de este token handler**

```
   <configSections>
       <section name="microsoft.identityModel"
type="Microsoft.IdentityModel.Configuration.MicrosoftIdentityModelSection,
Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
     </configSections>
     <microsoft.identityModel>
       <service>
         <securityTokenHandlers>
           <remove
type="Microsoft.IdentityModel.Tokens.WindowsUserNameSecurityTokenHandler,Microsoft.IdentityModel
, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
           <add type="Helper.CustomUserNameSecurityTokenHandler,Helper"/>
         </securityTokenHandlers>
       </service>
     </microsoft.identityModel>
```

Con estos tres pasos básicos ya tenemos un STS funcionando con autenticación personalizada.

7.- AUTORIZACIÓN DENTRO DEL PIPELINE DE WIF

En esta apartado se hablará de cómo incluir el mecanismo de validación, transformación y autorización de claims una vez se haya realizado el pipeline completo de WIF, es decir, una vez que el STS haya devuelto el token al Relaying Party.

Si se revisa la documentación, se verá rápidamente cómo existe una clase base denominada ClaimsAuthorizationManager que permite realizar esta tarea de una forma fácil, puesto que solamente se tiene que heredar de ella y sobrescribir el método CheckAccess, como se puede ver en el siguiente trozo de código.

```
public class GeekClaimAuthorizationManager
        : ClaimsAuthorizationManager
{
        public override bool CheckAccess(AuthorizationContext context)
        {
            //TODO: add claim transformation, validation, etc here!
            return true;
        }
}
```

El elemento **AuthorizationContext**, permite tener acceso a tres elementos fundamentales; por un lado el recurso o recursos a los que se desea acceder, por ejemplo si se está dentro de un sitio Web ASP.NET securizado con WS-Federation dará la página que se ha solicitado.

Otro de los elementos proporcionados por esta clase AuthorizationContext consiste en el **Action** o verbo utilizado (GET o POST si es por ejemplo un servicio WCF).

Para terminar el último elemento proporcionado por el contexto enviado al método CheckAccess es el **IPrincipal** establecido o autenticado dentro del STS y para el cual se dispone de un token de seguridad.

Una vez creada la clase y establecida la lógica de negocio que se, habrá que realizar un par de pasos adicionales. El primero es acudir a la configuración de Relaying Party y en la sección Microsoft.IdentityModel incluir el elemento ClaimsAuthorizationManager dentro de services

```
<claimsAuthorizationManager type="Utils.GeekClaimAuthorizationManager"/>
```

Una vez hecho esto aún queda un último paso. Por defecto, WIF no incluye dentro de su procesamiento la autorización de los tokens recibidos, solamente la autenticación y el establecimiento de la sesión para los cuales se debe incluir dos módulos http.

```
<system.webServer>
  <modules runAllManagedModulesForAllRequests="true">
    <add name="WSFederationAuthenticationModule" type="Microsoft.IdentityModel.Web.WSFe
    <add name="SessionAuthenticationModule" type="Microsoft.IdentityModel.Web.SessionAu
```

Figura 2.7.- Módulo HTTP de WIF

Si se quiere agregar dentro de la cadena de procesamiento el uso de autorización de claims se deberá incluir un nuevo módulo que es el siguiente:

```
   <add name="ClaimsAuthorizationModule"
type="Microsoft.IdentityModel.Web.ClaimsAuthorizationModule, Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
preCondition="managedHandler"/>
```

Lógicamente si se usa WebDev.WebServer se tiene que incluirlo también en la sección httpModules
Con estos pasos ya es posible habilitar el procesamiento de autorizaciones de claims.
Como acción adicional a estos pasos se podría extender el elemento ClaimsAuthorizationManager de la siguiente forma:

```
   <claimsAuthorizationManager type="Utils.GeekClaimAuthorizationManager">
   <policy resource="/[RESOURCE]" action="GET" claimName="[ClaimName]"
claimValue="[ClaimValue]"/>
   </claimsAuthorizationManager>
```

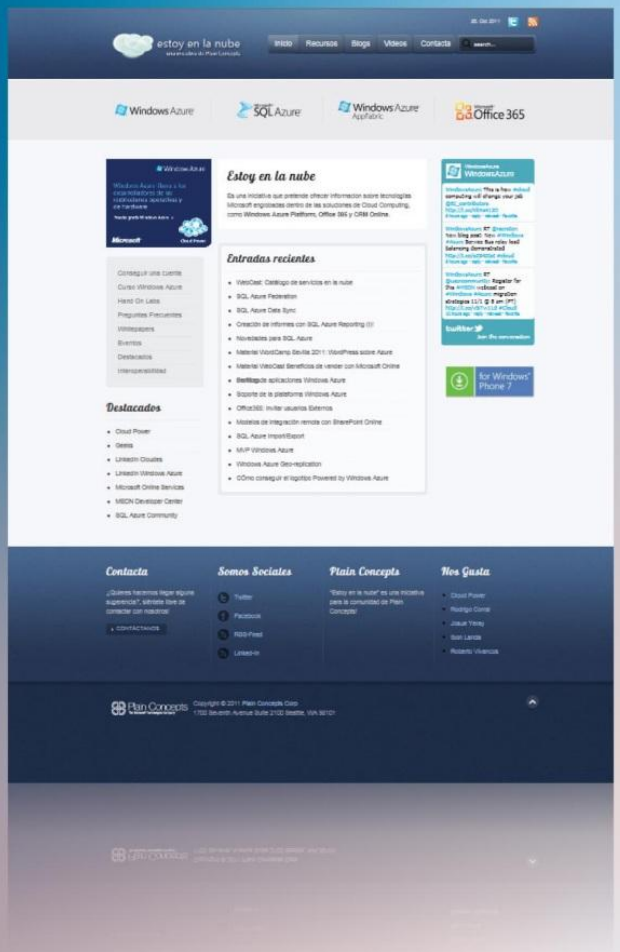
El contenido incluido dentro del elemento ClaimAuthorizationManager es completamente personalizado, y podría valer, por ejemplo, para establecer la configuración que se quiere en el sistema de autorización. Para que él mismo reciba esta información solamente se tendrá que agregar un constructor nuevo que tome un XmlNodeList, como se muestra a continuación.

```
public class GeekClaimAuthorizationManager
        : ClaimsAuthorizationManager
{
        public GeekClaimAuthorizationManager(XmlNodeList xmlExtended)
        {
            //review and set xmlExtended information
        }
        public override bool CheckAccess(AuthorizationContext context)
        {
            //TODO: add claim transformation, validation, etc here!
            return true;
        }
}
```

estoy en la nube

INICIATIVA DE PLAIN CONCEPTS

www.estoyenlanube.com



Windows Azure

www.plainconcepts.com

Plain Concepts is a company specialized in Microsoft technologies, agile methodologies, Application Lifecycle Management, performance tuning, advanced debugging, software architecture and User Experience.

Plain Concepts focuses on delivering high quality consulting, mentoring and training as well as in being an effective and reliable team resolving all type of software development issues.

Plain Concepts

¿Aún quieres más?

campus MVP

Formación online especializada en tecnologías Microsoft.

krasis PRESS

Los libros que lo saben todo sobre tecnologías Microsoft.

Síguenos y descubrirás los mejores trucos y recursos:

facebook.com/campusmvp twitter.com/campusmvp

feed your brain®

- Sin tener que desplazarse
- Sin romper el ritmo de trabajo
- Preguntándole a los que más saben

infórmate ya:
902 876 475
www.campusmvp.com

http://www.krasis.com

krasis

Microsoft Partner
Silver Learning
Silver Software Development