Nº29 septiembre 2016

CompartiMOSS

REVISTA ESPECIALIZADA EN TECNOLOGÍAS MICROSOFT

Entrevista Darwin Castro Marín

SharePoint y Azure WebJobs

Power BI: Conexión con SQL Server y Oracle

Nueva Experiencia de Usuario en Listas en SPO

Contenido

Staff

CompartiMOSS es una publicación independiente de distribución libre en forma electrónica. Las opiniones aquí expresadas son de estricto orden personal, cada autor es completamente responsable de su propio contenido.

DIRECCIÓN GENERAL
- Gustavo Velez
- Juan Carlos Gonzalez
- Fabian Imaz
- Alberto Diaz

DISEÑO Y DIAGRAMACIÓN
- Santiago Porras Rodríguez

Contacte con nosotros

revista@compartimoss.com
gustavo@gavd.net
jcgonzalezmartin1978@hotmail.com
fabian@siderys.com.uy
adiazcan@hotmail.com

BLOGS
http://www.gavd.net
http://geeks.ms/blogs/jcgonzalez
http://blog.siderys.com
http://geeks.ms/blogs/adiazmartin

REDES SOCIALES
Facebook:
http://www.facebook.com/group.php?gid=128911147140492
LinkedIn:
http://www.linkedin.com/groups/CompartiMOSS-3776291
Twitter:
@CompartiMOSScom

03

Editorial

Siguiendo con el proceso incesante de renovación e innovación que nos hemos impuesto desde el principio en CompartiMOSS, continuamos en este número integrando nuevas tecnologías de Microsoft en nuestro contenido. Como tal, damos la bienvenida a Gastón Cruz que se está encargando de manejar la sección correspondiente a BI (y, por supuesto, de Power BI). Estamos seguros que la revista continuará su ruta informativa con sus aportes y queremos animar a todos nuestros lectores que quieran escribir artículos sobre BI a que tomen contacto con Gastón directamente, o por intermedio de la revista misma (revista@compartimoss.com).

Azure toma cada vez más impulso, y lo vemos no solo en los resultados financieros que Microsoft presentó hace unas cuantas semanas, sino también en CompartiMOSS. Cuando hace un año no publicábamos más de un artículo por número, ahora prácticamente la mitad de la revista contiene información relacionada directa o indirectamente con Azure. La nube es la gran apuesta del momento, no solo para Microsoft, sino para todos los que trabajamos profesionalmente con herramientas informáticas. Solamente el futuro dirá como se van a seguir desarrollando la nube con Azure y Office 365, pero por el momento parece que es, sin ninguna duda, el camino a seguir.

Esperamos que la información de este número les parezca tan interesante como de costumbre, y que se diviertan leyendo los artículos, tanto como los autores lo hicieron escribiéndolos, y nosotros publicándolos.

El Equipo Editorial de CompartiMOSS

04

Enmarcha: Como facilita el Unit Testing en SharePoint / Office 365

Conceptos Previos

Antes de empezar con el articulo vamos a definir algunos conceptos de Unit Testing para dejar aclarado algunos puntos.

- Stub-> Un trozo de código usado como sustituto de alguna otra funcionalidad. Un stub puede simular el comportamiento de código existente (tal como un procedimiento en una máquina remota) o ser el sustituto temporal para un código aún no desarrollado.
- Mock -> Se usan para simular el comportamiento de objetos complejos cuando es imposible o impracticable usar al objeto real en la prueba. De paso resuelve el problema del caso de objetos interdependientes, que para probar el primero debe ser usado un objeto no probado aún, lo que invalida la prueba: los objetos simulados son muy simples de construir y devuelven un resultado determinado y de implementación directa, independientemente de los complejos procesos o interacciones que el objeto real pueda tener.

con Enmarcha podemos hacer Unit Testing sobre SharePoint de una forma sencilla

Los objetos simulados se usan en lugar de objetos reales que tengan algunas de estas características:

- Devuelven resultados no determinísticos (por ejemplo, la hora o la temperatura).
- Su estado es difícil de crear o reproducir (por ejemplo, errores de conexión).
- Es lento (por ejemplo, el resultado de un cálculo intensivo o una búsqueda en una BBDD).
- El objeto todavía no existe o su comportamiento puede cambiar.
- Debería incluir atributos o métodos exclusivamente para el testeo.

Otro de los conceptos que muchos desarrolladores tienen en la cabeza es que juntan un concepto como Test Driven Development (TDD) como hacer Test. Hacer Test no implica que necesariamente tengas que hacer TDD, TDD es una práctica que podemos decir que antes de empezar a tirar una línea de código ya implementas los Test que vas a hacer para que una vez vas implementando la funcionalidad vas comprobando si este es correcto o no. Y una vez es correcto podemos dedicarnos a refactorizar nuestro código y dejarlo como idealmente queremos.

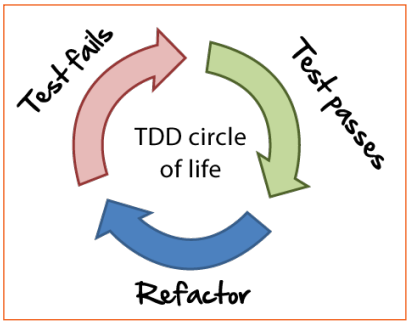


Imagen 1.- Conceptos de TDD.

Hagamos Testing con SharePoint

SharePoint como tal es una plataforma que ha ido evolucionando con el tiempo, principalmente por su mayor importancia dentro de grandes organizaciones. Esto ha provocado que versión a versión el producto haya ido mejorando más y más y de cara al desarrollador se ha ido haciendo mejoras en cuanto al ciclo de vida de la solución. Estas mejoras han sido adoptadas por la gran mayoría de equipos de desarrollo. Sin embargo, cuando se habla sobre temas como Unit Testing estos desarrolladores siguen igual que en sus inicios, es decir, no hay Test Unitarios en la mayoría de las ocasiones. Muchos de estos desarrolladores indican que es imposible hacer Unit Test en SharePoint. Algunos de estos argumentos se basan en que el SDK de SharePoint se compone de innumerables clases “selladas” y que por este motivo no se pueden hacer mocks. Esta afirmación no es más que una verdad a medias ya que existen productos comerciales como TypeMock o JustMock que se encargan de generar los mocks (no comento los emuladores de SharePoint porque podemos decir la propia Microsoft no los ha evolucionado desde 2010). Si bien es cierto que para alguien que empiece a hacer testing, SharePoint no es una plataforma que lo facilite de una forma sencilla (se han de tener claros cada uno de los conceptos y una forma de desarrollar clara).

Por estos motivos cuando empezamos a trabajar en SharePoint, uno de nuestros principales objetivos fue seguir las buenas prácticas y consejos que se utilizan en todas las plataformas .NET. No debemos de tratar SharePoint como una plataforma distinta y en la que todo vale. De este conocimiento como hablamos en el artículo del número 27

surgió ENMARCHA.

Entre sus múltiples utilidades Enmarcha tiene una Capa de Acceso a las listas de SharePoint que facilita el aprendizaje sobre la plataforma. Esta clase surgió de intentar en la medida de lo posible seguir una arquitectura N-Capas en nuestros desarrollos y tener una capa de acceso a datos siguiendo el patrón Repositorio. Este patrón lo podemos definir como un repositorio que es un mediador entre el dominio de la aplicación y los datos que le dan persistencia. Con este planteamiento podemos pensar que el usuario de este repositorio no necesitaría conocer la tecnología utilizada para acceder a los datos, sino que le bastaría con saber las operaciones que nos facilita este "mediador", el repositorio.

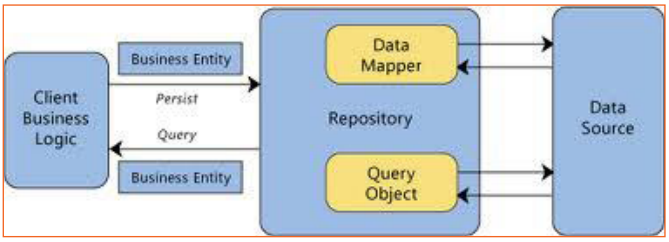


Imagen 2.- Patrón Repositorio.

Este patrón en Enmarcha lo hacemos implementando la interfaz IRepository, la cual tiene la siguiente definición.

```
public interface IRepository<T> : IPageable
  {
    T Get(int id);
    ICollection<T> GetAll();
    ICollection<T> GetAll(int page);
    ICollection<T> Query(IQuery query, int page);
    ICollection<T> Query(string query, int page);
    int Insert(T data);
    bool Save(int id, T data);
    bool Delete(int id);
  }
```

Para el acceso a listas utilizando SharePoint Server Object Model hemos implementado la clase SharePointRepository de tal forma que dado una clase de C# podamos "mapearla" con una lista de SharePoint.

Hacer Test no implica que necesariamente tengas que hacer TDD

Ahora bien, ¿cómo hacemos Unit Testing con Enmarcha?

Partimos de la base de que tenemos un WebPart que nos muestran los clientes que tienen un saldo de más de 1.000 €. Para ello en SharePoint lo que tendremos es una lista de SharePoint llamada cliente. Y dentro de nuestra solución de Visual Studio tendremos lo siguiente:

- Una clase "Customer"(mapea la clase C# con la lista de SharePoint) con la siguiente definición

```
public class Customer
  {
    [Enmarcha(AddPrefeix = false, Create = false, Type =
TypeField.Text)]
    public string ID { get; set; }
    [Enmarcha(AddPrefeix = false, Create = false, Type =
TypeField.Text)]
    public string CIF { get; set; }
    [Enmarcha(AddPrefeix = false, Create = false, Type =
TypeField.Text)]
    public string Title { get; set; }
    [Enmarcha(AddPrefeix = false, Create = false, Type =
TypeField.Text)]
    public string Direccion { get; set; }
    [Enmarcha(AddPrefeix = false, Create = false, Type =
TypeField.Number)]
    public double Saldo { get; set; }
  }
```

- En la Capa de Servicio tendremos una interfaz ICustomerService con los siguientes métodos

```
public interface ICustomerService
  {
    IList<Customer> GetCustomerWhitDebt();
  }
```

- Esta interfaz la implementaremos de la siguiente forma

```
public class CustomerService : ICustomerService
  {
    IRepository<Customer> repository;

    public CustomerService(IRepository<Customer> repository)
    {
      this.repository = repository;
    }
    public IList<Customer> GetCustomerWhitDebt()
    {
      var customerCollection = this.repository.GetAll();
      return customerCollection.Where(x => x.Saldo > 1000).
ToList();
    }
  }
```

NOTA: Por lo general cuando programamos una arquitectura de N-Capas bien en una aplicación de escritorio, móvil o en un entorno web utilizamos un contenedor de inyección de dependencias como puede ser AutoFac. Estos contenedores de forma "mágica" son los encargados de resolver esta inyección de dependencias. Esto en SharePoint no es posible de una forma sencilla y sin que el rendimiento de la Granja de SharePoint se vea afectado por lo que cada vez que invoquemos a este método debemos de proporcionarle la definición de esta interfaz.

Una vez ya tenemos implementada la capa de negocio, lo que queremos hacer es comprobar que esta capa funciona según las especificaciones del cliente. Para hacer esta tarea vamos a utilizar Moq. Es una librería que lo que nos hace es implementarnos un Mock de una interfaz, en la que configuraremos los valores que espera que devuelvan cada método de la interfaz. De esta forma obtenemos varios beneficios:

- Verificamos que nuestra clase funciona tal y como esperamos.
- Desacoplamos las clases, de esta forma hacemos que nuestros desarrollos cumplan los principios SOLID y sean mucho más mantenibles.
- No dependemos de tener un SharePoint instalado para comprobar que nuestro código funciona. Con lo cual

además tenemos mucha más agilidad a la hora de desarrollar, ya que no tenemos que desplegar en SharePoint.

Como utilizar Moq.NET

1.– Creamos un Proyecto de Test utilizando la plantilla de Visual Studio (Proyectos-> Visual C# Test-> Unit Test Project).

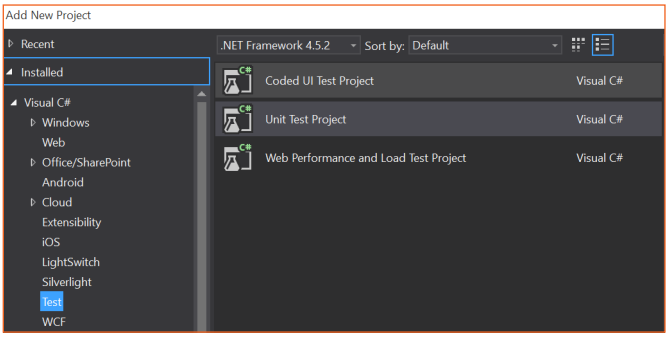


Imagen 3.- Proyecto Unit Test Project.

2.– Instalamos el paquete Nuget Moq.

3.– En el proyecto de Test nos creamos un método Inicializador. Dentro de este inicializador lo que vamos a hacer es crearnos el Mock de nuestra Interfaz IRepository haciendo uso de Moq. Para ello se quedaría un código como el siguiente:

```
IRepository<Customer> repository;
    [TestInitialize]
    public void Init()
    {
        var moqRepository = new Moq.
Mock<IRepository<Customer>>();
        var customerCollection = new List<Customer>
        {
            new Customer { CIF="AAAA", Saldo=2000,
Direccion="Calle ...", Title="COMPARTIMOSS" }
        };
        moqRepository.Setup(moq => moq.GetAll()).
Returns(customerCollection);
        repository = moqRepository.Object;
    }
```

Lo que hace este código es un objeto "simulado" que cada que en nuestro desarrollo se llame al método GetAll, devolverá el valor que hemos introducido. En este caso la variable customerCollection. En el caso de que tengamos la necesidad de que en nuestros métodos tengan varios parámetros se puede configurar que dado un parámetro devuelva unos datos y cuando se le pase otros parámetros se comporte de otra forma. De esta forma podemos testear todas las opciones que tiene nuestra función y hacer nuestra clase más robusta y con menos fallos.

1.– Por último, implementaremos el test que compruebe el correcto funcionamiento. En este caso lo que vamos a verificar que el método "GetCustomerWhitDeubt" devuelve los clientes que tienen un saldo mayor de 1.000. Para ello implementaremos el siguiente Test.

```
[TestMethod]
    public void GetCustomerWithDebt()
    {
        var customerService = new CustomerService(repository);
        var data = customerService.GetCustomerWhitDebt();
        Assert.IsTrue(data.Where(x => x.Saldo > 1000).Any());
    }
```

2.– Ahora ejecutamos el Test y si todo ha ido bien veremos la siguiente pantalla:

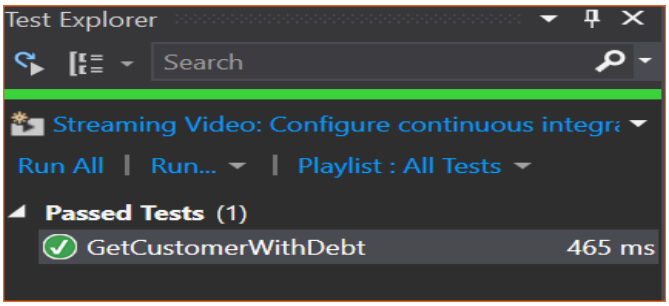


Imagen 4.- Resultado de la ejecución del Test.

Conclusión

Hacer Unit Testing en SharePoint es posible y es muy necesario por muchos motivos:

- Garantizar que se cumplen los requisitos a nivel de negocio.
- Mantenibilidad de la aplicación.
- Facilitar la evolución y el posterior mantenimiento de la aplicación.

Hacer Test no es solo es hacerlo a nivel de código, sino que también es necesario hacerlos a nivel de la interfaz. Por este motivo, cada vez son más importantes los equipos de Q&A. Son los encargados de probar de arriba abajo la aplicación y de dotarle de una calidad a nuestros desarrollos haciendo posible que la aplicación llegue sin errores/bugs al cliente.

Pero además de los equipos de Q&A, los desarrolladores debemos de cumplir con nuestra parte y hacer las pruebas pertinentes para tener un desarrollo de calidad. En SharePoint, ya sea utilizando Enmarcha o no, es posible hacer Test y sin hacer uso de herramientas comerciales por lo que si no haces Test en SharePoint ya no tienes escusa.

ADRIÁN DIAZ CERVERA
Architect Software Lead at Encamina
MVP Office Server and Services
http://blogs.encamina.com/desarrollandosobreshare-point
http://geeks.ms/blogs/adiazcervera
adiaz@encamina.com @AdrianDiaz81

04

Configuración de Enterprise Gateway – Refresco Automático

En el presente artículo se mostrarán los pasos a seguir para la correcta configuración de Gateway en Power BI, y la actualización de información de data sources utilizando dicha Gateway.

Un tema a tener presente es la finalidad del Gateway que proporciona una transferencia de datos rápida y segura entre los datos locales, y servicios. Se puede utilizar un único Gateway con diferentes servicios al mismo tiempo.

Recordar que se debe contar con una cuenta configurada de Power BI PRO y con la posibilidad de loguearse en plataforma Power BI Web.

en este artículo se mostrarán los pasos a seguir para la correcta configuración de Gateway en Power BI

Se recomienda instalar el Gateway en un servidor/máquina que cuente con las siguientes características (servicio):

- El equipo debe permanecer encendido o tener ciclos cortos de apagado (que no coincidan con las actualizaciones programadas).
- Se recomienda que la conexión a internet sea a través de una red física y no a través de redes inalámbricas.

Instalación

Para realizar la instalación del Gateway realizar las siguientes acciones:

1.– Ingresar en el sitio de Power BI e ingresar en Download – Data Gateway

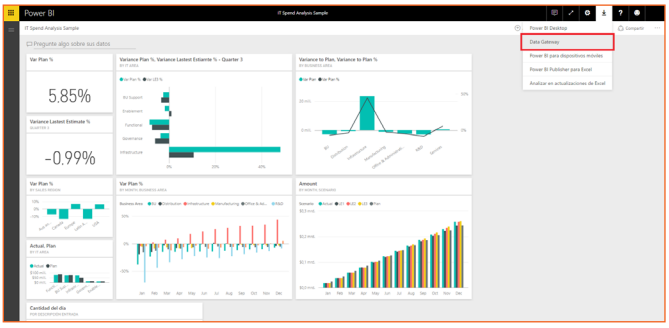


Imagen 1.- Descarga de Data Gateway.

2.– Una vez seleccionada dicha opción, se abrirá una pestaña con la opción para descargar el ejecutable de Gateway

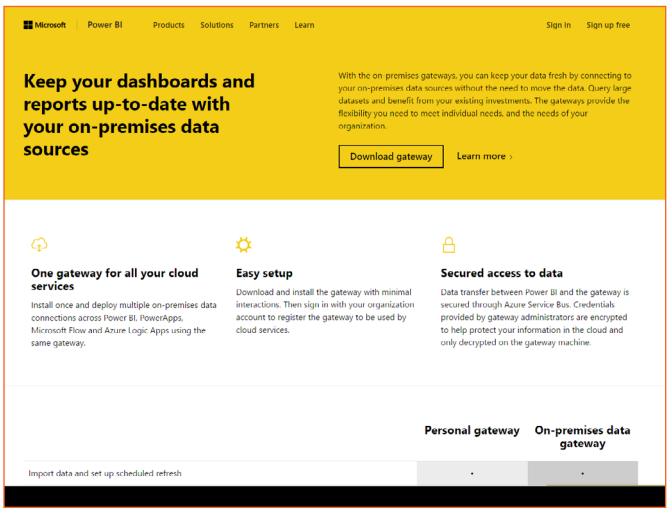


Imagen 2.- Página de descarga del Gateway.

3.– Al ejecutar el programa se ejecutará un wizard de instalación del Gateway On-Premises. En la primera ventana se generarán recomendaciones de equipo necesario para llevar adelante la instalación del Gateway, así como también una explicación de funcionamiento y objetivos del mismo.

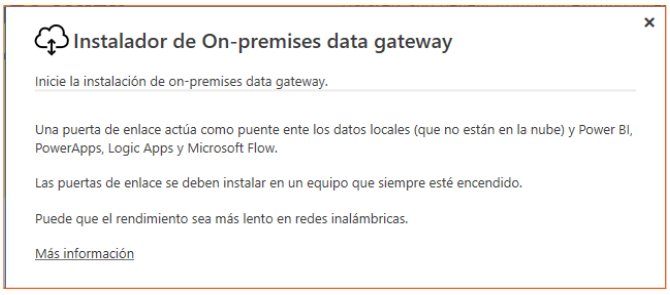


Imagen 3.- Asistente de instalación del gateway.

4.– En la siguiente pantalla se debe seleccionar el tipo de Gateway a instalar, si será Enterprise o Personal. En este caso se seleccionará la opción Enterprise para continuar.

Imagen 4.- Configuración del tipo de puerta de enlace.

5.– Luego comenzará la descarga del instalador de Gateway.

6.– Una vez finalizada la descarga se presentará la siguiente pantalla:

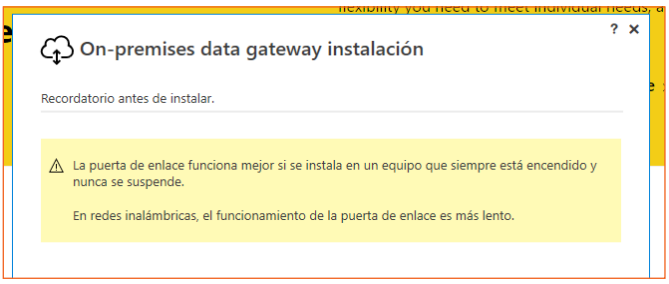


Imagen 5.- Aviso antes de instalar el Gateway.

7.– En la siguiente ventana, deberemos especificar donde se instalará el Gateway y si se aceptan términos de uso y privacidad. Una vez seleccionado el path de destino y aceptado los términos y condición se selección la opción de Instalar.

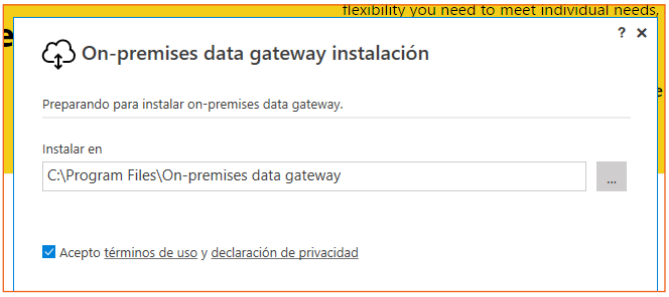


Imagen 6.- Ubicación de instalación del Gateway.

8.– La instalación puede demorar varios minutos. Se reflejará un control de avance en la siguiente pantalla:

Imagen 7.- Proceso de instalación del Gateway.

9.– El siguiente paso será la configuración del Gateway, y para ello nos solicitará que iniciemos sesión (se debe contar con una cuenta de Power BI PRO). Si contamos con los datos necesarios para iniciar sesión, seleccionar la opción Iniciar Sesión.

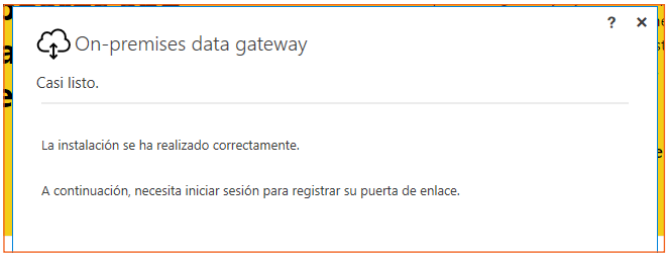


Imagen 8.- Instalación correcta del Gateway.

10.– Una vez iniciada la sesión, debemos proporcionar el nombre al Gateway y una clave de recuperación asociada al mismo (esta clave no puede ser cambiada por lo que se recomienda guardarla en lugar seguro). Una vez completados los datos, elegir la opción de Configuración

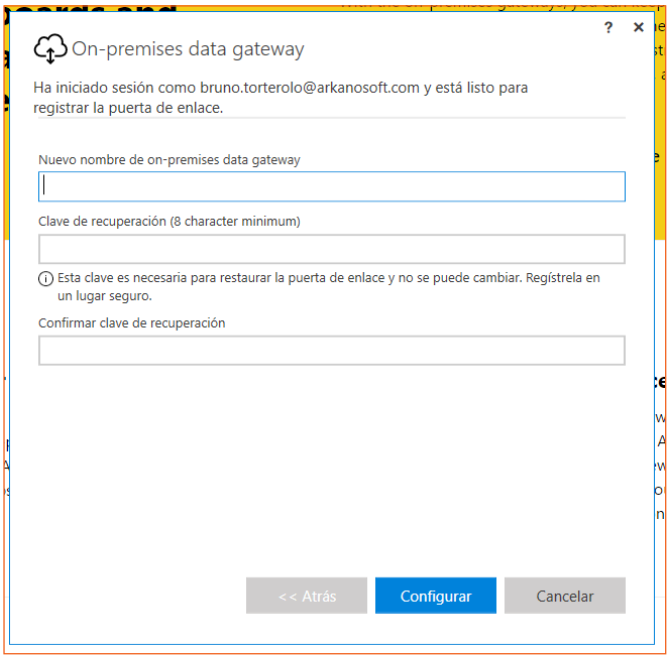


Imagen 9.- Opciones de configuración del Gateway.

1. Al completar la configuración, se mostrará el siguiente cuadro de confirmación. En esta etapa se finaliza la instalación y configuración del Gateway.

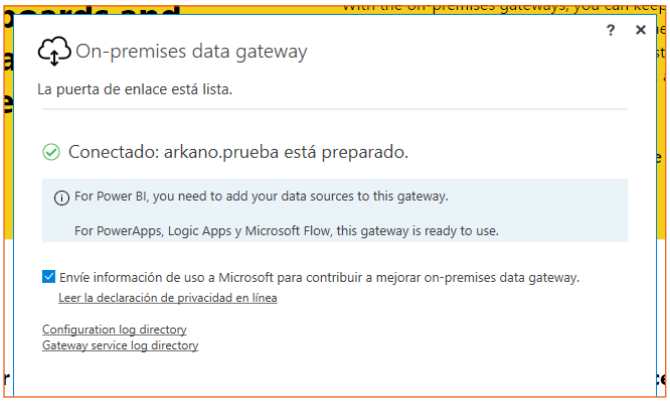


Imagen 10.- Fin de la configuración del Gateway.

Agregando orígenes de datos al Gateway

Una vez configurado el Gateway, se debe asociar los orígenes de datos al mismo. Para asociar los orígenes de datos se debe realizar el siguiente procedimiento:

1.– Ingresar al Sitio de Power BI, y dirigirse en el borde superior derecho a la opción Manage Gateways.

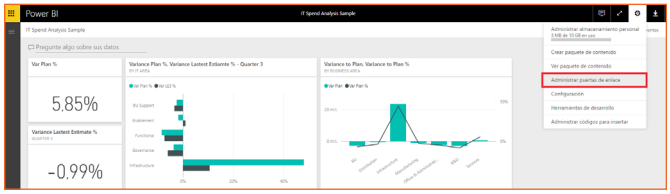


Imagen 11.- Acceso a las opciones de administración del Gateway.

2.– En la pantalla de administración de Gateways, se contará con un listado de gateways instalados y configurados. Para agregar un Data Source a un Gateway se debe seleccionar el mismo y elegir la opción de Add Data Source

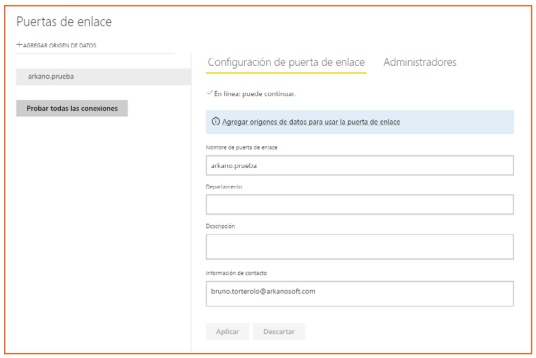


Imagen 12.- Configuración de una puerta de enlace.

3.– En la página de configuración de data source, se debe proporcionar el nombre de data source, y el origen de datos (una vez seleccionada el origen, se habilitarán otros campos a completar dependiendo el origen elegido).

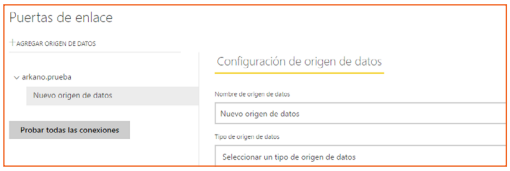


Imagen 13.- Configuración de un origen de datos.

Configuración de Datasets

Al contar con una cuenta de Power BI PRO y un Gateway configurado, podemos establecer un máximo de 8 actualizaciones diarias de los dataset. Para programar dichas actualizaciones, se debe realizar el siguiente procedimiento:

1.– Ingresar en sitio de Power BI, y dirigirse a la opción Settings – Settings

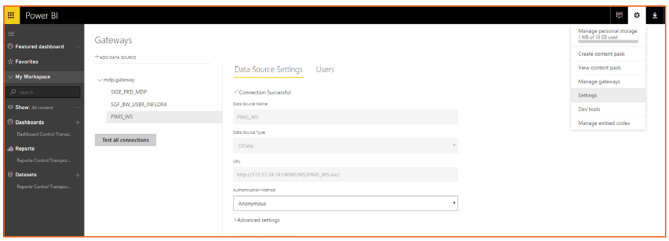


Imagen 14.- Acceso a la configuración de datasets.

2.– Dentro de la página de Settings, dirigirse a la pestaña Datasets

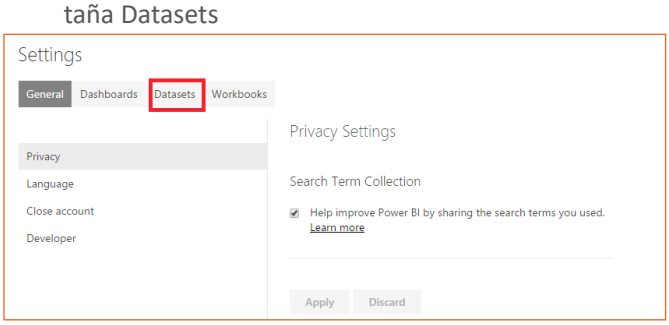


Imagen 15.- Acceso a la configuración de un Dataset.

3.– En la página se listarán los datasets definidos, para los cuales se podrán configurar las siguientes opciones:

- Gateway Connection
- Schedule Refresh
- Q&A and Cortana
- Featured Q&A Questions

Gateway Connection

En Gateway Connection podremos especificar a partir de qué Gateway se actualizará el dataset. Podemos elegir entre uno nuevo (Power BI – Personal) y uno existente (en este ejemplo seleccionamos el creado anteriormente)

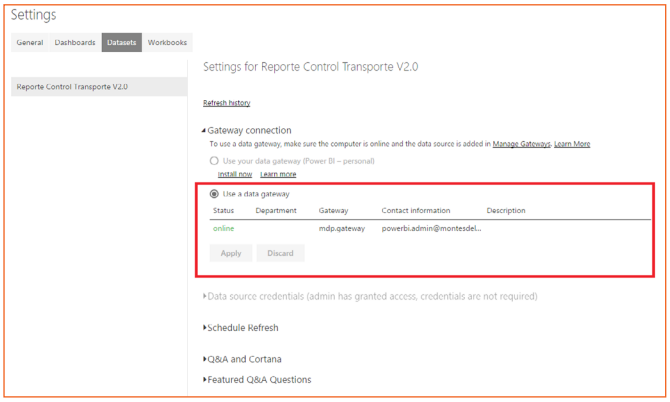


Imagen 16.- Configuración de un Dataset.

podemos establecer un máximo de 8 actualizaciones diarias de los dataset

Schedule Refresh

En este caso se especificará la configuración de periodicidad de actualización de los datos del dataset respecto a los orígenes del Gateway establecido. Podremos especificar actualización diaria o semanal. Tomar en cuenta que se pueden establecer hasta 8 procesos de actualización diaria.

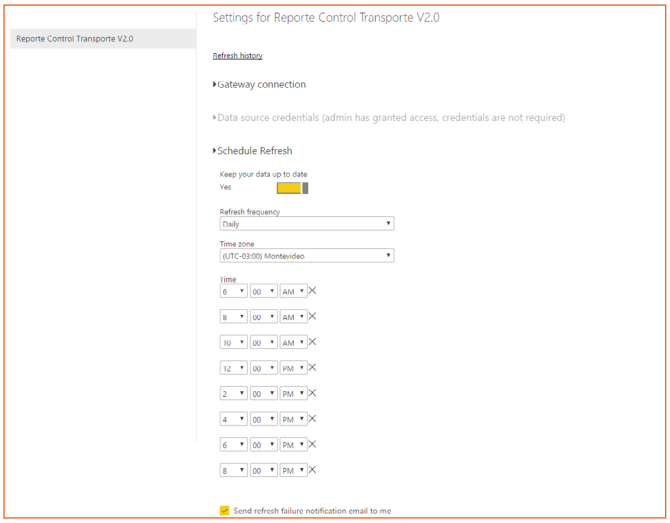


Figura 17.- Configuración del refresco.

Q&A and Cortana

En la sección Q&A and Cortana se puede especificar si permitiremos consultas en lenguaje natural, a través de la utilización de Cortana.

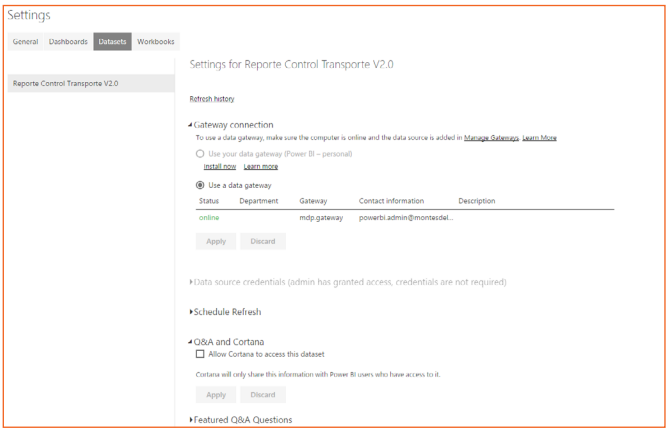


Imagen 18.- Q & A and Cortana.

Featured Q&A Questions

En esta sección podemos especificar un conjunto de preguntas que el usuario puede realizar, de forma de acceder a determinado comportamiento dentro del reporte.

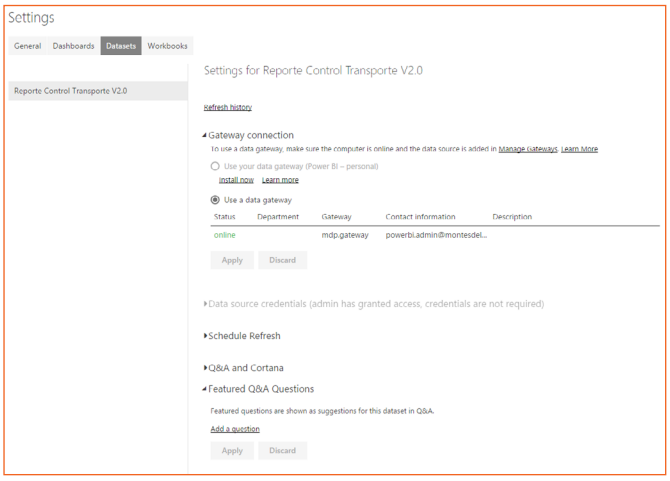


Imagen 19.- Featured Q&A Questions.

GASTON GRUZ
Business Intelligence Technical Manager en Arkano Software
MCSE: SharePoint
MCSA: Office 365

04

Custom Extensibility Handlers para el framework de provisioning del PnP

Office 365
Developer
Patterns & Practices

Continuando la serie de artículos hablando del framework de Provisioning del PnP, veremos cómo crear nuestros propios ExtensibilityHandlers, con los que podremos añadir nuestras propias acciones, tanto al exportar un sitio como plantilla, como al provisionar una plantilla a un nuevo sitio.

Antes de empezar, es necesario recordar un par de cosas con respecto al programa PnP. Primero, sabed que recientemente se ha liberado la reléase de agosto, que incluye numerosas mejoras en la parte de Provisioning, y que va acompañada de un nuevo Schema XML. Tenéis todos los detalles en el enlace https://dev.office.com/blogs/PnP-August-2016-Release

recientemente se ha liberado la release de agosto, que incluye numerosas mejoras en la parte de Provisioning

Algunas de las novedades más destacadas son:

- Soporte para configurar la Navegación, tanto Estructural, como por Metadatos.
- Soporte para romper la herencia de roles en Sub sitios.
- Soporte para User Custom Actions a nivel de listas.
- Soporte para Terminos reusados y "deprecated".
- Eliminar Custom Action (hasta la fecha solo se podían añadir nuevas CustomActions).

La segunda cosa a recordar, sería, tanto el grafico con los diferentes componentes del programa PnP:

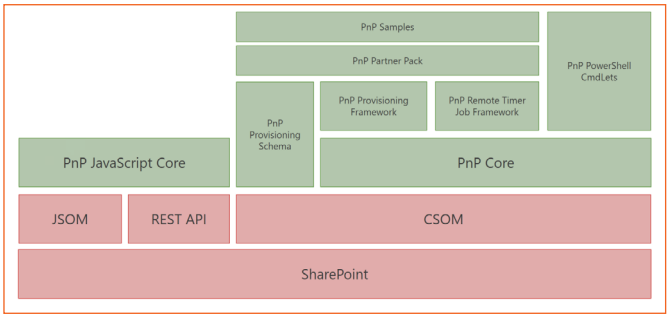


Imagen 1.- Componentes del programa PnP.

Así como el diagrama de arquitectura del proceso de provisioning, donde en el punto 5 de la imagen, podemos ver que se ejecutan los diferentes ExtensibilityHandlers que se hayan configurado en la plantilla:

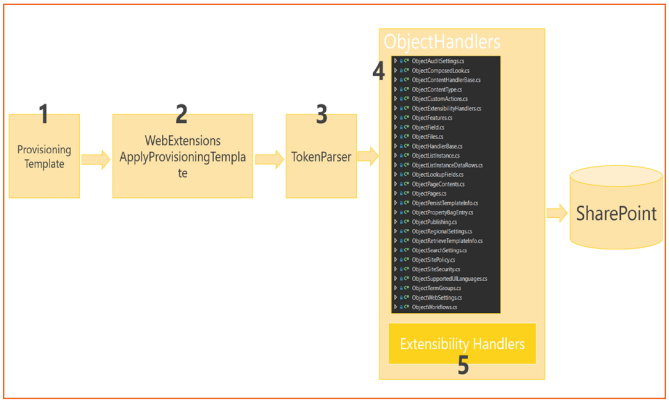


Imagen 2.- Proceso de provisioning.

Como último apunte antes de entrar en el código, cabe destacar que los Extensibility Handlers se pueden ejecutar tanto en el proceso de Provisioning, como en el de Export, (inicialmente, los Extensibilbity Handlers se llamaban Provisioning Extensiblity Providers, y solo se ejecutaban en el proceso de provisioning, por lo que no era posible ejecutar custom code a la hora de exportar un site como plantilla).

Creando nuestro custom Extensibility Handler

En el ejemplo del artículo, vamos a crear un custom Extensiblity Handler, que servirá para hace traza de las diferentes plantillas aplicadas a un sitio. Para ello, crearemos primero una sencilla plantilla PnP en XML, que provisionará una lista personalizada, que servirá como "tabla" de Log. Seguidamente, crearemos nuestro custom Extensibility Provider, que en cada acción de provisioning, insertara un nuevo ítem en la lista de Log, incluyendo cierta información de la plantilla. Finalmente, veremos cómo registrar nuestro Extensibility Provider dentro de la propia plantilla PnP, y ejecutaremos el provisioning para verlo en funcionamiento.

Para crear nuestro Extensibility Handler, basta con crear una nueva biblioteca de clases, y que nuestra clase, implemente la interfaz IProvisioningExtensibilityHandler definida dentro del PnP Core. Tras hacer esto e implementar la interfaz, tendremos el código de la siguiente imagen:

```
namespace PnP.CompartiMOSS.ExtensibilityHandler
{
    0 references
    public class NotifyExtensibilityHandler : IProvisioningExtensibilityHandler
    {
        0 references
        public ProvisioningTemplate Extract(
            ClientContext ctx,
            ProvisioningTemplate template,
            ProvisioningTemplateCreationInformation creationInformation,
            PnPMonitoredScope scope,
            string configurationData)
        {
            throw new NotImplementedException();
        }

        0 references
        public IEnumerable<TokenDefinition> GetTokens(
            ClientContext ctx,
            ProvisioningTemplate template,
            string configurationData)
        {
            throw new NotImplementedException();
        }

        0 references
        public void Provision(
            ClientContext ctx,
            ProvisioningTemplate template,
            ProvisioningTemplateApplyingInformation applyingInformation,
            TokenParser tokenParser,
            PnPMonitoredScope scope,
            string configurationData)
        {
            throw new NotImplementedException();
        }
    }
}
```

El primer método Extract, se ejecutará cuando se haga el Export de un sitio existente a plantilla PnP. En nuestro ejemplo, nos vamos a centrar en la acción de provisioning, y no vamos a hacer nada en el momento de exportar el sitio, así que simplemente, devolveremos la misma plantilla que recibe el método:

```
public ProvisioningTemplate Extract(
    ClientContext ctx,
    ProvisioningTemplate template,
    ProvisioningTemplateCreationInformation creationInformation,
    PnPMonitoredScope scope,
    string configurationData)
{
    return template;
}
```

El método GetTokens nos va a permitir ampliar la colección de Tokens que posteriormente recibirá la acción de Provision. El framework de Provisioning del PnP, hace uso de una clase TokenParser, que nos permite utilizar tokens en la definición de la plantilla. Dichos tokens son resueltos en tiempo de ejecución, y sustituidos por el valor real, dependiendo del contexto. Por ejemplo, podemos hacer uso del token ~sitecollectiontermstoreid que será remplazado por el ID del TermStore de la tenant, y que es muy útil para cuando queremos provisionar Site columns de metadatos administrados. El Framework dispone de muchos tokens, y seguramente cubra todo lo que necesitamos, pero si no es así, podemos utilizar este método para ampliar el listado de Tokens.

nos va a permitir ampliar la colección de Tokens que posteriormente recibirá la acción de Provision

Para el ejemplo, vamos a crear un Token, que será reemplazado por el valor de una PropertyBag del web. Para ello, creamos una nueva clase, derivada de TokenDefinition

```
1 reference
public class WebPropertyBagToken : TokenDefinition
{
    private string propertyBagKey = null;
    0 references
    public WebPropertyBagToken(Web web, string key)
        : base(web, string.Format("{{webpropertybag:{0}}}", key))
    {
        propertyBagKey = key;
    }

    0 references
    public override string GetReplaceValue()
    {
        if (string.IsNullOrEmpty(CacheValue))
        {
            CacheValue = Web.GetPropertyBagValueString(propertyBagKey, String.Empty);
        }

        return CacheValue;
    }
}
```

En el constructor de la clase, simplemente llamamos a su clase base, especificando el patrón de definición del Token: "{{webpropertybag:{0}}}". Esto define como se utilizará el token desde el XML de la plantilla. En este caso, si queremos usar el valor de una PopertyBag con Key "PnPTemplateVersion", desde el XML haremos algo como: "{webpropertybag:PnPTemplateVersion}"

Finalmente, sobrescribimos el método GetReplaceValue() donde se hace el remplazo del token por el verdadero valor. En dicho código, hacemos uso de una Extension del propio PnP, que nos facilita el trabajo para obtener una PropertyBag.

Una vez definido el token, es momento de volver al Extensibility Handler, y utilizarlo:

```
0 references
public IEnumerable<TokenDefinition> GetTokens(
    ClientContext ctx,
    ProvisioningTemplate template,
    string configurationData)
{
    var tokens = new List<TokenDefinition>();

    tokens.Add(new WebPropertyBagToken(ctx.Web, "PnPTemplateVersion"));

    return tokens;
}
```

El último paso es rellenar el método Provision, donde insertaremos el nuevo ítem en la lista de Log

Pero antes de eso, vamos a ver como registrar nuestro custom Handler en la plantilla PnP. Para ello, usaremos el nodo Provider, disponible dentro del schema del PnP, y lo apuntamos a nuestra nueva clase custom Handler:

```
<pnp:Providers>
  <pnp:Provider Enabled="true"
                HandlerType="PnP.CompartiMOSS.ExtensibilityHandler.NotifyExtensibilityHandler,
                PnP.CompartiMOSS.ExtensibilityHandler, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null">
    <pnp:Configuration>
      <Value xmlns="http://schemas.mycompany.com/MyExtensibilityProviderConfiguration">
        {{webpropertybag:PnPTemplateVersion}}
      </Value>
    </pnp:Configuration>
  </pnp:Provider>
</pnp:Providers>
```

Hasta el nodo Configuration, todo sigue el schema del PnP. Dentro de dicho nodo, ya podemos añadir todo el XML que necesitemos, la única condición es que en el nodo raíz, definamos un NameSpace (xmlns), que será el que usaremos desde código para cargar el XDocument. En el ejemplo, lo hemos simplificado al máximo, y tan solo estamos usando el Token previamente definido, para comprobar que todo funciona bien. Dicho valor, lo insertaremos en el ítem de

la lista Log.

Bug en el PnP-Sites-Core dll

Si ponéis un breakpoint en el método GetTokens, os daréis cuenta que dicho método no se está ejecutando nunca. Por lo que he podido comprobar, se trata de un bug en la propia dll del PnP Core. Voy a intentar hacer una Pull Request al proyecto de GitHub para corregir el error, y que esté resuelto para la siguiente reléase de septiembre, pero de no ser así, os pongo aquí como resolverlo, que obviamente pasa por tocar el código de la .dll del PnP, así que, si tenéis que hacerlo, no podréis usar el paquete de Nuget.

Si ponéis un breakpoint en el método GetTokens, os daréis cuenta que dicho método no se está ejecutando nunca

Para resolverlo, primero hay que editar el archivo ExtensiblityManager.cs, y cambiar la cabecera del método ExecuteTokenProviderCallOut

Quedando como:

```
public IEnumerable<TokenDefinition>
ExecuteTokenProviderCallOut(ClientContext ctx,
ExtensibilityHandler provider, ProvisioningTemplate template)
```

También necesitaremos editar el fichero ObjectExtensibilityHandlers.cs y modificar el método AddExtendedTokens cambiando la línea 28 como:

```
foreach (var handler in template.ExtensibilityHandlers.
Union(applyingInformation.ExtensibilityHandlers))
```

Con ambos cambios resolveremos el bug, y nuestro Extensibility Provider funcionara tal y como esperamos.

Volviendo al método de Provision de nuestro custom Handler, tendremos el siguiente código para crear un nuevo ítem en la lista de Log.

```
2 references
public void Provision(
    ClientContext ctx,
    ProvisioningTemplate template,
    ProvisioningTemplateApplyingInformation applyingInformation,
    TokenParser tokenParser,
    PnPMonitoredScope scope,
    string configurationData)
{

    // configurationData es el string con nuestro XML custom de configuracion
    // ademas, en este punto el valor ya ha pasado por el TokenParser
    // por lo que nuestro custom Token ya ha sido resuelto
    string propertyBagValue = GetValueFromXMLString(configurationData);

    var list = ctx.Web.GetListByUrl("Lists/PnPLog");
    ListItemCreationInformation itemCreateInfo = new ListItemCreationInformation();
    ListItem listItem = list.AddItem(itemCreateInfo);
    listItem["Title"] = Guid.NewGuid().ToString();
    listItem["DeployedVersion"] = propertyBagValue;
    listItem["Info"] = "Template applied using PnP Framework. Demo para CompartiMOSS. Autor: Luis Manez";

    listItem.Update();

    ctx.ExecuteQuery();
}
```

Como digo en el comentario, el parámetro de entrada configurationData, contiene el XML personalizado que hemos definido en la plantilla PnP, y además, este string llega con todos los tokens resueltos por el TokenParser, incluido los tokens personalizados que hemos añadido en la implementación del GetTokens.

Una vez creado nuestro Handler, podemos aplicar una plantilla PnP, y ver el resultado. La template PnP utilizada es la siguiente:

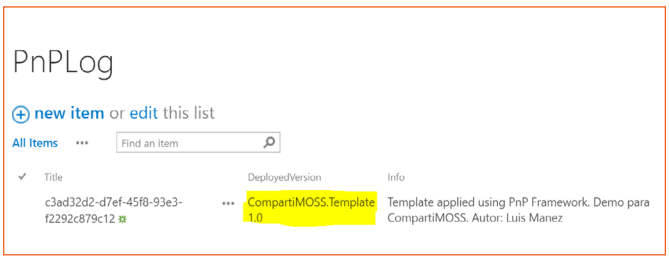
```
<pnp:Provisioning xmlns:pnp="http://schemas.dev.office.com/PnP/2016/05/ProvisioningSchema">
  <pnp:Preferences />
  <pnp:Templates ID="CONTAINER-Lists">

    <pnp:ProvisioningTemplate ID="CompartiMOSS.Lists">

      <pnp:PropertyBagEntries>
        <pnp:PropertyBagEntry Key="PnPTemplateVersion" Value="CompartiMOSS.Template 1.0" Overwrite="true" />
      </pnp:PropertyBagEntries>

      <pnp:Lists>

        <pnp:ListInstance
            Title="PnPLog"
            Description=""
            DocumentTemplate=""
            TemplateType="100"
            Url="Lists/PnPLog"
            MinorVersionLimit="0"
            MaxVersionLimit="0"
            DraftVersionVisibility="0"
            TemplateFeatureID="00bfea71-de22-43b2-a848-c05709900100"
            EnableFolderCreation="false">
          <pnp:ContentTypeBindings>...</pnp:ContentTypeBindings>
          <pnp:Views>...</pnp:Views>
          <pnp:Fields>
            <Field Type="Text" DisplayName="DeployedVersion" Required="FALSE" EnforceUniqueValues="FALSE" Indexed="FALSE" MaxLength="255"
            <Field Type="Note" DisplayName="Info" Required="FALSE" EnforceUniqueValues="FALSE" Indexed="FALSE" NumLines="6" RichText="FAL
          </pnp:Fields>
        </pnp:ListInstance>

      </pnp:Lists>

      <pnp:Providers>
        <pnp:Provider Enabled="true"
                      HandlerType="PnP.CompartiMOSS.ExtensibilityHandler.NotifyExtensibilityHandler, PnP.CompartiMOSS.ExtensibilityHandle
          <pnp:Configuration>
            <Value xmlns="http://schemas.mycompany.com/MyExtensibilityProviderConfiguration">
              {webpropertybag:PnPTemplateVersion}
            </Value>
          </pnp:Configuration>
        </pnp:Provider>
      </pnp:Providers>

    </pnp:ProvisioningTemplate>

  </pnp:Templates>
</pnp:Provisioning>
```

En dicha template se provisiona una property bag con el valor de la versión de la template que estamos aplicando. Además, se provisiona la lista de Log con un par de campos, y finalmente, se registra nuestro custom Handler.

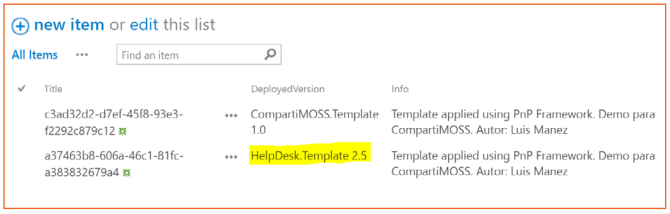
Con el siguiente código podemos aplicar la plantilla desde el archivo XML y provisionarla a un sitio de SharePoint online:

```
AuthenticationManager authManager = new
AuthenticationManager();
var context = authManager.
GetSharePointOnlineAuthenticatedContextTenant(
        "https://tenant.sharepoint.com/sites/demo",
        "UserName",
        "PasswOrd");

XMLFileSystemTemplateProvider provider = new
XMLFileSystemTemplateProvider("C:\\TemplateFolder", "");

var template = provider.GetTemplate("CompartiMOSS.PnP.
HandlersDemo.xml");

context.Web.ApplyProvisioningTemplate(template);
```

Una vez aplicada la plantilla, podemos ver nuestra lista de Log, con un elemento con la versión de la plantilla.



Si ahora re-aplicamos la plantilla, pero cambiamos el valor a la property bag, tenemos un segundo elemento en la lista, con el identificador que le hemos dado a la plantilla:



Nada más, ahora que ya sabemos cómo crear nuestros propios Extensibility Handlers, y siendo que estos sirven tanto para extracción de la plantilla, como cuando se aplica a un sitio, ya no hay excusa para no utilizar el Provisioning Framework del PnP, ya que cualquier necesidad especifica que no cubra el framework, se podrá desarrollar con custom CSOM.

LUIS MAÑEZ
SharePoint / Cloud Solutions Architect en ClearPeople LTD
@luismanez
http://geeks.ms/lmanez/

04

Microsoft Bot Framework para automatizar conversaciones

En el pasado //build/ Microsoft presentó Microsoft Bot Framework, una nueva tecnología que ofrece grandes posibilidades tanto para las empresas como para los desarrolladores ya que permitirá automatizar tareas como por ejemplo atención al cliente siendo capaz de interactuar por canales tan diversos como Skype, chat web, Slack, Telegram, etc.

Construyendo nuestro primer Bot

Dado que aún es una tecnología que está naciendo, los primeros pasos para construir nuestro Bot son algo engorrosos y manuales contrariamente a lo que nos tiene acostumbrados Microsoft, así que vamos a ver de forma muy esquemática cómo comenzar.

1.– Instalar VS 2015 (vale la versión Community).
2.– Instalar la plantilla de Bots que podemos encontrar aquí.
3.– Descomprimir el contenido del archivo .zip que nos descargamos en la carpeta "%USERPROFILE%\Documents\Visual Studio 2015\Templates\ProjectTemplates\Visual C#\"
4.– Abrir Visual Studio.
5.– Crear un proyecto usando la plantilla "Aplicación Bot".

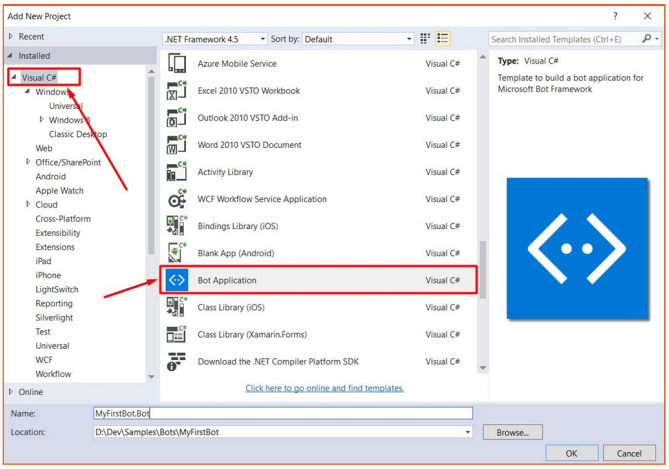


Imagen 1.- Plantilla de proyecto para aplicación Bot.

Estructura del proyecto

El proyecto, como podremos observar, se corresponde con WebApi lo que, a todas luces, es lógico dado que un Bot no será más que un servicio que se comunicará con los usuarios, respondiendo ante determinadas acciones que el controlador correspondiente, por defecto MessagesController, se encargará de procesar.

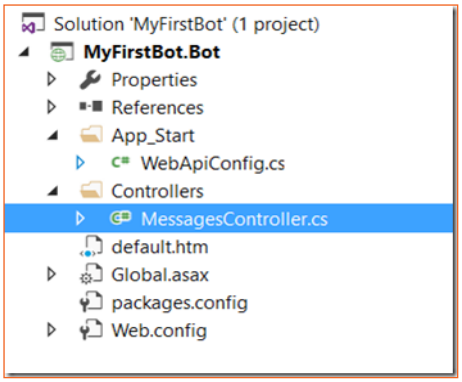


Imagen 2.- Estructura del proyecto de aplicación Bot.

Como se puede observar en la imagen 2, disponemos también de una página por defecto, default.htm, que será la página donde deberemos definir los términos de uso además de indicarnos información para el registro de nuestro Bot para que pueda ser usado.

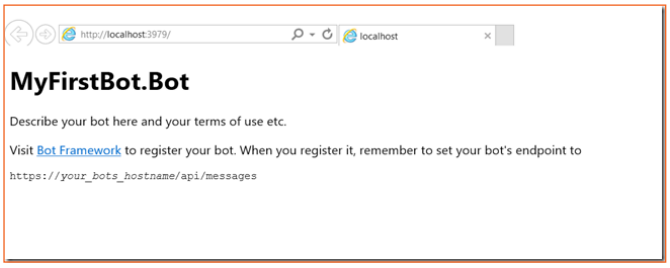


Imagen 3.- Página de inicio de la aplicación Bot.

Testeando el Bot localmente

Para poder testear el Bot, disponemos de un emulador, Bot Framework Emulator, que podremos descargarnos desde https://aka.ms/bf-bc-emulator que nos permitirá comunicarnos con nuestro Bot de diversas formas como, por ejemplo, simular una conversación entre múltiples usuarios, testear diferentes tipos de comunicación, cambiar el lenguaje de comunicación, etc.

en el pasado //build/ Microsoft presentó Microsoft Bot Framework, una nueva tecnología que ofrece grandes posibilidades

Para empezar, tenemos que establecer los parámetros correctos de comunicación con nuestro Bot de la siguiente forma:

- Local Port: 9000.
- Emulator Url: http://localhost:9000/
- Bot Url: Bot Url: La dirección en la que se encuentra publicado el Bot que, en caso de realizar una prueba en local como la que estamos tratando, será http://localhost:{PORT}/api/messages. Para conocer el puerto en el que se está ejecutando, basta con mirar la dirección resultante a la hora de ejecutar el proyecto desde Visual Studio o, simplemente configurarlo en el proyecto tal y como se ve en la siguiente imagen.
- Microsoft App Id: El id de la aplicación relacionada con el Bot. Si lo estamos ejecutando en local y no establecemos nada en nuestro archivo de configuración "web.config", debemos dejarlo vacío.
- Microsoft App Password: El password de acceso a la aplicación relacionada con el Bot. Si lo estamos ejecutando en local y no establecemos nada en nuestro archivo de configuración "web.config", debemos dejarlo vacío.

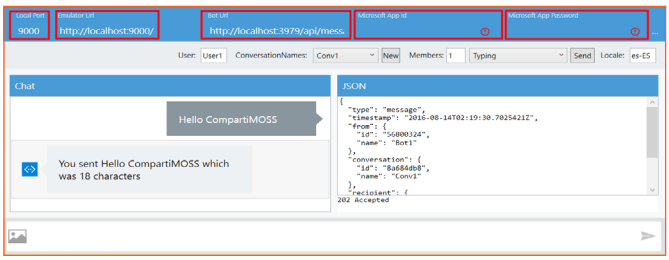


Imagen 4.- Bot Framework Emulator.

Por defecto, la plantilla de aplicación Bot viene configurada para respondernos con una cuenta del número de caracteres que hemos escrito como mensaje, además, en el panel derecho podemos visualizar el mensaje JSON que comunica el Bot.

Publicación del Bot

Una vez hayamos desarrollado la funcionalidad de nuestro Bot y testeado que todo funciona correctamente, tendremos que publicarlo para que pueda ser consumido por los usuarios.

Tal y como ya habíamos comentado, el Bot no es más que una aplicación WebApi y, por lo tanto, se publicará como Azure Web App.

1.– Seleccionamos publicar desde el menú contextual del proyecto.

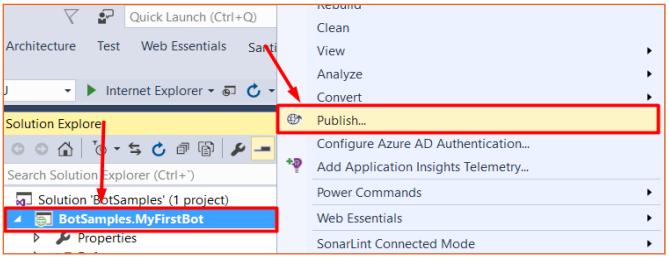


Imagen 5.- Menú contextual del proyecto.

2.– Seleccionamos Microsoft Azure App Service.

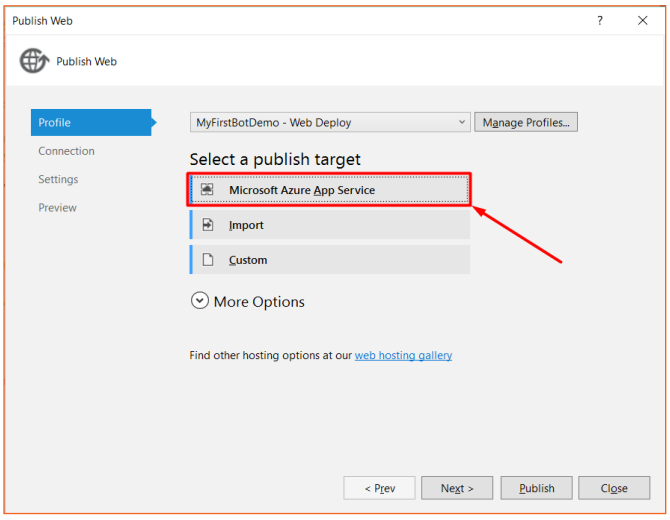


Imagen 6.- Selección del destino de la publicación.

3.– Si ya disponemos de una Web App creada, seleccionamos la cuenta, la suscripción, un grupo de recursos y finalmente la Web App donde queremos publicar el Bot. En caso contrario, seleccionamos nuevo.

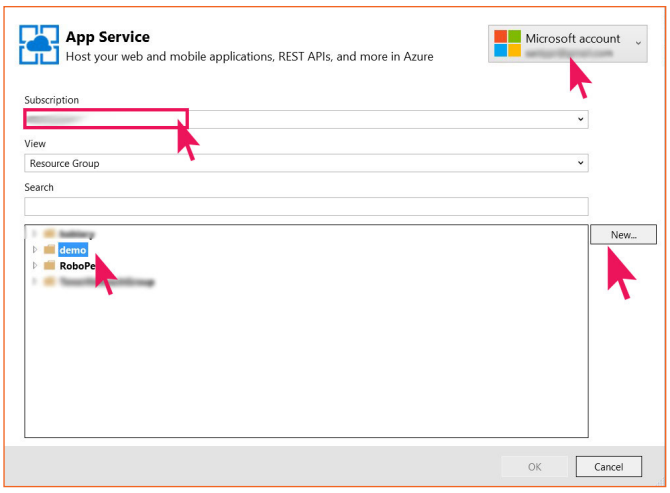


Imagen 7.- Selección de Web App.

4.– En caso de que hayamos seleccionado crear una nueva Web App, tendremos que seleccionar una suscripción, seleccionar o crear un grupo de recursos, seleccionar o crear un plan de servicio y establecer un nombre para la Web App.

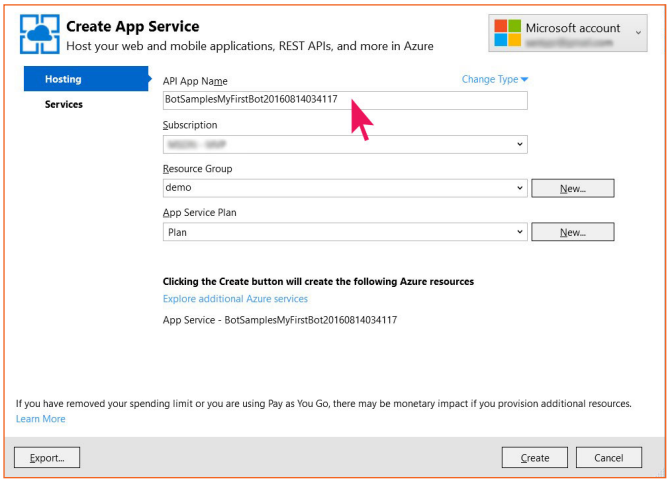


Imagen 8.- Creación de Web App.

5.– Una vez creada la Web App, tan sólo tendremos que publicar.

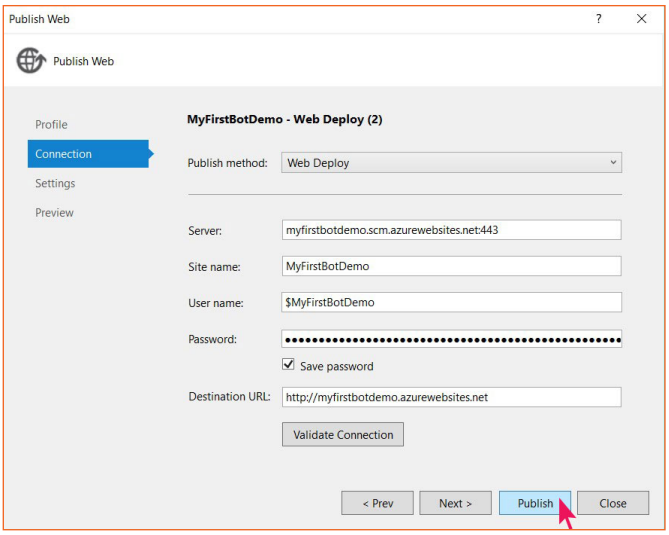
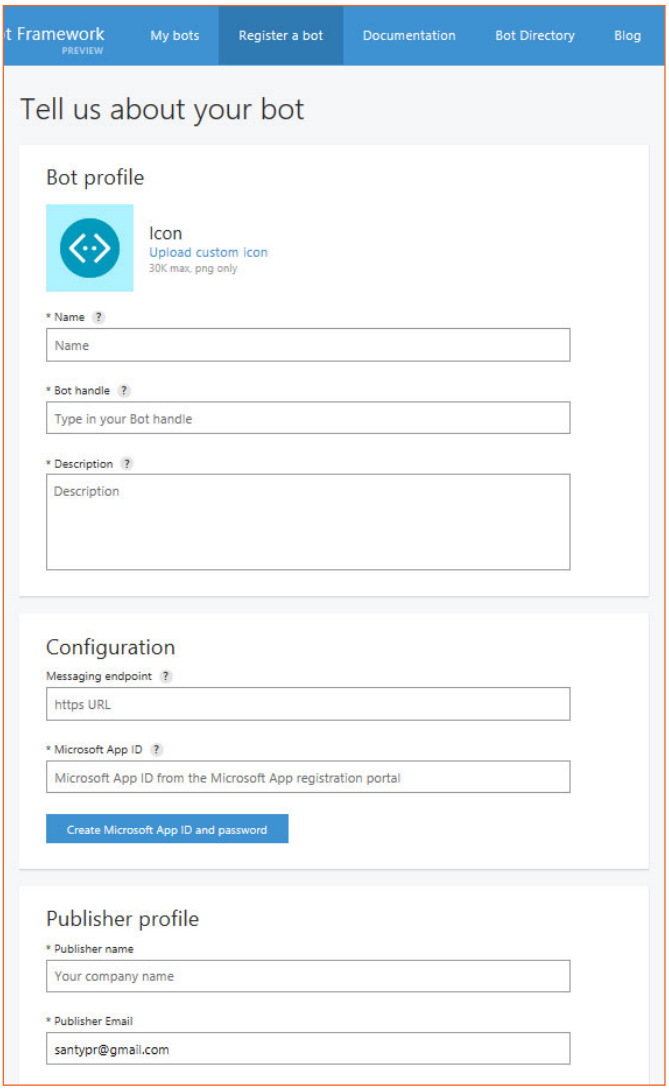


Imagen 9.- Publicación de la aplicación.

Registrando el Bot

No será suficiente con publicarlo, sino que, además, habrá que registrarlo en el propio servicio de Bots desde la url https://dev.Botframework.com/Bots/new. En este formulario debemos rellenar todos los campos para que nuestro Bot sea accesible.



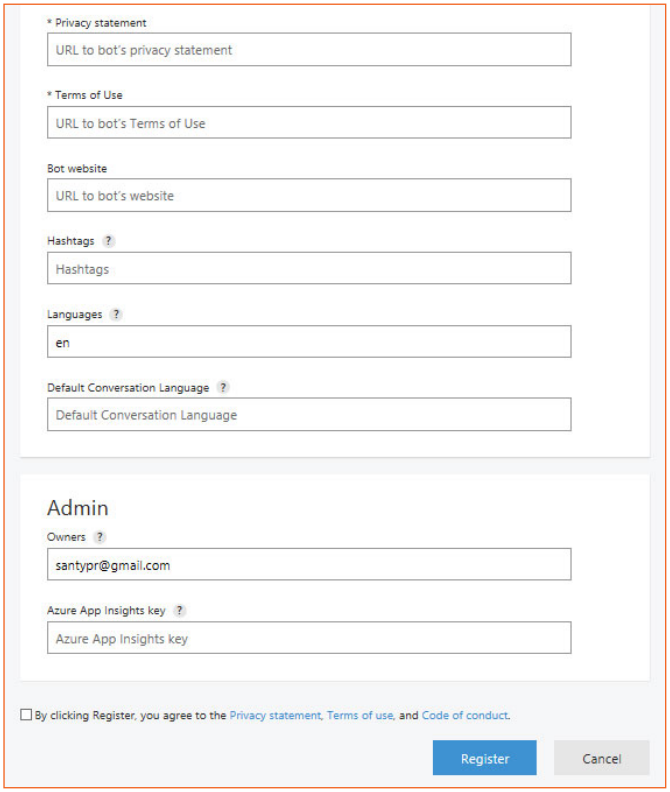


Imagen 10.- Formulario de registro del Bot.

como podremos observar, se corresponde con WebApi lo que, a todas luces, es lógico

Es importante tener en cuenta que para publicarlo necesitamos crear un Microsoft App Id y su password correspondiente mediante el Botón "Create Microsoft App Id and Password" que estará directamente relacionado con el nombre de nuestro Bot, establecido en el primer campo del formulario. Una vez seleccionemos el Botón, nos aparecerá otro formulario donde podremos generar la contraseña.

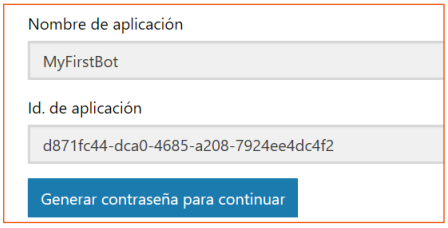


Imagen 11.- Creación de Id y password de la aplicación.

No debemos perder estos datos ya que habrá que introducirlos en el archivo "web.config" para poder comunicarnos con el Bot.

```
<appSettings>
  <!-- update these with your BotId, Microsoft App Id and your
Microsoft App Password-->
  <add key="BotId" value="[YourBotId]" />
  <add key="MicrosoftAppId" value="[YourBotAppId]" />
  <add key="MicrosoftAppPassword"
value="[YourBotAppPassword]" />
</appSettings>
```

Una vez completado el formulario de registro, podremos ver nuestro Bot en la pestaña "My Bots" o en la url https://dev.Botframework.com/Bots.

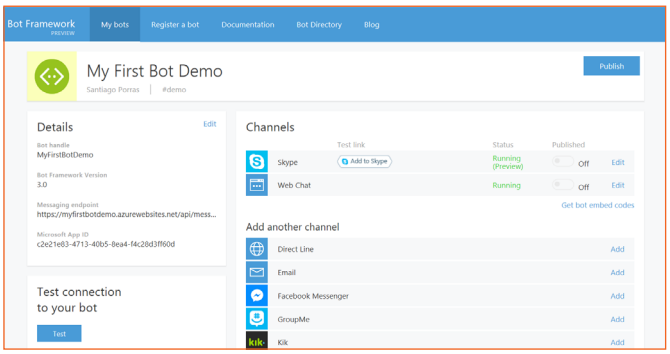


Imagen 12.- Detalles del Bot.

Por último, debemos volver a publicar la aplicación después de haber modificado el archivo "web.config" con los valores anteriormente indicados.

Una vez publicado de nuevo el Bot con la actualización de archivo "web.config", podremos testear por fin la conexión al Bot que, de ir todo correcto, nos mostrará un mensaje de confirmación como en la imagen

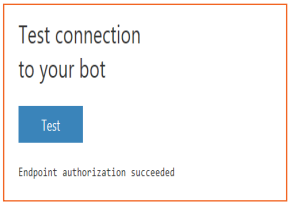


Imagen 13.- Test de conexión con el Bot.

Testeando el Bot en remoto

También es posible testear de forma remota un Bot publicado, pero para ello tendremos que usar otra herramienta que nos permita comunicar el emulador con nuestro Bot. Microsoft recomienda el uso de ngrok, una herramienta de línea de comandos que nos permite exponer servidores locales, ya que es muy sencillo de usar. Los pasos a seguir para poder testear el Bot son los siguientes:

1.– Descarga de ngrok: Podemos descargar ngrok desde su sitio web https://ngrok.com/.

2.– Exponer el emulador por medio de ngrok ejecutando un simple comando:
ngrok http -host-header=rewrite 9000

3.– Establecer el valor de "Emulator Url" del Bot Framework Emulator con la url https que nos indica ngrok.

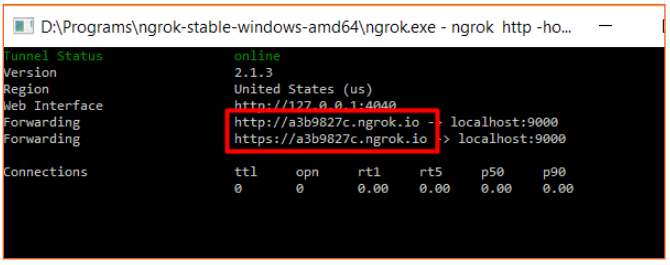


Imagen 14.- Url con la que ngrok expone el emulador.

4.– Volver a configurar el emulador con los parámetros correspondientes para Bot Url, Microsoft App Id y Microsoft App Password.

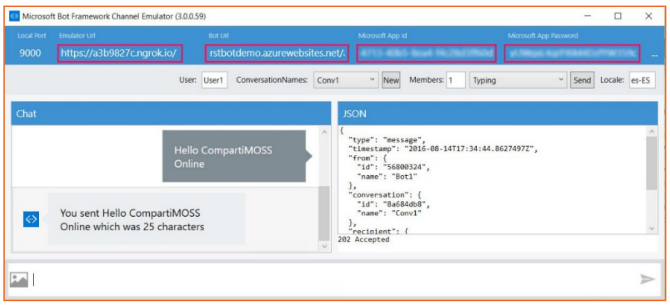


Imagen 15.- Emulador conectado al bot de forma remota.

Integración con Skype

Por defecto, los Bots están preparados para integrarse con Skype y como Web Chat, así que será muy sencillo comenzar a interactuar con ellos.

En primer lugar, debemos configurar el bot según las características que queremos que tenga para interactuar con Skype.

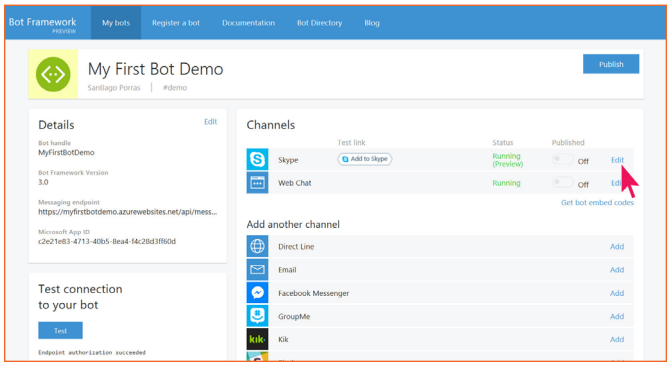


Imagen 16.- Configurar el Bot para funcionar con Skype.

En el formulario debemos activar aquellas características que nos interesen y para las que hayamos preparado el Bot.

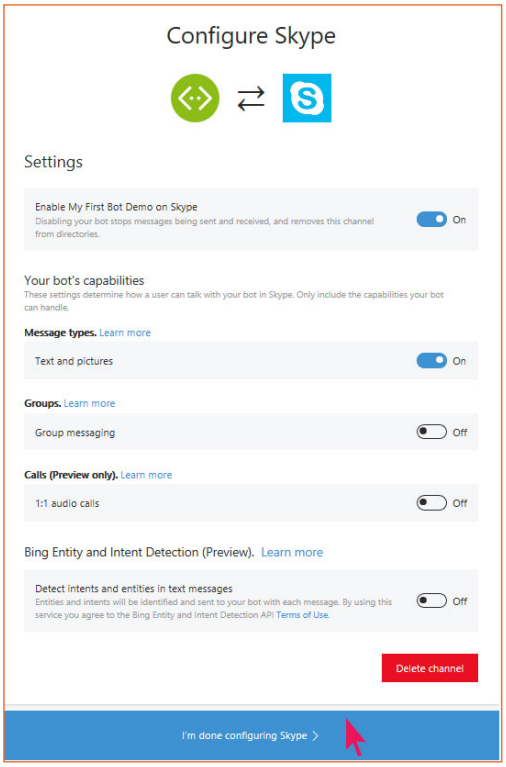


Imagen 17.- Configuración del Bot.

Una vez configurado el Bot, procederemos a añadirlo como contacto en Skype y así poder interactuar con él. En el panel de detalles del Bot, disponemos de un botón "Add to Skype" que nos llevará a la página de aceptación de permisos como podemos ver en la imagen 19.

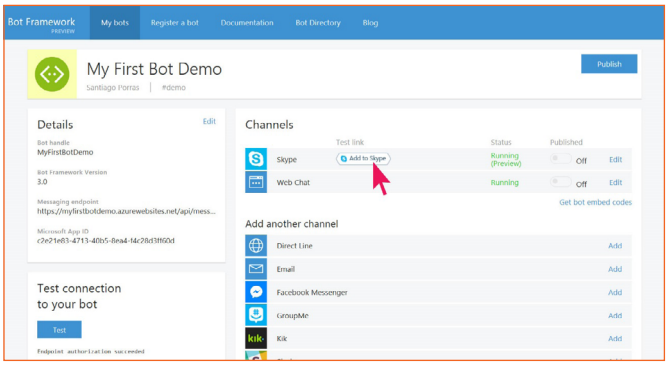


Imagen 18.- Añadir Bot a Skype.

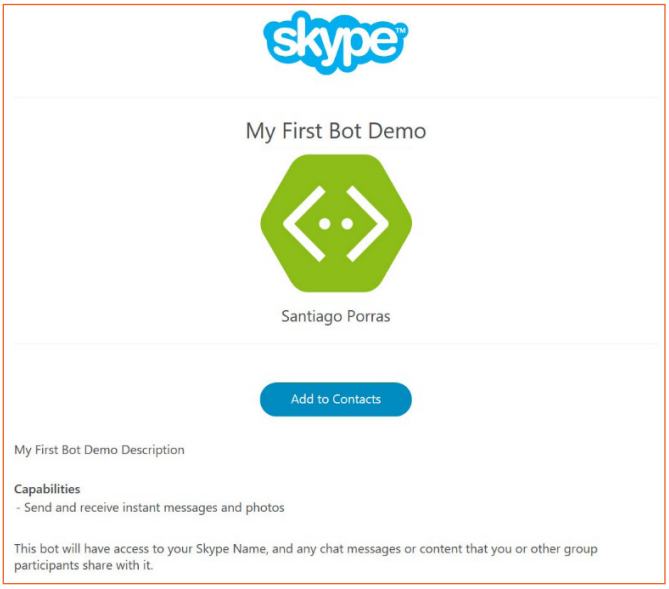


Imagen 19.- Aceptación de permisos.

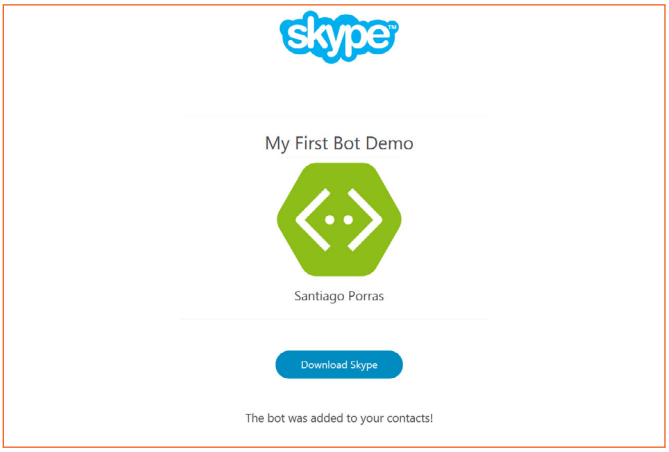


Imagen 20.- Bot añadido correctamente.

por defecto, los Bots están preparados para integrarse con Skype

Una vez hemos añadido nuestro Bot a Skype, podremos interactuar con él como si de otra conversación se tratara. De esta forma, si hemos desarrollado nuestro Bot para que responda a determinadas palabras clave, podremos hacer que mantenga una conversación guiada con los usuarios.

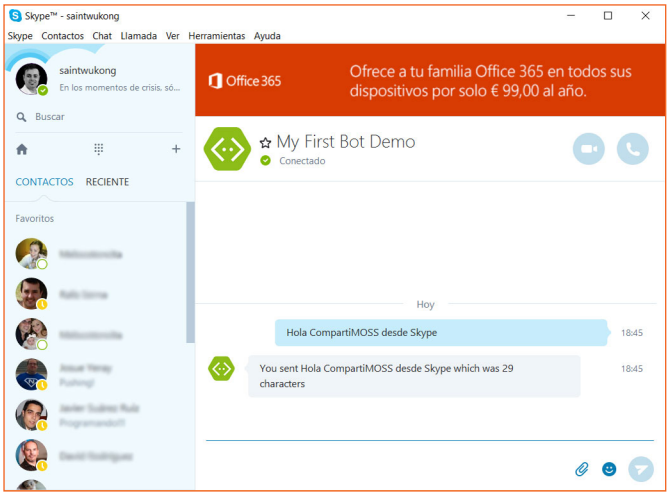


Imagen 21.- Conversación con el Bot a través de Skype.

Finalmente, podremos embeber una conexión a nuestro Bot gracias a la opción "Get bot embed codes" que nos aportará el código necesario para integrarlo en un sitio web y que así los usuarios puedan conectarse de forma sencilla.

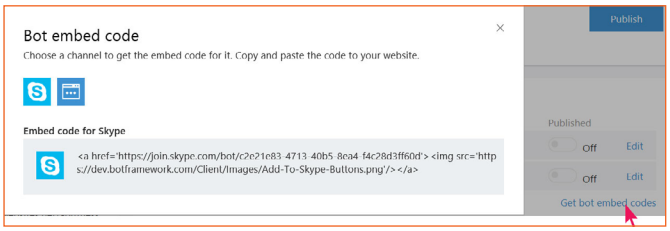


Imagen 22.- Código para embeber el Bot.

Conclusiones

Microsoft ha puesto en nuestras manos la posibilidad de automatizar conversaciones con los usuarios de una forma realmente sencilla y que, de esta forma, ciertas tareas como por ejemplo la atención al cliente puedan dar un salto de calidad.

Referencias y más información

- Documentación oficial: https://docs.botframework.com/en-us/skype/getting-started/
- Documentación ngrok: https://ngrok.com/docs
- Skype bots: https://developer.microsoft.com/en-us/skype/bots

SANTIAGO PORRAS RODRÍGUEZ
Mobile & Cloud Experience Lead at Encamina
MVP Windows Platform Development
MVP Visual Studio Technologies
http://blogs.encamina.com/en-tu-casa-o-en-la-mia/
http://geeks.ms/santypr
@saintwukong

04

SharePoint y Azure – Azure WebJobs

Introducción

Muchos de los servicios ofrecidos por Azure se pueden utilizar para ampliar y mejorar el funcionamiento de SharePoint, tanto on-premises como en la nube. Uno de los servicios que ofrece Azure, parte de Azure Apps, es «Web Apps», que es una solución de Plataforma como Servicio (PaaS) que proporciona toda la infraestructura física más el software de base para servir como host de programas Web. Una de las alternativas de Web Apps es WebJobs, que utiliza toda la infraestructura de Azure Apps para ejecutar programas de forma automática basado en un esquema de tiempo.

Como funciona Azure WebJobs

Probablemente el servicio más importante de Azure para desarrollo de nuevas aplicaciones es Azure Apps, que consta a su vez de cuatro servicios: Web Apps, Mobile Apps, API Apps y Logic Apps. Por medio de Web Apps se pueden crear aplicaciones Web (ASP.NET, JavaScript, PHP, etc.) y hostearlas en Azure. Azure ofrece toda la infraestructura de Máquinas Virtuales, perfectamente configuradas y con todas las posibilidades de extensibilidad, mantenimiento y escalabilidad, de tal forma que lo único que es necesario es configurar el servicio y subir el código de la aplicación.

Web Apps permite crear dos tipos de aplicaciones: las aplicaciones que funcionan con una interface de usuario y realizan algún tipo de trabajo tal como las que se utilizan en sitios de Internet, y WebJobs, aplicaciones que no disponen de una interface de usuario y se ejecutan automáticamente basado en un esquema de tiempo (cada minuto, cada hora, un determinado día, continuamente, etc.). En SharePoint, los WebJobs se pueden comparar a cómo funcionan los Timer Jobs de SharePoint: realizan trabajos temporalizados de forma completamente automática y sin intervención de los usuarios.

Azure WebJobs se pueden utilizar en conjunción con SharePoint en dos escenarios típicos:

- En SharePoint OnPremises para mover procesos que utilizan considerables cargas de recursos de los servidores de la granja (CPU y RAM) hacia servidores externos en Azure, que se pueden escalar de forma muy sencilla. También hay que hacer notar que la programación de Azure WebJobs es considerablemente más fácil que la programación de Timer Jobs de SharePoint.
- En SharePoint Online no es posible crear Timer Jobs personalizados, por lo que Azure WebJobs constituyen su reemplazo.

Un WebJob no es más que un ejecutable del tipo cmd, bat, exe (.NET), ps1, sh, php, py, js o jar. Para el control del tiempo de ejecución se utiliza la infraestructura de Scheduling de Azure.

el corazón de las herramientas de colaboración y manejo de conocimiento

Creación y configuración de un WebJob en Azure

WebJobs se crean y configuran de la misma forma que Web Apps en Azure. También todas sus posibilidades de tipo de infraestructura a utilizar, escalabilidad, etc., son exactamente las mismas. Aunque toda la configuración se puede realizar desde Visual Studio, es recomendable hacerla antes de comenzar a programar y directamente desde el Portal de Azure pues se tiene mucho mejor control sobre los parámetros a utilizar. La creación que se indica a continuación también se puede automatizar por medio de PowerShell o utilizando un Template de Azure.

1.– Desde el Portal de administración de Azure (http://portal.azure.com) seleccione “Resource groups” => “Add”. Defina un nombre para el Grupo de Recursos, la suscripción a utilizar y la localización deseada.
2.– Una Web App necesita tener un “App Service Plan”, que define las características y capacidades que todas las Web Apps en el Grupo de Recursos van a compartir. Seleccione el Grupo de Recursos creado en el punto anterior, utilice el botón de “Add” y en la casilla de búsqueda escriba “app service plan” y seleccione “App Service Plan” en la lista resultados. Utilice el botón de “Create” en la nueva ventana.
3.– En la ventana de creación del Service Plan indique un nombre, la suscripción a utilizar y el Grupo de Recursos (si no ha sido seleccionado automáticamente). La localización del Servicio y el tipo de Máquina Virtual a utilizar también se pueden elegir o utilizar los predefinidos. Note que hay un “tier” que es gratis, el “F1”, aunque ofrece la menor capacidad y algu-

na funcionalidad menos que los otros tiers.

4.– Regrese a la ventana del Grupo de Recursos creado en el punto uno y utilice el botón de “Add”. Escriba “storage account” en la casilla de búsqueda y seleccione “Storage account – Web + Mobile” y “Create” en la ventana que aparece. Un Storage Account no es estrictamente necesario para el funcionamiento de un WebJob (no es indispensable), pero si se desean ven los logs creados por el WebJob, es necesario tenerlo pues Azure escribe todas las salidas del Job en un Blob del Azure Storage.

5.– Defina un nombre (todo en minúsculas y sin signos) para el nombre del Storage, la suscripción, Grupo de Recursos y Localización; utilice todos los otros parámetros por defecto.

6.– Regrese de nuevo a la ventana del Grupo de Recursos, utilice el botón de “Add”, escriba “web app” en la casilla de búsqueda y seleccione “Web App” y “Create”. Defina un nombre, seleccione la Suscripción y el Grupo de Recursos y seleccione el Service Plan creado en el punto tres. Azure crea una instancia de “Application Insights” automáticamente por si es necesario monitorizar el funcionamiento de la aplicación web (si no es necesario el monitoreo, se puede eliminar sin problemas). Al final, el Grupo de Recursos aparecerá con los recursos creados:

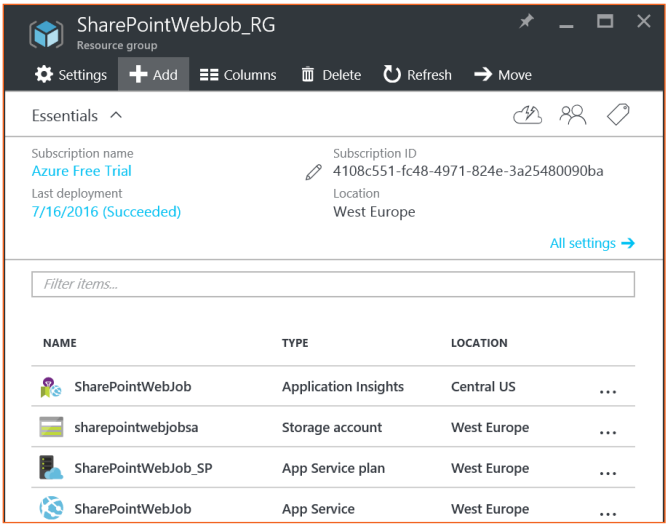


Imagen 1.- Detalle de los elementos creados.

Ejemplo de utilización de Azure WebJobs con SharePoint

El siguiente ejemplo simplemente lee el número de documentos en una Biblioteca de SharePoint. Aunque extremadamente sencillo en cuanto a funcionalidad, la idea del ejemplo es mostrar cómo crear y publicar un Azure WebJob y cuáles son las posibilidades de debugging y loging, no crear nueva funcionalidad. En el ejemplo se utiliza SharePoint Online, pero el código se puede utilizar sin ningún problema con una granja de SharePoint OnPremises, cambiando solamente el URL del servidor.

7.– Inicie Visual Studio (versión 2015 con Update 3 en el ejemplo, pero también se puede utilizar Visual Studio 2013).

8.– Si Visual Studio no tiene instalado el Azure SDK, es necesario descargarlo desde el sitio de Microsoft (https://azure.microsoft.com/en-us/downloads/) e instalarlo en Visual Studio.

9.– Seleccione en Visual Studio “New Project” y desde la sección “Visual C#” – “Cloud” seleccione “Azure WebJob”. Asígnele un nombre al proyecto. Esta plantilla crea una aplicación de consola con todas las referencias a dlls de Azure necesarias.

10.– La forma más fácil para agregar todas las referencias necesarias para poder trabajar con SharePoint Online es utilizar NuGet. En Visual Studio seleccione “Tools => NuGet Package Manager => Manage NuGet Packages for Solution => Browse” y busque por “app for sharepoint online web toolkit”. Seleccione “AppForSharePointOnlineWebToolkit” e instale el NuGet. Si se está trabajando con SharePoint OnPremises, se puede utilizar el NuGet “AppForSharePointWebToolkit”, que produce los mismos resultados. Cuando la instalación termina, el NuGet descarga todos los dlls necesarios y crea los archivos “SharePointContext.cs” y “TokenHelper.cs” en el proyecto.

11.– En “Program.cs” elimine las líneas 17 (var host = new JobHost();) y 19 (host.RunAndBlock();) del código fuente creado por defecto pues no son necesarias para este ejemplo.

12.– Los datos para poderse logear en SharePoint se van a almacenar en el archivo de configuración del proyecto por simplificar el ejemplo, aunque esta no sea una buena práctica, y menos manteniendo la clave en texto simple. Una mejor opción es utilizar el Directorio Activo, si tanto Azure como SharePoint utilizan el mismo servicio, o el token de autorización de OAuth. Abra el archivo “App.config” y agregue las siguientes líneas al final del archivo, entre las líneas “</runtime>” y “</configuration>”:

```
 </runtime>
 <appSettings>
  <add key=”CuentaSharePoint” value=”usuario@dominio.
onmicrosoft.com”/>
  <add key=”ClaveSharePoint” value=”clave”/>
 </appSettings>
</configuration>
```

13.– En el mismo archivo “App.config”, configure la cadena de conexión de la propiedad “AzureWebJobsDashboard” que se encuentra al principio del código. El parámetro “connectionString” es de la forma:

```
connectionString=”DefaultEndpointsProtocol=https;Account-
Name=sharepointwebjobsa;AccountKey=s/JLuH/OtUYkM0zm-
6b3vPK0Ro6A/rWTWXrsnHUTzKrERZkChXcDuDFEgb9GYh-
59dR”
```

En donde “AccountName” es el nombre de la instancia de Azure Storage creado en el punto 4 y 5. “AccountKey” se puede encontrar en el portal de Azure, seleccionando el Storage creado y yendo a “All settings” => “Access keys” => “key1”.

La configuración de "AzureWebJobsStorage" se puede utilizar si es necesario usar un depósito de información adicional de Azure. El "AzureWebJobsDashboard" se utiliza para guardar los logs del WebJob, como se indicó anteriormente

los servicios ofrecidos por Azure se pueden utilizar para ampliar y mejorar el funcionamiento de SharePoint

14.– Agregue las siguientes directivas using en la cabecera de "Program.cs":

```
using Microsoft.SharePoint.Client;
using System.Security;
using System.Configuration;
```

Modifique el método "Main" para que contenga el siguiente código fuente:

```
static void Main()
{
  string mySharePointSiteUrl = "https://dominio.sharepoint.
com/sites/test01";
  string mySharePointBiblioteca = "Libreria Uno";
  string myCuenta = ConfigurationManager.AppSettings["Cuen-
taSharePoint"];
  string myClave = ConfigurationManager.AppSettings["Clave-
SharePoint"];

  using (ClientContext myContext = new ClientContex-
t(mySharePointSiteUrl))
  {
    SecureString myClaveSegura = new SecureString();
    foreach (char unChar in myClave)
    {
      myClaveSegura.AppendChar(unChar);
    }

    myContext.AuthenticationMode = ClientAuthentication-
Mode.Default;
    myContext.Credentials = new SharePointOnlineCreden-
tials(myCuenta, myClaveSegura);

    try
    {
      List myLista = myContext.Web.Lists.GetByTitle(myShare-
PointBiblioteca);
      myContext.Load(myLista);
      myContext.ExecuteQuery();

      if (myLista != null && myLista.ItemCount > 0)
      {
        Console.WriteLine(myLista.Title.ToString() + " tiene " +
myLista.ItemCount + " documentos");
      }
      else
      {
        Console.WriteLine("No hay documentos");
      }
    }
    catch (Exception ex)
    {
      Console.WriteLine("Error: " + ex.ToString());
    }
  }
}
```

Al inicio de la rutina se definen cuatro variables que contienen el URL del sitio con la Biblioteca de SharePoint a examinar, el nombre de la Biblioteca misma y los valores de la cuenta para acceder a SharePoint. Luego se crea el contexto de SharePoint utilizando su Modelo de Objetos de Cliente y se ensambla la cadena segura a utilizar para enviar las credenciales al sistema. Una vez obtenido el acceso a SharePoint, se utiliza el contexto para crear un objeto con la Biblioteca y se muestra en la consola el número de documentos que contiene. Aunque el ejemplo es muy básico en cuanto a funcionamiento de SharePoint, muestra claramente cómo se puede hacer contacto con el sistema y realizar alguna función en él.

15.– Es posible utilizar todas las funciones de debugging de Visual Studio: utilice "F5" o "Debug ▷ Start Debugging" para que el programa sea compilado e iniciado localmente.

16.– Desde el Explorador de Soluciones de Visual Studio, seleccione el proyecto y desde su menú contextual seleccione "Publish as Azure WebJob". La primera ventana que se abre automáticamente permite configurar el momento en el que el Job iniciara su funcionamiento:

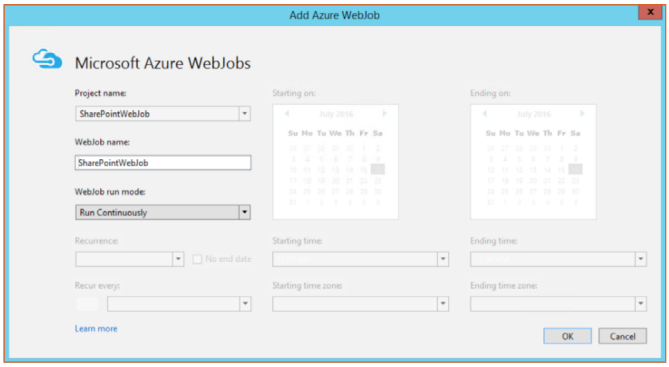


Imagen 2.- Ventana inicial para publicar el WebJob.

Hay múltiples formas para configurar esta función: continuamente, manualmente, cada determinado tiempo recurrentemente o en una fecha determinada. Por el momento seleccione "Run on demand" en la sección "WebJob run mode" y utilice el botón de "OK".

La siguiente ventana permite seleccionar el servicio en Azure. Haga clic sobre "Microsoft Azure App Service":

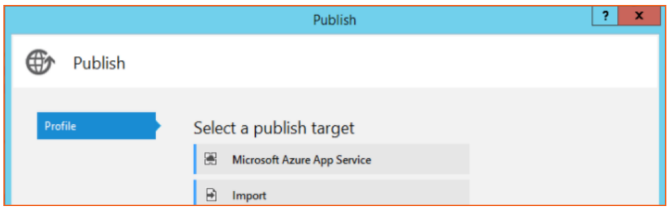


Imagen 3.- Selección del servicio de Azure.

Y en la siguiente ventana seleccione la cuenta a utilizar (probablemente tiene que entrar de nuevo sus credenciales), lo mismo que la suscripción, Grupo de Recursos, etc., y seleccione la Web App que va a contener el WebJob:

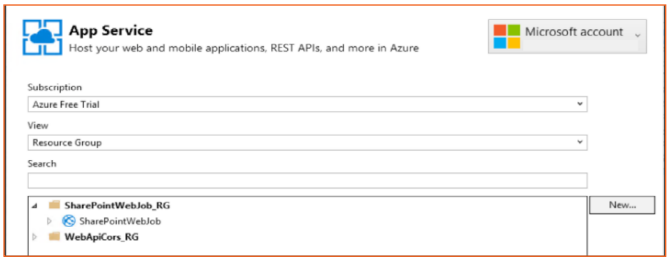


Imagen 4.- Selección de la WebApp dónde publicar.

Las dos ventanas siguientes permiten ver los diferentes parámetros de configuración. Acepte todos los valores por defecto y publique el Job. La ventana de "Output" de Visual Studio permite seguir el procedimiento de publicación, y si ha ocurrido algún error, cual ha sido.

17.– Una vez publicado el WebJob sin que se hayan detectado problemas, es posible ejecutarlo desde el portal de Azure (en el punto anterior se definió que el Job va a ejecutar "On demand"). Desde el portal de Azure seleccione la Web App que contiene el Job, abra la ventana de "All settings" y en la sección de "Web Jobs" haga clic sobre "WebJobs". En la ventana que abre se puede ver una lista con todos los WebJobs contenidos en la Web App. Seleccione el WebJob y utilice el botón de "Run". Después de unos segundos se puede ver el estado del Job:

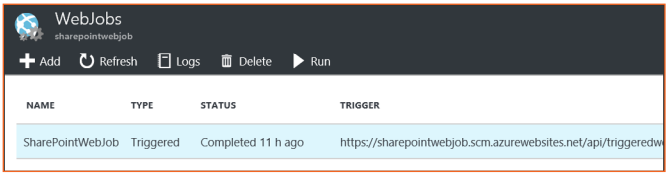


Imagen 5.- Ejecución del WebJob.

Utilizando el botón de "Logs", el navegador abre una nueva pestaña mostrando información sobre las veces que el Job a ejecutado:

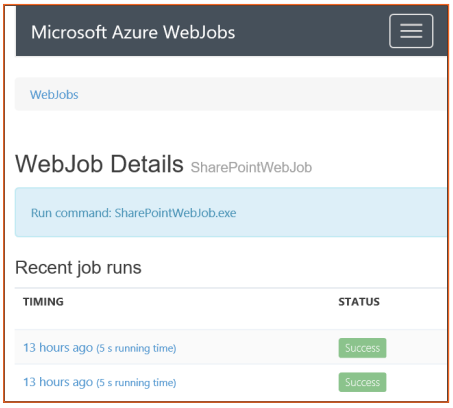


Imagen 6.- Historial de ejecuciones del WebJob.

Y si se hace clic sobre una de las instancias, se puede ver la información completa de ejecución:

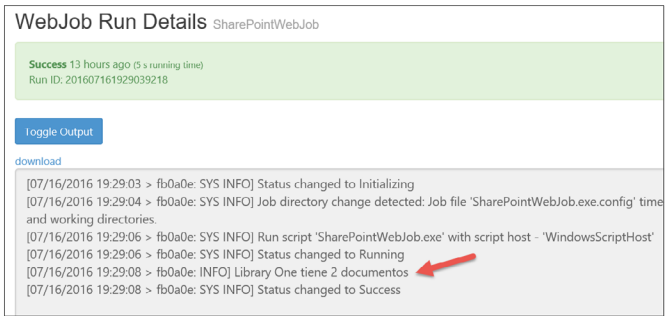


Imagen 7.- Detalle de la ejecución del WebJob.

Note que la salida por consola que se programó en el código (línea 41)

```
Console.WriteLine(myLista.Title.ToString() + " tiene " + myLista.
ItemCount + " documentos");
```

aparece correctamente en los logs. Cualquier información sobre errores atrapados o no atrapados también aparecerá en ellos.

18.– Otra forma para ejecutar el Job es seleccionando la App Web en el portal de Azure, seleccionar "Tools" (en lugar de "Settings") y utilizar la "Consola" de ejecución. El programa se encuentra en el directorio "D:\home\site\wwwroot\app_data\jobs\triggered\[NombrePrograma]" y se puede ejecutar directamente desde la consola:

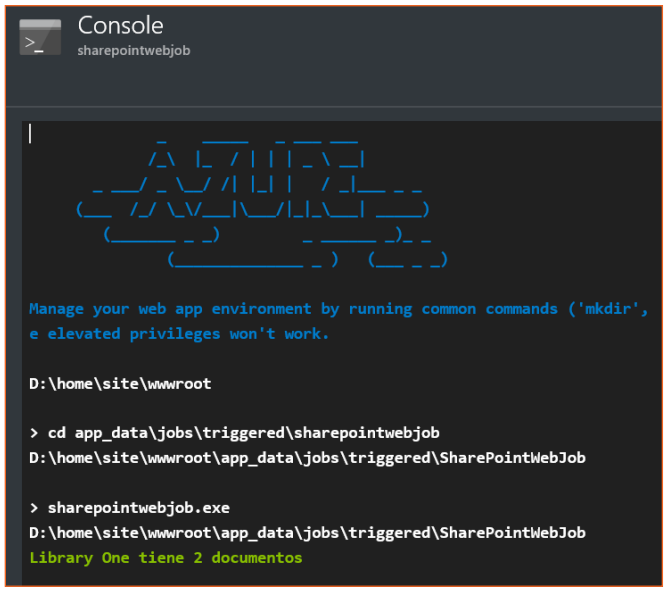


Imagen 8.- Ejecución del WebJob desde la Consola de Azure.

19.– La forma de temporizar el Job se definió en el momento de publicarlo en Azure (punto 16). Cuando se define el momento de ejecución, Visual Studio crea un archivo llamado "webjob-publish-settings.json" en el directorio "Properties" que contiene la configuración:

```
{
 "$schema": "http://schemastore.org/schemas/json/web-
job-publish-settings.json",
 "webJobName": "SharePointWebJob",
 "startTime": null,
 "endTime": null,
 "jobRecurrenceFrequency": null,
 "interval": null,
 "runMode": "OnDemand"
}
```

Si se desea cambiar la configuración, elimine este archivo y publique el Job de nuevo, lo que obliga a Visual Studio a mostrar de nuevo la ventana de definición del tiempo de ejecución y crear un nuevo archivo. No es posible cambiar la forma o el tiempo de ejecución desde el portal de Azure.

los servicios de Azure para complementar el funcionamiento de SharePoint, Azure WebJobs permite mover operaciones que requieren muchos recursos

Conclusiones

Dentro de las múltiples maneras de utilizar los servicios de Azure para complementar el funcionamiento de SharePoint, Azure WebJobs permite mover operaciones que requieren muchos recursos de CPU y/o memoria fuera de la granja de SharePoint hacia procesos remotos en Máquinas Virtuales, y ejecutar programas según un esquema de tiempo predefinido.

WebJobs se pueden utilizar en conjunto con instalaciones de SharePoint OnPremises como una alternativa a los Timer Jobs nativos, o con SharePoint Online para mimetizar el funcionamiento de los Timer Jobs personalizados que no son posibles de crear en SharePoint OnLine.

GUSTAVO VELEZ
MVP Office Servers and Services
gustavo@gavd.net
http://www.gavd.net

04

Integra cualquier sistema en tu PowerApp con Microsoft Flow y Custom API: Agenda de usuarios sobre MongoDB

Antes de adentrarnos en el mundo de Microsoft Flow y la integración de APIs personalizadas sobre PowerApp, debemos entender que solo en casos muy concretos vamos a necesitar utilizar estas herramientas. Es decir, tenemos que delimitar muy bien cuando utilizar conectores estándar que ya nos aporta PowerApp, y cuando tender a desarrollar APIs personalizadas que requieren de un tiempo de implementación.

A lo largo de este artículo veremos cómo integrar una WebApi en nuestras PowerApps como una conexión más dentro de nuestro catálogo de conexiones, y además veremos cómo integrar flujos de trabajos de Microsoft Flow con nuestras aplicaciones.

Dado que siempre es más intuitivo trabajar sobre un ejemplo, en los siguientes puntos vamos a desarrollar una app sencilla en PowerApp que consiste en una Agenda de usuarios que contenga la información de contacto, y la actividad en Twitter de cada usuario de la agenda. La particularidad de la aplicación es que todos los datos van a estar alojados en una base de datos de MongoDB.

Agenda de Contactos: PowerApps y MongoDB

Vamos a implementar una aplicación en PowerApp desde una plantilla en blanco (revisar artículos anteriores si surgen dudas), dejando un poco el diseño a elección propia. La idea de implementar esta APP es poder tener una agenda de contactos en el móvil, diseñar un listado de usuarios, un alta de contactos, una actualización de contactos y una vista de detalle.

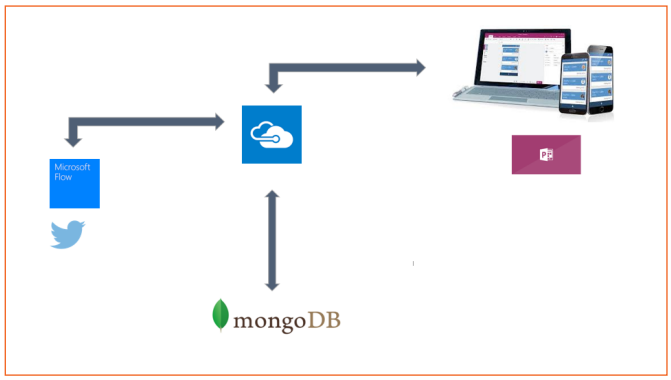


Imagen 1.- Arquitectura de la solución.

Además de almacenar los datos de contactos de un usuario, vamos a almacenar la actividad de Twitter relacionada con nuestra agenda utilizando el hastag #AgendaPowerApps. La actividad de Twitter la vamos a capturar por medio de un Microsof Flow, y la integración entre la Agenda y la base de datos la vamos implementar con un webApi. El api web lo podemos desplegar como un webSite o como un api App en Azure.

WebApi gestión de contactos en la Agenda: AgendaController

Vamos a implementar una webApi sencilla para desplegarla en nuestro entorno de Azure. Para ello desde el Visual Studio creamos una nueva solución y añadimos un proyecto del tipo Web Application / Web Api.

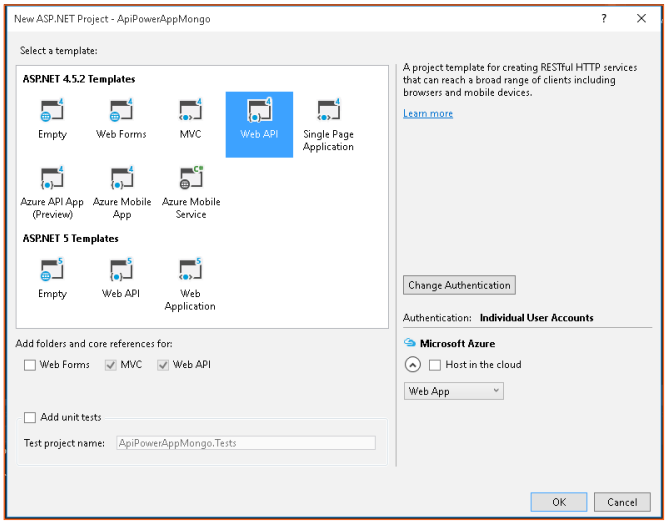


Imagen 2.- Creación web api desde Visual Studio.

Lo vamos a hospedar en Azure por lo que podemos aprovechar a dejar configurado el despliegue antes de continuar. En cuanto a la autenticación no es relevante para este ejemplo, pero aclarar que PowerApps soporta autenticación con OAuth contra Azure AD.

Una vez creada la solución y la WebApi vamos a crear los métodos necesarios para gestionar la Agenda.

Lo primero es crear un nuevo controlador en la Api que llamaremos AgendaController, y yo recomiendo borrar todos los controladores que no vayamos a utilizar y que la plantilla base de Visual Studio nos ha generado.

AgendaController podría quedar algo de la siguiente forma:

```
[HttpGet]
    [Route("GetAllContact")]
    public IEnumerable<BaseContact> GetAllUser(){}

    [HttpGet]
    [Route("GetUserById")]
    public Contact GetUser(string id){}

    [HttpPost]
    [Route("AddUser")]
    public bool AddUser([FromBody]Contact contact){}

    [HttpPost]
    [Route("DeleteUser")]
    public bool DeleteUser(string id){}

    [HttpPost]
    [Route("AddTweet")]
    public bool AddTweet(Tweet tweet){}
```

Vamos a tener dos métodos GET para obtener el listado de usuarios y el detalle de un usuario por Id, y tres métodos POST para implementar el Añadir, Borrado y Añadir nuevo Tweet. Más adelante veremos que contienen los modelos BaseContact, Contact y Tweet y una posible implementación de la solución.

debemos entender que solo en casos muy concretos vamos a necesitar utilizar estas herramientas

Supongamos que ya tenemos implementada la WebApi tal y como necesitamos, y queremos proceder a registrar la API en nuestra PowerApp. Decir que el primer punto que debemos cubrir es tener instalado Swagger en nuestra Api. Este framework para API's, nos va a permitir extraer la definición de nuestra WebApi e integrarla en PowerApps. Para ello desde el visual studio necesitamos instalar el paquete Nuget Swashbuckle, tal cual se aprecia en la siguiente figura.

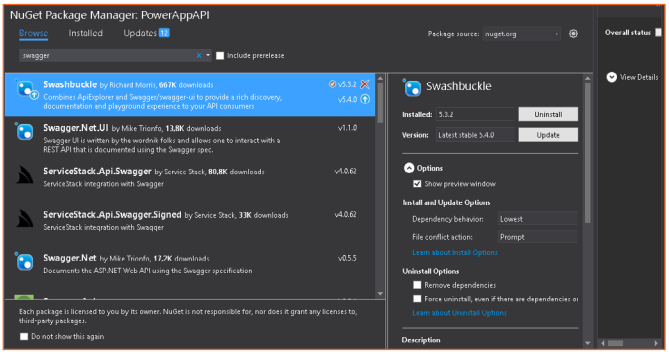


Imagen 3.- Añadiendo Swagger al proyecto.

Una vez instalado el Swagger, y tras haber implementado correctamente el controlador, podemos publicar el api en el webSite de Azure. Si accedemos a <url publicación api>/swagger podremos ver la estructura de nuestra API, realizar testeo de los diferentes métodos expuestos y lo más importante para el registro obtener el fichero de definición.

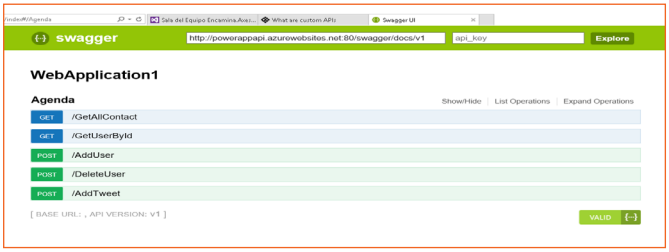


Imagen 4.- Utilizando Swagger.

Para ello accedemos a <url api>/swagger/docs/v1, y descargamos el fichero .json que contiene toda la definición de la API que acabamos de desplegar. Este fichero no hace falta modificarlo ya que Swagger ha extraído todas las configuraciones y métodos que hemos ido definiendo desde el Visual Studio.

Una vez descargado el fichero vamos a acceder a https://powerapps.microsoft.com/es-es/ con nuestra sesión de usuario y seleccionamos "conexiones" y nueva conexión.

En la siguiente pantalla veremos que tenemos las conexiones estándar, y un apartado para conexiones "Custom". Seleccionamos New Custom Api para registrar el API que hemos implementado.

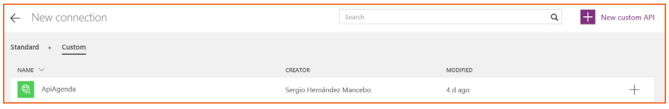


Imagen 5.- Registro de una API en PowerApps.

En este punto se nos solicitará el archivo de definición que nos ha generado Swagger, y algún dato de personalización como el nombre o una imagen personalizada. Una vez registrada la API, (veremos que es un paso casi instantáneo), debemos registrarla como nueva conexión, para que nuestras PowerApps puedan consumirlas.

Si vemos la imagen anterior, es tan fácil como seleccionar "+", y seleccionar "Add Connection".

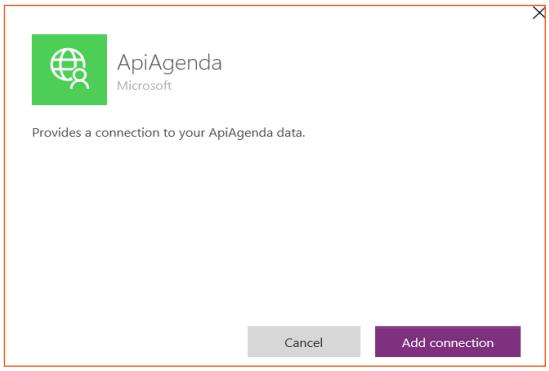


Imagen 6.- Añadir conexión a las App en PowerApps.

Desde este momento ya podemos añadir como nuevo "DataSource" nuestra API en la PowerApp, como veremos un poco más adelante.

Capa de Acceso a Datos, Modelos y MongoDB

Una vez elegido MongoDB como motor de base de datos, debemos estructurar bien la información para huir de los métodos de diseño de base de datos que utilizábamos en

los modelos relacionales. Para conseguir esto, el primer objetivo que debemos cumplir es evitar los cruces entre nuestras entidades de datos, ya que en MongoDB este hecho nos podría penalizar, ya que nuestros datos están almacenados en documentos y no en tablas relacionadas.

Por tanto, para este ejemplo se ha desarrollado un modelo de documento que almacena el listado de todos los usuarios y la mínima información necesaria, y otro modelo de documento que almacena la información extendida del contacto y las entradas de Twitter que realice.

Podemos decir entonces que el documento Index (listado de contactos) va a ser único y se va a cruzar con nuestro modelo BaseContact, y el documento "Contacto" va estar replicado una vez por cada contacto y se va a mapear con el modelo de clase Contact.

Toda la información va ir serializada en formato JSON, para simplificar los mapper, y por seguir un estándar.

Modelos

BaseContact

```
public class BaseContact
  {
    public string Id { get; set; }
    public string Name { get; set; }
    public string UserName { get; set; }
    public string Image { get; set; }
  }
```

Contact

```
Public class Contact : BaseContact
  {
    public string Adress { get; set; }
    public string Phone { get; set; }
    public string Date { get; set; }
    public List<Tweet> TweetMessage { get; set; }
  }
```

El modelo contact contiene una lista de Tweet, que no es más que la información recogida desde Twitter como puede ser el nombre, usuario, fecha…..; un poco lo que necesitemos almacenar.

Los documentos que vaya almacenando el api en el MongoDB, deberán llevar un metadato adjunto al documento para poder hacer consultas contra la base de datos. En este caso vamos a utilizar el campo "Id" de BaseContact, que va a llevar la cuenta de Twitter asociada al contacto, y que sabemos a ciencia cierta que es única en el sistema.

deberán llevar un metadato adjunto al documento para poder hacer consultas contra la base de datos

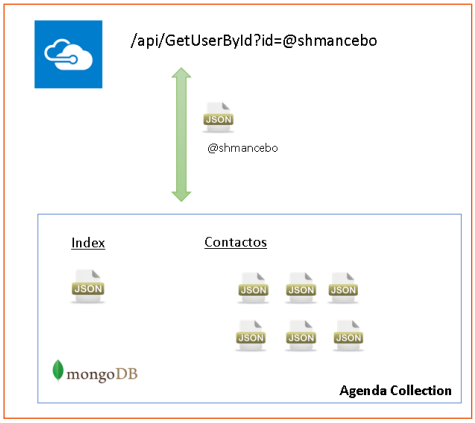


Imagen 7.- Modelo base de datos en Mongo DB.

Estructura de la solución

Para consumir la base de datos en MongoDB vamos a implementar una clase repositorio que nos permita insertar, actualizar y obtener los ficheros JSON que necesitamos, así como mapearlos en los modelos de clases que hemos definido con anterioridad.

Como podemos ver en la imagen de la solución, se ha optado por un patrón repositorio para el acceso a datos, y un modelo de N-capas para distribuir la aplicación.

Lo importante del artículo no es la implementación de la solución, pero es un ejemplo de cómo podía quedar en caso de querer abordarla.

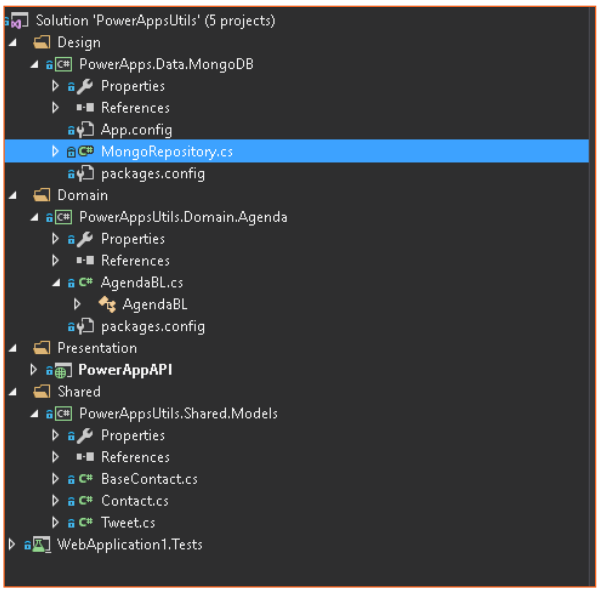


Imagen 8.- Estructura de la solución.

Para consumir la base de datos en MongoDB vamos a implementar una clase repositorio que nos permita insertar, actualizar y obtener los ficheros JSON que necesitamos, así como mapearlos en los modelos de clases que hemos definido con anterioridad.

Como podemos ver en la imagen de la solución, se ha optado por un patrón repositorio para el acceso a datos, y un modelo de N-capas para distribuir la aplicación.

Lo importante del artículo no es la implementación de la

solución, pero es un ejemplo de cómo podía quedar en caso de querer abordarla.

No voy a profundizar en la capa de negocio, en mi caso he implementado una clase AngendaBL, que se encarga de orquestar las llamadas a la capa de acceso a datos y recibir las llamadas desde la API. Esta capa se puede implementar a medida de la solución que queramos dar, en mi caso podemos ver un ejemplo de cómo queda el método "AddIndexContact".

```
public bool AddIndexContact(Contact contact)
    {
        var users = GetContacts()?.ToList();
        MongoRepository<BaseContact> mongoRepository = new
MongoRepository<BaseContact>();
        users = users ?? new List<BaseContact>();
        var existUsers = users.Where(c => c.Id == contact.Id).Fir-
stOrDefault();
        if (existUsers != null)
        {
            users.Remove(existUsers);
            DeleteContact(existUsers.Id);
            DeleteContact("index");
        }
        users.Add(contact.GetMinContact());
        var document = JsonConvert.SerializeObject(users);
        var id = mongoRepository.UploadFileFromBytes(System.
Text.Encoding.UTF8.GetBytes(document), "index");
        return !string.IsNullOrEmpty(id);
    }
```

Este método se encarga de "verificar" si el contacto ya existe en la colección de MongoDB, y si es así descarga el documento, borra de la lista el usuario para actualizarlo de nuevo, borra el documento index y lo sube de nuevo actualizado. Esto es porque MongoDB no aporta un update directo, por lo que debemos descargar el documento, incrementarlo y borrar la copia anterior antes de subirla, para no tener varios documentos con el mismo metadato asociado. Con este método actualizamos el documento Index.json Sería un proceso similar actualizar el documento contact.json

MongoRepository: controla tu MongoDB

Igual que la clase negocio no resulta interesante para este artículo, si merece la pena detenernos un poco más en la clase repositorio de MongoDB, y ver que paquetes necesitamos para manejar una base de datos no relacional de este tipo.

necesitamos añadir un nuevo paso para registrar el Tweet en nuestra base de datos

Para implementar MongoRepository hemos hecho uso principalmente de los paquetes NuGet MongoDB.Driver.GridFS y MongoDB.Bson.

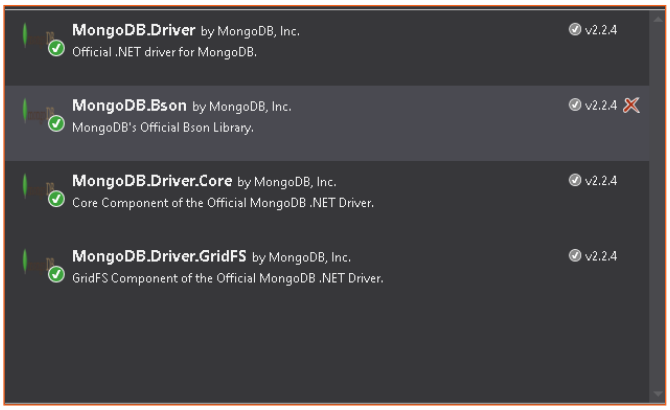


Imagen 9.- Paquetes necesarios para implementar MongoRepository

Con el paquete MongoDB.Bson controlamos todo el apartado de conexiones y consultas, y por el contrario con el paquete MongoDB.Driver.GridFS controlamos el tráfico de ficheros.

Lo primero que necesitamos para hacer una consulta o un uplodad de un documento en MongoDB desde código es instanciar una conexión contra el servidor de base de datos.

Una buena forma es implementar en el constructor de la clase repositorio la conexión a "MongoClient", y así tenerlo disponible desde el primer momento.

```
public MongoRepository()
    {
        ServerMongoDb = ConfigurationManager.AppSet-
tings["urlMongoDb"];
        this.Database = ConfigurationManager.AppSettings["da-
tabaseMongoDB"];
        this.Collection = ConfigurationManager.AppSettings["col-
lectionMongoDB"];
        this.CollectionInternalName = this.Collection + ".files";
        Client = new MongoClient(this.ServerMongoDb);
    }
```

No debemos preocuparnos de cerrar la conexión, porque en este caso MongoClient no es disposable.

Como apreciamos en el código necesitaremos la url al servidor, una base de datos en Mongo, y el nombre de la colección donde haremos las operaciones (aunque Mongo aporta una colección files por defecto si no se lo indicamos).

Para realizar una consulta contra MongoDB podemos fijarnos en el método GetDocument(string id), que vamos a implementar para obtener documentos desde el MongoDB.

```
public dynamic GetDocument(string codDocumento)
    {
        try
        {
            var database = Client.GetDatabase(Database);
            var bucket = new GridFSBucket(database);
            var listaFiltros = new List<FilterDefinition<GridFSFileIn-
fo>>();
            var filtro = Builders<GridFSFileInfo>.Filter.Eq(fi => fi.Meta-
data["codDocumento"], codDocumento);
            listaFiltros.Add(Builders<GridFSFileInfo>.Filter.And(fil-
tro));
```

```
            var filters = Builders<GridFSFileInfo>.Filter.Or(listaFiltros);
            var files = bucket.Find(filters).ToList();
            if (files.Any())
            {
              var datos = bucket.DownloadAsBytesByNameAsync(-
files.FirstOrDefault()?.Filename).Result;
              var valueJson = System.Text.Encoding.UTF8.GetString(-
datos);
              return JsonConvert.DeserializeObject<T>(valueJson);
            }
            else return null;
          }
          catch
          {
            throw;
          }
      }
```

En un primer paso recuperamos la base de datos con el método GetDataBase, y con esta vamos a crear una instancia de GridFSBucket, que nos va a permitir aplicar los filtros necesarios para realizar una consulta sobre la base de datos.

En este caso solo tenemos un campo metadato asociado al documento que es el "codDocumento", y que como dijimos antes será la cuenta de Twitter del usuario. Si obtenemos algún tipo de file relacionado al codDocumento consultado, basta con hacer una descarga del documento con el método "DowloadAsByteNameAync", que nos devuelve el fichero en bytes desde la bbdd.

Una vez tenemos descargado el fichero ya podemos trabajar con el tal y como necesitemos, como podemos ver en el código ejemplo nosotros necesitamos deserializarlo en un tipo anónimo que bien el caso puede ser BaseContact o Contact en función de la petición que necesitemos devolver.

Por ultimo nos quedaría por implementar un Upload de un documento en MongoDB. Para ello necesitamos el siguiente código:

```
 public string UploadFileFromBytes(byte[] content, string cod-
Documento)
      {
        try
        {
          ObjectId result;
          var database = Client.GetDatabase(Database);
          var bucket = new GridFSBucket(database);
          var options = new GridFSUploadOptions
          {
            Metadata = new BsonDocument
          {
            {"codDocumento", codDocumento},
          }
          };
          result = bucket.UploadFromBytes($"loadFile(codDocu-
mento)", content, options);
          return result.ToString();
        }
        catch
        {
          throw;
        }
      }
```

Como vemos es muy sencillo, basta con asociar el metadato correcto al fichero que recibimos en el parámetro content en byte[] y realizar un UploadFromBytes.

Anotar una entrada en Twitter: Microsoft Flow

Para registrar en el modelo Contact los tweets de un usuario vamos a diseñar un pequeño flujo de trabajo. Para ello desde el mismo portal de PowerApps que hemos visitado en el punto de registro del api seleccionamos "Flows" en el menú global izquierdo y después "Create a Flow".

Esto nos lleva al diseñador de flujos de Microsoft Flow, y que como vemos es muy intuitivo. Podemos utilizar una plantilla base, o en nuestro caso una en blanco.

Se nos va a solicitar el evento que fuerza el arranque del flujo, en nuestro caso vamos a utilizar el conector de Twitter, y el evento "When a new tweet is posted", configurándolo como vemos en la imagen.

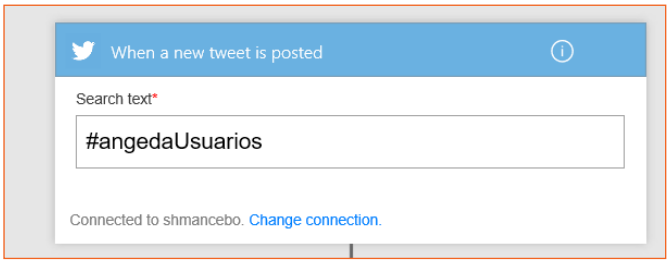


Imagen 10.- Iniciando un Microsoft Flow.

Esta primera caja, va a esperar a que alguien en Twitter escriba un nuevo mensaje con "#agendaUsuarios", en caso de que esto suceda se iniciará una instancia del flujo.

Una vez iniciado necesitamos añadir un nuevo paso para registrar el Tweet en nuestra base de datos. Para conseguirlo vamos a aprovechar de que tenemos una API, y seleccionando en nuevo paso, buscamos el método "AddTweet" que tenemos expuesto como método POST en nuestra API (muy importante que sea un método post o no va a parecer en este tipo de caja).

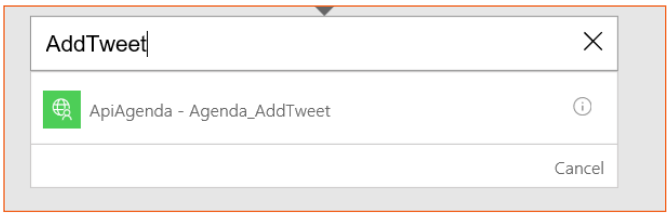


Imagen 11. Conectando un Microsoft Flow a una custom API.

Como vemos podemos conectar un Flow directamente a un método de la API, sin necesidad de realizar ninguna petición compleja o configuración intermedia.

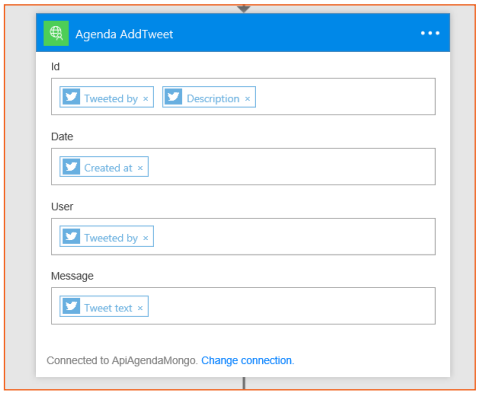


Imagen 12. Petición POST contra una custom API.

Añadimos la caja y la configuramos tal y como viene en la siguiente imagen. Como podemos deducir, se exponen los campos del modelo “Tweet” que si recordamos le pasábamos al método AddTweet de la apiAgenda.

Mapeamos estos campos del modelo con valores de Twitter de tal forma que la API reciba todos los parámetros necesarios para rellenar el modelo en base de datos. Por ultimo le damos un nombre al Flow, y lo guardamos para que pueda empezar a usarse.

Si escribimos un Tweet, podemos hacer un seguimiento de la ejecución de este trabajo desde el propio portal de flow, seleccionando en el apartado “info” que tenemos junto al flujo seleccionado.

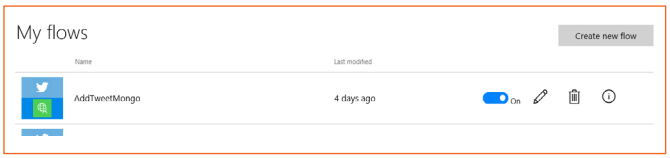


Figura 13.- Seguimiento de un Flow.

App Agenda: Consumir una Api desde PowerApps

Una vez tenemos los procesos y la Api implementados podemos centrarnos en implementar la propia aplicación. Para ello podemos crearla desde la aplicación PowerApps de Windows 10, o desde hace poco desde la versión preview del editor en versión web.

Como he adelantado al principio del artículo no me voy a detener mucho en la “implementación” de la app, ya que en artículos anteriores hemos hablado mucho de este punto, decir que solo se han utilizado controles básicos como Galerías personalizadas, Botones, eventos de Navegación y alguna colección de datos que repasaremos en este punto.

Lo primero que debemos hacer es añadir como DataSource la api que hemos implementado durante el ejemplo. Para ello desde la barra superior en el apartado content, seleccionamos DataSource y buscamos la apiAgenda seleccionando “Add data Source”.

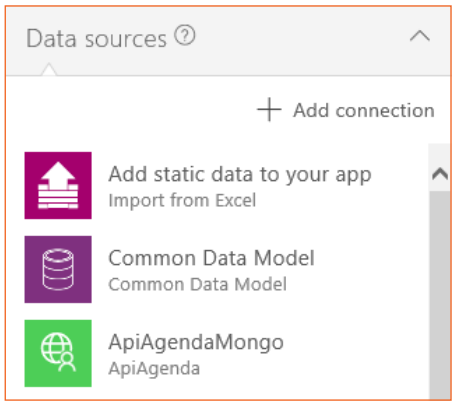


Imagen 14.- Añadir un API como DataSource.

Como adelantamos al inicio la primera pantalla de la APP es el listado de usuarios.

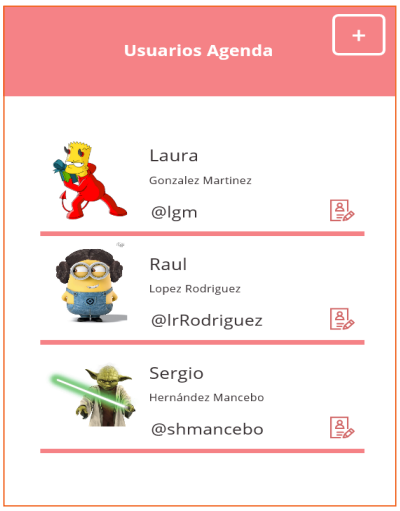


Imagen 15.- Listado de contactos.

Esta pantalla se ha implementado con una galería personalizada que mapea un listado del modelo “BasicContact”, de la forma que en el evento “OnVisible” de esta pantalla, se ha aprovisionado una colección de datos llamada Usuario.

ClearCollect(usuarios,ApiMongo.AgendaGetAllUser())

Esto nos permite almacenar la lista de usuarios en una colección propia de PowerApp, y en caso necesario poder trabajar con Filtros o Trasformaciones de datos. Además de este listado de usuarios se han implementado dos pantallas más como son AddContact y ViewContact.

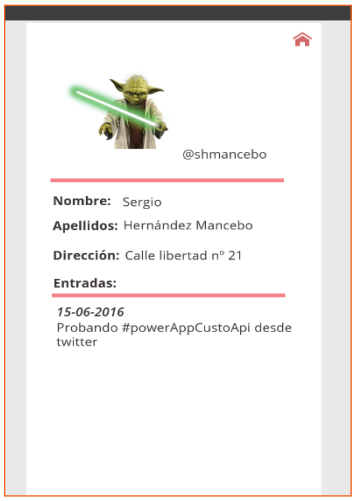


Imagen 16. Pantalla detalle contacto y entradas.

En el caso de la vista de detalle, en el evento de carga se crean dos colecciones de la forma:

- ClearCollect(usuarios,ApiMongo.AgendaGetUser(currentId));
- ClearCollect(entradas,ApiMongo.AgendaGetUser(currentId).TweetMessage)

Si vemos en la imagen los datos personales y la imagen se corresponden con la colección usuarios, y por otro lado la lista de entradas (Tweets) se corresponde con la colección entrada.

Solo falta por deducir que el parámetro “currentId”, se envía desde la lista de usuarios de la forma Navigate

(VistaDetalle,ScreenTransition.Fade,{idUsuario:Id}).

La función Navigate, además de permitir hacer enlaces entre las distintas vistas de la aplicación, permiten pasar en la petición parámetros, como puede ser el idUsuario. Eso sí tenemos que recordar que en la carga de la página destino tenemos que capturar en el context de la página el parámetro enviado, tal y como hacemos en la página de detalle con la instrucción UpdateContext({currentId: idUsuario}).

Si nos centramos en la página de "AddContact", poco interesante podemos deducir a nivel de implementación en PowerApps, es un formulario simple con un control de carga de imágenes, y múltiples TexBox, solo a destacar el evento Guardar, que muestra de forma muy visual lo rápido que es hacer peticiones post desde PowerApp contra una custom Api.

```
ApiMongo.AgendaAddUser({Name:Nombre.Text,Image:ImagenPerfil.Media,Id:Twitter.Text,UserName:Apellidos.Text,Adress:Dirección.Text})
```

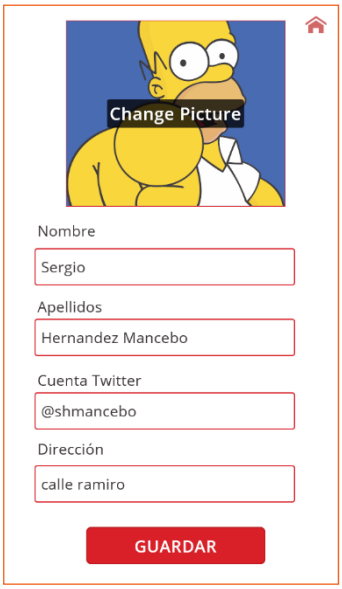


Figura 17.- Pantalla añadir contacto.

Como podemos ver en la imagen, basta con mandarle al método AgendaAddUser el modelo "Contact" que espera recibir entre {}. Si analizamos la petición el propio editor de fórmulas te indica que parámetros requiere la petición y basta con cruzarlos con los textBox de la pantalla.

Conclusiones

Bueno con este artículo hemos ido un paso más allá en nuestra serie de PowerApps. Posiblemente lo que hemos podido ver inquiete a los usuarios de negocio dado que hemos tratado conocimientos complejos, y motive a los desarrolladores a lanzarse a usar esta tecnología ya que este modelo de trabajo con PowerApps es mucho más flexible y libre que trabajar solo con fórmulas y conectores estándar.

Pero como casi siempre en la vida no todo es blanco o negro, y en este caso lo único complejo es "decidir" cuando optar más por un modelo basado en conectores estándar, formulas y controles básicos, o por otro lado optar más por implementar una API personalizada e impactar en horas de desarrollo un proyecto.

En nuestro caso es evidente, a día de hoy no podemos trabajar con MongoDB si no es con el segundo modelo, pero también hay otros factores como puede ser el rendimiento de una PowerApps a grandes volúmenes de información, o la complejidad de las consultas, que nos pueden hacer movernos a implementar una Api y aligerar así nuestros DataSource.

Lo que sí que es cierto, es que según avanzas con este producto ves que cada vez incluye más perfiles potenciales que pueden centrarse en su uso y desarrollo, y lo más importante se empieza a ver integrado en capa web, en listas de Sharepoint Online….; tenemos PowerApps para rato, así que empecemos a darles cariño.

SERGIO HERNANDEZ MANCEBO
Team Leader en ENCAMINA
shernandez@encamina.com

Mentoring

Comparti*MOSS*

Un servicio experto alrededor de su SharePoint

CompartiMOSS le puede ayudar a través de su programa de Mentoring!

Contacte con nosotros y le enviaremos los planes de mentoring que tenemos disponibles para SharePoint.

33

Entrevista Darwin Castro Marín

Mi nombre es Darwin Castro Marín, soy Ingeniero Microsoft y vivo en Tenerife – España.

Soy Microsoft Certified Professional desde el año 2001.

Me especializo en Sistemas Operativos e infraestructura de redes basadas en dominios, además cuento con un fuerte background en soluciones de colaboración y mensajería como Microsoft Exchange.

Soy el tipo de profesionales que le encanta la interacción e integración de distintas tecnologías, desde que conocí Microsoft Office365 me ha fascinado el mundo del cloud computing y sobre todo con la visión que tiene Microsoft en este gran ámbito de la tecnología.

¿Por qué y cómo empezaste en el mundo de la tecnología?

Desde antes de comenzar oficialmente en la informática me he sentido motivado por los sistemas operativos en general, empecé a trabajar con Microsoft MS-DOS y Windows 3.0.
Considero esa época excepcional para la evolución de la tecnología, al mismo y a nivel de servidores tuve la gran oportunidad de trabajar con OS/2 Warp 3.0, en ese momento enlazar sistemas basados en MS-DOS y OS/2 Warp era la demanda clásica del mercado.
Profundicé mis conocimientos en Windows con la salida de Microsoft Windows 95, me especialicé en ejecutarlo en equipos de bajos recursos de hardware lo que me dio la oportunidad de conocer el sistema operativo a fondo.
Una vez liberado Microsoft Windows NT 4.0 Server abandoné toda la parte de cliente y me dediqué 100% a Servidores.
Con la llegada de Microsoft Windows 2000 Server y Active Directory empecé a trabajar en ambientes más complejos gracias al apoyo de Microsoft al incorporarme en el programa de estudiantes excepcionales de tecnología “Fuerza de Apoyo Microsoft” (FAM Program). Desde allí me he dedicado a la infraestructura de sistemas a nivel empresarial.

¿Cuáles son tus principales actividades tecnológicas hoy en día?

Actualmente soy socio y director de una de las compañías más grandes en servicios tecnológicos de Tenerife. Dentro de mis actividades actuales están: la dirección y visión de la compañía en las diferentes áreas del mercado, llevando a nuestros clientes a servicios basados en cloud computing, básicamente nuestra oferta se basa en Microsoft Azure & Microsoft Office365.
Además de ello dirijo el departamento de diseño y desarrollo de nuestra compañía, esto me ha dado la oportunidad de conocer plataformas basadas en Unix como Linux, FreeBSD y Mac OSX.

¿Cuáles son tus principales actividades NO tecnológicas hoy en día?

Intento pasar el mayor tiempo disponible con mi familia.

¿Cuáles son tus hobbies?

Cuando tengo la oportunidad, viajar; pero la otra pasión más grande es la astronomía, además me considero afortunado de vivir en unos de los cielos más limpios del mundo.

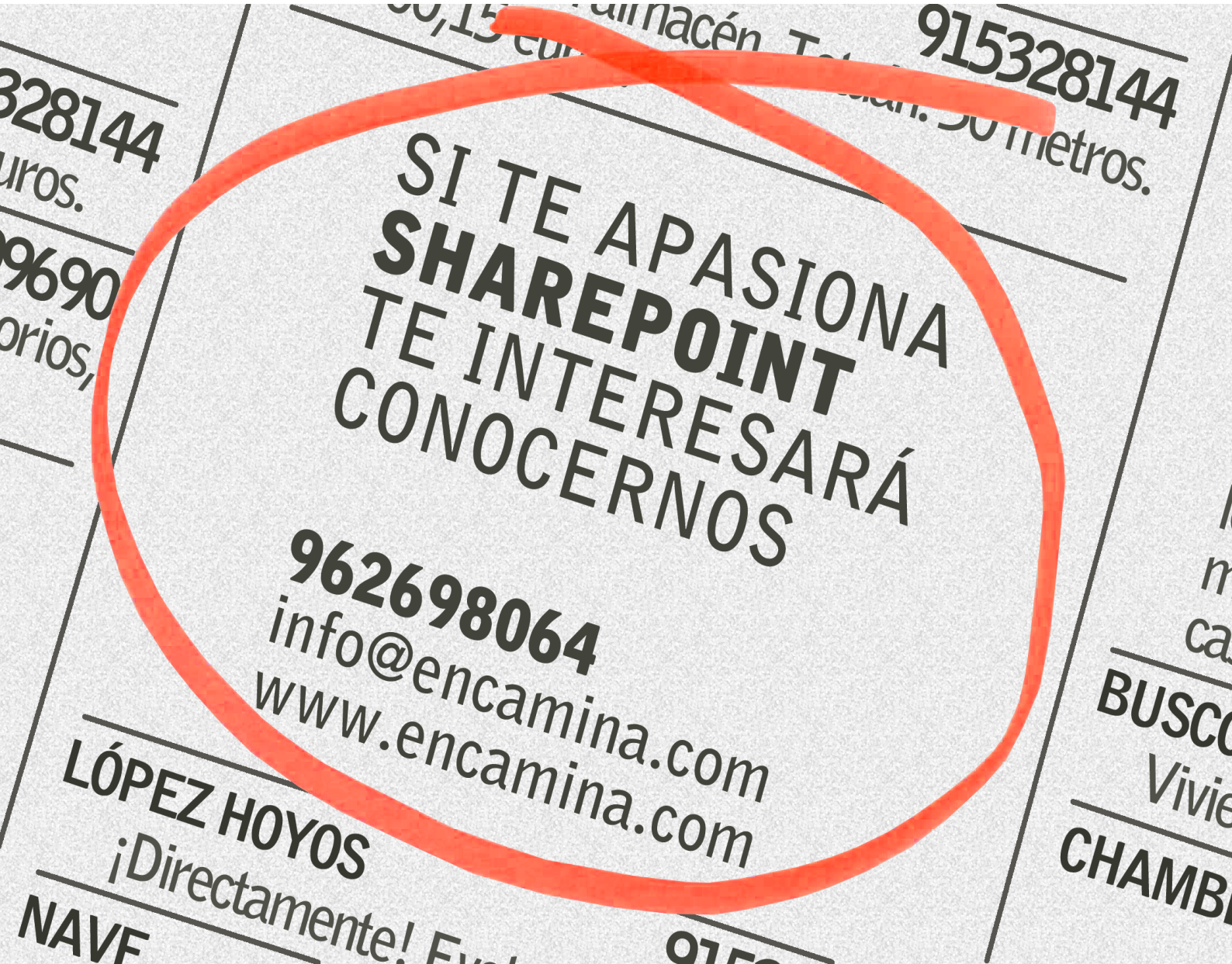
¿Cuál es tu visión de futuro en la tecnología de acá a los próximos

años?

Para esta pregunta cualquiera podría responder casi de manera automática: el cloud computing, obviamente lo es, la mejor apuesta que han hecho los fabricantes es el software como servicio, además de limpiar de alguna manera la piratería nos ha ayudado a disfrutar de software que nunca alcanzaríamos con el modelo de comercialización anterior.

Pero mi visión de la tecnología en los próximos años es muy distinta a lo que tenemos en este momento ya que muy a mi pesar los sistemas operativos están perdiendo importancia dentro del ámbito del consumo de los usuarios e inclusive se empieza a notar en las empresas. La adopción de tecnologías basadas en web seguirá favoreciendo la movilidad y la posibilidad de dispositivos de bajo recursos y consumo.
Soluciones como Microsoft Azure, Amazon Web Services, etc., triunfarán como plataformas de procesamientos de datos, datos que llegaran a los usuarios con móviles, tabletas, ordenadores, pero al final son datos, datos que seguirán favoreciendo el Power BI.
Los coches, los aviones y los drones se integrarán más y mejor con nuestros dispositivos, en mi entender particular la informática tradicional de cliente/servidor ha muerto, hoy en día los usuarios no esperan el próximo Windows para ver cómo será el nuevo icono de la papelera de reciclaje, nos da igual lo que se ejecute por dentro, queremos conexiones veloces, respuestas automáticas, acceso a nuestra información.

DARWIN CASTRO MARÍN
Office Servers and Services MVP
darwin.castro@ancadia.com
@solomicrosoft
http://www.solomicrosoft.com



04

Xamarin Forms y Sharepoint On Premise

Los desarrolladores de aplicaciones multiplataforma para móvil estamos de enhorabuena, tenemos una plataforma como Xamarin y el Framework de Xamarin Forms que nos facilita el desarrollo para diferentes plataformas. Imaginemos que un cliente nos pide una aplicación para el móvil en la que se muestren datos procesados de su SharePoint On-Premises y que funcione en Android, en iOS y en Windows Phone. Ahora esto es una tarea bastante fácil y en este artículo vamos a ver un ejemplo de cómo acceder a datos de On-Premises desde una app Xamarin.

Lo primero es crear una nueva App de tipo Xamarin. Forms:

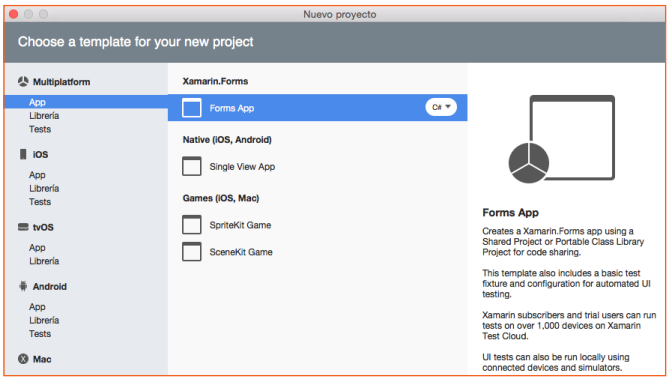


Imagen 1.- Creación de una App de Xamarin.Forms.

Elegimos el tipo PCL para portable class library aunque como luego veremos vamos a incluir un proyecto de tipo Shared Library.

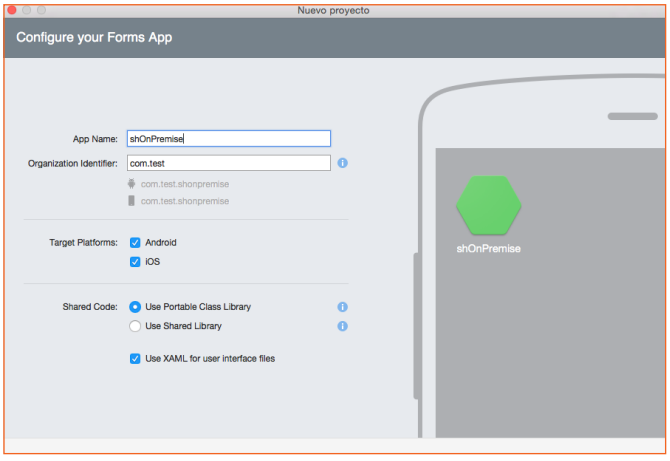


Imagen 2.- Parámetros de configuración de la App.

Una vez creado el proyecto vamos a crear unas clases que nos van a servir para realizar peticiones a los servicios REST de SharePoint haciendo uso de la clase System.Net.Http-Client. El truco es que estas clases hay que crearlas en un nuevo proyecto que vamos a añadir a nuestra solución de tipo Shared project, la explicación para tener que hacer esto es doble:

- por un lado, los proyectos de tipo Portable library no admiten operaciones Async (y queremos realizar las llamadas de forma Async), esto se puede solucionar añadiendo la librería BCL pero no es lo que queremos ahora mismo.
- por otra parte, para acceder a SharePoint On-Premises vamos a utilizar credenciales de Windows, pero el tipo de proyecto Portable library no admite la librería System.Net.Credentials y para solucionarlo deberíamos utilizar extensores de código para cada tipo de proyecto (Android, IOS, etc).

Como digo la forma simple de solucionar esto es crear un proyecto de tipo Shared code y añadir estas dos clases:

```
#region Espacios de nombres

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

#endregion

namespace shApp
{
  /// <summary>
  /// Cliente HTTP para SharePoint
  /// </summary>
  public class SPHttpClient : HttpClient
  {

    public SPHttpClient(Uri webUri, string user-
Name, string password) : base(new SPHttpClientHandler(webU-
ri, userName, password))
    {
      BaseAddress = webUri;
    }

    /// <summary>
    /// Execute request method
    /// </summary>
    /// <param name="requestUri"></param>
    /// <param name="method"></param>
    /// <param name="headers"></param>
    /// <param name="payload"></param>
    /// <returns></returns>
    public async Task< JObject> ExecuteJson<T>(string reques-
tUri, HttpMethod method, IDictionary<string, string> head-
ers, T payload)
    {
```

```
        try
        {
          HttpResponseMessage response;
          switch (method.Method)
          {
            case “POST”: // Operaciones Insert/Update/Delete
              var requestContent = new StringContent(JsonCon-
vert.SerializeObject(payload));
              requestContent.Headers.ContentType = MediaType-
HeaderValue.Parse(“application/json;odata=verbose”);
              DefaultRequestHeaders.Add(“X-RequestDi-
gest”, GetDigest(this));
              if (headers != null)
              {
                foreach (var header in headers)
                {
                  DefaultRequestHeaders.Add(header.Key, header.
Value);
                }
              }
              response = await PostAsync(requestUri, requestCon-
tent);
              break;
            case “GET”:
              response = await GetAsync(requestUri);
              break;
            default:
              throw new NotSupportedException(string.Format(“-
Method {0} is not supported», method.Method));
          }

          response.EnsureSuccessStatusCode();          var con-
tenidoRespuesta = response.Content.ReadAsStringAsync().
Result;
          return String.IsNullOrEmpty(contenidoRespues-
ta) ? new JObject() : JObject.Parse(contenidoRespuesta);
        }
        catch (Exception ex)
        {
          return null;
        }

      }


      public async Task< JObject> ExecuteJson<T>(string reques-
tUri, HttpMethod method, T payload)
      {
        return await ExecuteJson(requestUri, method, null, pay-
load);
      }


      public async Task< JObject> ExecuteJson(string requestUri)
      {
        return await ExecuteJson(requestUri, HttpMethod.
Get, null, default(string));
      }


      public string GetDigest(HttpClient client)
      {
        if (Common.Token != “”)
          return Common.Token;

        string retVal = null;
        // string cmd = “_api/contextinfo”;
        var endpointUrl = string.Format(“{0}/_api/contextinfo”, Ba-
seAddress);
        HttpResponseMessage response = this.PostAsync(end-
pointUrl, new StringContent(string.Empty)).Result;
        if (response.IsSuccessStatusCode)
        {
          string content = response.Content.ReadAsStringAsync().
Result;
          JToken t = JToken.Parse(content);
          retVal = t[“d”][“GetContextWebInformation”][“FormDi-
gestValue”].ToString();
          Common.Token = retVal;
        }
        return retVal;
      }
    }
}
```

y el handler en el cual inyectamos las credenciales:

```
#region Espacios de nombres

using System;
using System.Net;
using System.Net.Http;
using System.Security;
using System.Threading;
using System.Threading.Tasks;
#endregion

namespace shApp
{

  public class SPHttpClientHandler : HttpClientHandler
  {
    public SPHttpClientHandler(Uri url, string usu-
ario, string password)
    {
      this.Credentials = new NetworkCredential(usuario, pass-
word);
    }
  }
}
```

Con esas dos clases ya tenemos el acceso a SharePoint mediante servicios REST y credenciales Windows. Ahora nos queda crear una clase que se corresponda con nuestra lista de SharePoint, en nuestro caso se trata de una lista llamada List_Locations (lista de ubicaciones), esta lista solo tiene una propiedad Title y por tanto nuestra clase queda así…

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Newtonsoft.Json.Linq;
using SQLite;

namespace shApp
{
  public class List_Locations
  {

    public List_Locations()
    {
    }

    public static async Task<List<LocationsItem>> getList()
    {
      List<LocationsItem> lista = new List<LocationsItem>();
      // Variable para gestionar los resultados JSON
      JObject datos;
      try
      {
        using (var cliente = new SPHttpClient(new Uri(Common.
urlSharepoint), Common.MasterUser, Common.MasterPass-
word))
        {
          //cliente.Timeout = new TimeSpan(50);
          var urlPeticion = string.Format(“{0}/_api/web/lists/get-
bytitle(‘{1}’)/items”, Common.urlSharepoint, “List_Locations”);
          datos = await cliente.ExecuteJson(urlPeticion);

          if (datos != null)
          {
            foreach (var item in datos[“d”][“results”])
            {
              lista.Add(new LocationsItem
              {
                ID = System.Convert.ToInt32(item[“ID”].To-
String()),
                Title = item[“Title”].ToString()
              });
              //Console.WriteLine(item[“Title”]);
            }
          }
        }
```

```
        }
        catch (Exception ex)
        {
          return null; // new JObject(ex.Message);
        }
        return lista;

      }
    }

    public class LocationsItem
    {
      public LocationsItem()
      {
      }

      [PrimaryKey, AutoIncrement]
      public int databaseId { get; set; }
      public int ID { get; set; }
      public string Title { get; set; }
    }
}
```

Como veis es una clase muy simple con una función getList() que llama a SharePoint y carga la lista; pues ya tenemos todo listo para integrarlo con Xamarin.Forms, vamos a crear una página en Xaml:

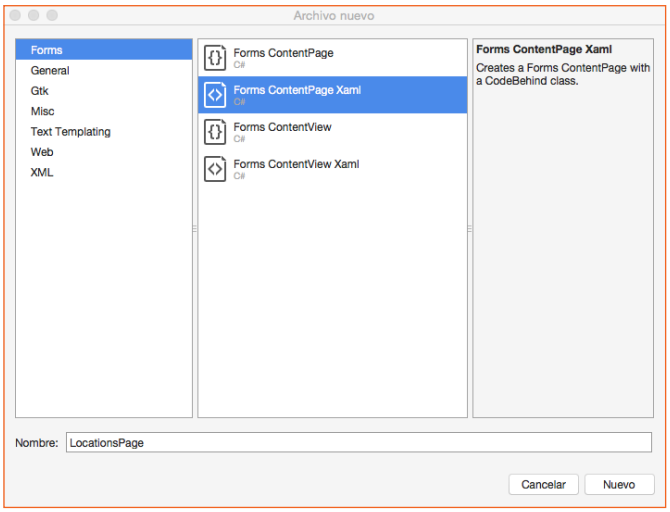


Imagen 3.- Añadiendo un nuevo formulario a la App.

Modificamos el XAML de la página para crear un listView (lo que queremos es sacar un listado de las ubicaciones de nuestra lista)

```
<?xml version="1.0" encoding=»UTF-8»?>
<ContentPage xmlns=»http://xamarin.com/schemas/2014/
forms» xmlns:x=»http://schemas.microsoft.com/winfx/2009/
xaml» x:Class=»shApp.LocationsPage»>
<ContentPage.Content>
    <StackLayout>
      <ListView x:Name=»locationsList»>
        <ListView.ItemTemplate>
          <DataTemplate>
            <ViewCell>
              <ViewCell.View>
                <StackLayout Orientation=»Horizontal»>
                  <Label Text=»{Binding Title}» />
                </StackLayout>
              </ViewCell.View>
            </ViewCell>
          </DataTemplate>
        </ListView.ItemTemplate>
      </ListView>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

y en el código de la página llamamos a nuestra clase

```
using System;
using System.Collections.Generic;

using Xamarin.Forms;

namespace shApp
{
  public partial class LocationsPage : ContentPage
  {
    public LocationsPage()
    {
      InitializeComponent();
      bindData();
    }

    public async void bindData()
    {
      List<LocationsItem> lista = new List<LocationsItem>();
      lista = await List_Locations.getList();
      if (lista != null)
        locationsList.ItemsSource = lista;
    }
  }
}
```

para realizar peticiones a los servicios REST de SharePoint haciendo uso de la clase System.Net.HttpClient

y el resultado es el siguiente:

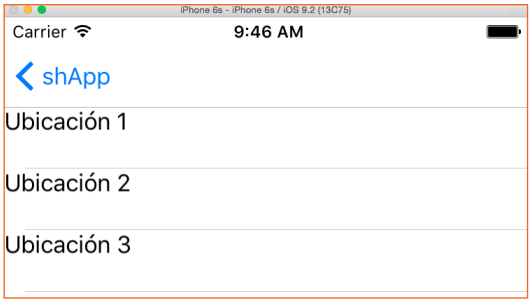


Imagen 4.- Aplicación resultante

Como veis nuestro código queda bastante claro y limpio.

Conclusiones

He intentado que el artículo fuese lo más simple posible y por tanto he obviado operaciones de inserción y modificación, si queréis saber cómo modificar o insertar nuevos elementos en la lista tan sólo tenéis que echar un vistazo a este artículo https://blog.vgrem.com/2015/04/04/consume-sharepoint-online-rest-service-using-net/ , es el artículo que he utilizado como base para mi código y en él hay ejemplos de diferentes operaciones sobre los items de una lista.

LUIS MOLINA MARTINEZ
Software engineer
micuentadecasa@gmail.com

04

Azure Task Scheduler: Planifica tus procesos en Azure

En el desarrollo de aplicaciones normalmente nos encontramos que debemos implementar tareas que deben ejecutarse periódicamente y que están fuera del entorno de nuestra aplicación. Estas tareas necesitan ser planificadas para su ejecución. En Windows utilizaríamos el Task Scheduler de Windows, y si tuviéramos muchas tareas utilizaríamos Frameworks como Quartz[1] que nos ayudan a planificar todas las tareas desde un solo punto y no una a una.

En entorno On-premise lo tenemos claro, pero si estoy en Azure, ¿cómo lo hago? La respuesta es Azure Task Scheduler.

Terminología

Antes de continuar es necesario conocer la terminología que envuelve a Azure Task Scheduler:

- Job –> Define la configuración de cómo y cuándo debe ejecutarse una tarea.
- Job Collection –> Colección de Jobs que permite compartir configuraciones y cuotas entre los Jobs.
- Job History –> Aquí encontraremos todos los detalles de las ejecuciones de los Jobs.

¿Qué nos ofrece azure task scheduler?

1.– Planificar la ejecución de tareas según sea deseado: periódicamente, una sola vez o durante un intervalo de tiempo.
2.– Planificar tareas tanto que estén alojadas en Azure como fuera de Azure.
3.– Configuración de reintento en el caso de que no se haya podido ejecutar la tarea.
4.– Historial de ejecuciones donde se nos informa del estado del Job, detalles de la ejecución, número de reintentos, periodicidad y hora de inicio y fin de la ejecución.
5.– Realizar acciones correctoras al suceder un error y finalizar los reintentos.
6.– Cinco formas de ejecución: Http, Https, Sotrage Queues, Azure Service Bus Queues y Azure Service Bus Topics.
7.– Alta disponibilidad.

Azure task scheduler en detalle

Ahora vamos a entrar en detalle de cómo podemos crear un Job, que opciones tenemos para hacerlo y entraremos en detalle en como configurar cada opción disponible.

Para crear y configurar un Job y/o su Job Collection tenemos tres opciones:

1.– Portal de Azure.
2.– API REST
3.– .NET SDK.

Portal de Azure

Los pasos para realizar la configuración desde el Portal de Azure son:

1.– Iniciamos sesión en el portal de azure[2].
2.– Seleccionamos +Nuevo y en el buscador ponemos scheduler y seleccionamos la primera opción.

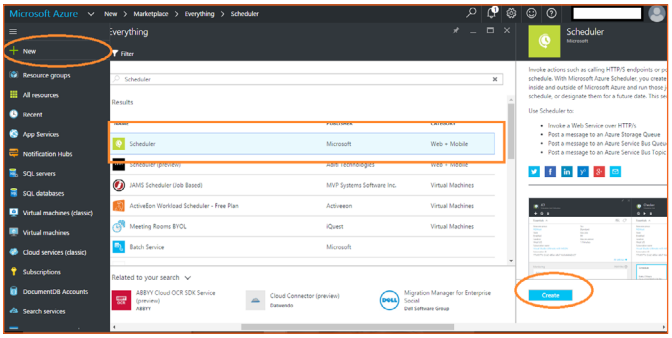


Imagen 1.- Creación scheduler en el portal.

3.– Los campos a rellenar son:
- Nombre del Job.
- Seleccionar la subscripción.
- Seleccionar o crear la Job Collection a la que queremos asociar el Job.
- Configuramos el Job: Tipo de acción, autenticación, política de reintentos y acción a realizar frente al error.
- Programamos el Job.

Api rest

Es necesario realizar una llamada a la URL con PUT como HTTP Verb:

https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Scheduler/jobCollections/{jobCollectionName}/jobs/{jobName}?api-version=2016-01-01

Es obligatorio enviar la cabecera Content-type, donde

ARGUMENTO	DESCRIPCIÓN
subscriptionId	El identificador de la subscripción donde quieres crear el Job.
resourceGroupName	El nombre del resource group al que quieres vincular al Job.
jobCollectionName	El nombre de la Job Collection a la que pertenecerá el Job.
jobName	El nombre del Job.
api-version	La versión de la API REST a la que quieres realizar la petición.

Y en el body enviaremos la especificación del Job:

```
{
  "properties": {
    "startTime": "2016-08-09T00:0:00Z",
    "action": {
      "request": {
        "uri": "http://yourURL.com",
        "method": "GET",
        "headers": {}
      },
      "type": "http",
      "retryPolicy":{"retryType":"Fixed", "retryInterval":"PT1H",
"retryCount":5}
    },
    "state": "enabled",
    "recurrence":
    {
      "frequency": "minute",
      "interval": 30,
      "count": 10
    },
  }
}
```

nos encontramos que debemos implementar tareas que deben ejecutarse periódicamente

Donde:

PARÁMETRO	DESCRIPCIÓN	
startTime	Fecha supuesta de inicio. Si no se informa se considerará que el Job se ejecutará en el mismo momento de su creación.	Opcional
action	La acción del Job.	Obligatorio
type	El tipo de la acción con la que se ejecutará: http, https, storage que, azure service bus queue o azure services bus.	Obligatorio
request	Los parámetros necesarios para la ejecución del Job. Dependerá del tipo de acción a realizar. Más adelante explicaremos con más detalle estos parámetros.	Obligatorio
retryPolicy	Política de reintento. Se debe especificar: ✓ retryType: Tipo de reintento: Fixed o None. ✓ retryInterval: Intervalo del reintento ✓ retryCount: Cuántas veces de debe reintentar.	Opcional
recurrence	Planificación del Job.	Opcional
state	El estado del Job. Puede ser Activado o Desactivado.	Obligatorio

Antes de crear el Job con la API REST, y a diferencia de hacerlo por el portal, debéis tener creada la Job Collection, podéis ver cómo hacerlo en: https://msdn.microsoft.com/en-us/library/mt629159.aspx

.NET SDK

Por último, podemos crear el Job y su collection con C#. Para ello primero se debe instalar el paquete Nuget Microsoft.Azure.Management.Scheduler.

Una vez instalado podemos crear la Job Collection:

```
var schedulerServiceClient = new SchedulerManagementClient(yourCredentials);
var result = schedulerServiceClient.JobCollections.CreateOrUpdate("resourceMOSS", "jobCollectionMOSS", new JobCollectionDefinition()
        {
          Location ="West Europe",
          Name = "jobCollectionMOSS",
          Properties = new JobCollectionProperties()
          {
            Quota = new JobCollectionQuota()
            {
              MaxJobCount = 3,
              MaxJobOccurrence = 5,

              MaxRecurrence = new JobMaxRecurrence()
              {
                Frequency = RecurrenceFrequency.Minute,
                Interval = 1
              }
            },
            Sku = new Sku(SkuDefinition.Standard),
            State = JobCollectionState.Enabled
          }

        });
```

Y después crear un Job a esta collection:

```
var schedulerClient = new SchedulerManagementCli-
ent(yourCredentials);
var resultJob = schedulerClient.Jobs.CreateOrUpdate("resourc-
eMOSS", "jobCollectionMOSS", "JobMOSS", new JobDefini-
tion()
      {
        Properties = new JobProperties()
        {
          Action = new JobAction()
          {
            Type = JobActionType.Http,
            Request = new HttpRequest()
            {
              Headers = new System.Collections.Generic.Dictio-
nary<string,string>()
            {

            },
              Method = "GET",
              Uri = new Uri("http://yourURL.com")
            }
          },
          StartTime = DateTime.UtcNow,
          Recurrence = new JobRecurrence()
          {
            Frequency = RecurrenceFrequency.Minute,
            Interval = 1,
            Count = 5
          },
          State = JobState.Enabled
        }

      });
```

En los ejemplos habréis visto muchos campos a configurar, no os preocupéis que ahora entramos en detalle a explicar para cada tipo de acción las opciones que hay disponibles.

Ahora veremos más en detalle como configurar el Job en base a la acción seleccionada y como planificarlo. Me basaré en cómo hacerlo en el portal de Azure, pero es extrapolable tanto a la API REST como al .NET SDK.

Configuración del job en base a la accion seleccionada

- Http Action:

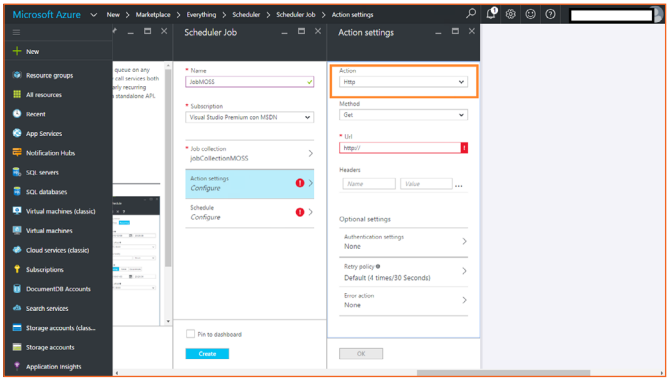


Imagen 2.- Job con tipo de acción Http.

Utilizaremos este tipo de acción cuando nuestro proceso puedas ser ejecutado a través de URL, como por ejemplo si tenemos un Azure WebJob. Debemos seleccionar que tipo de Http Verb utilizaremos, las opciones son:

- GET
- POST
- PUT
- DELETE

Una vez seleccionado el Http Verb a utilizar añadimos la URL a la que queremos llamar. Para los Verbs POST y PUT podemos añadir un body a la petición. Al seleccionarlo en el menú nos aparecerá la opción body donde podremos poner está información. También tenemos la opción de añadir cabeceras (headers), por ejemplo:

Content-Type: text/plain.

Por último, tenemos una sección de propiedades opcionales que incluye: autenticación, política de reintentos y acción ante el error.

- Autenticación: Si necesitamos que al realizar la llamada se envíe en la cabecera de la petición o en el body según el Http Verb elegido configuraremos esta opción que tiene cuatro opciones:
  - None: No necesita autenticación.
  - Basic: En esta opción realizaremos la autenticación mediante usuario, por lo que debemos rellenar los campos username y password.
  - Client Certificate: Realizamos la autenticación mediante certificado. Debemos informar los campos pfx y password.
  - Active directory OAtuh: La autenticación se realizará mediante OAuth contra Active Directory, para ello deberemos informar todos los campos necesarios para realizar ele OAuht sobre AD que son: Tenant, ClientId, Secret y Audience-
- Política de reintentos: Configuraremos que queremos hacer en el caso que al realizar la petición haya un error. Tenemos tres opciones:
  - None: No realizaremos ningún reintento.
  - Default: Por defecto se reintenta cada 30 segundos y el número máximo de reintentos es 4. Estos valores no se pueden modificar.
  - Custom: Definimos cuantas veces queremos reintentar y cuál es el intervalo de reintento. En este caso podemos reintentar cada x segundos, minutos, horas o días.

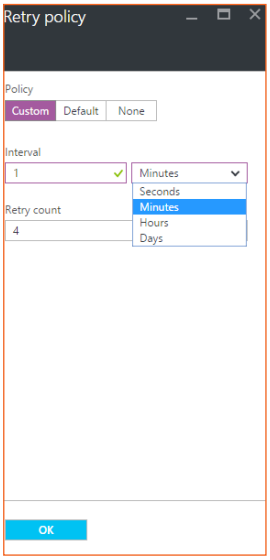


Imagen 3.- Política de reintentos.

- Acción en caso de error Acción que queremos realizar en caso de error. La acción que se pueden realizar es lanzar algún otro tipo de Job que tengamos para indicar este error o realizar alguna acción correctora, por lo que de nuevo tendremos que definir qué tipo de acción a realizar queremos con su respectiva configuración:
  - Http.
  - Https.
  - Storage queue.
  - Azure Service Bus queue.
  - Azure Service Bus topic.

Definiendo todas estas opciones ya tendríamos listo nuestro Job que llamaría a una URL concreta donde estaría nuestro proceso a ejecutar.

Https Action

La configuración para Https es exactamente la misma que la anterior con la diferencia que la URL a la que debemos llamar es https.

Storage queue

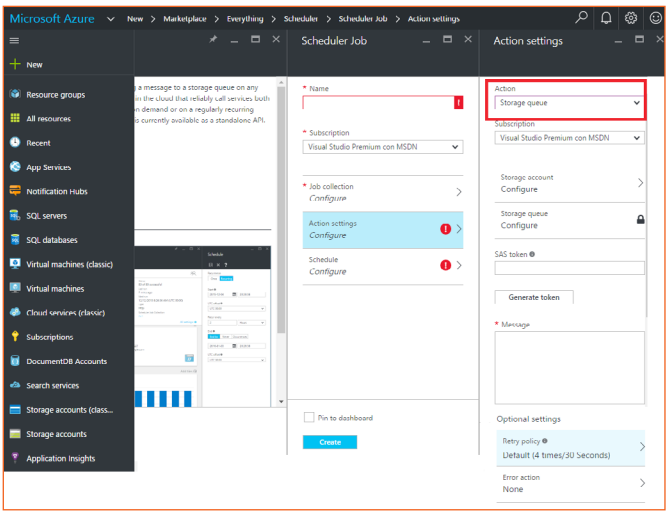


Imagen 4.- Job con tipo de acción Storage queue.

Utilizaremos este tipo de acción cuando nuestro proceso esté alojado en un WorkerRole por ejemplo. Para poder utilizar este tipo de acción primero debemos tener creado un Azure Storage account y tener creada una queue en este storage account.

Una vez realizado todos estos pasos añadiremos el Message, que será el mensaje que se enviará en la queue

Para este tipo de acción debemos seleccionar el storage account que queremos utilizar y acto seguido debemos seleccionar la queue que queremos usar. Después de esto para que el Job pueda insertar un mensaje en la queue debemos tener una Shared acces key (SAS) que le dé permiso para escribir en ella. Si no la tenemos disponemos de un botón que nos creará una por nosotros. Después insertaremos el mensaje que queremos enviar para que nuestro proceso sepa si debe ejecutarse o no. Este mensaje es un string del XML serializado de la clase StorageQueueMessage, donde nos viene información de ejecución (ExecutionTag y ClientRequestId), cuando se ha ejecutado (ExpectedExecutionTime), información sobre el Job (SchedulerJobId y SchedulerJobCollectionId), la región donde está alojada la Job Collection (Region) y por último el mensaje de la queue (Message).

```
<?xml version = “1.0” encoding=»utf-16»?>
<StorageQueueMessage xmlns:xsd=»http://www.w3.org/2001/
XMLSchema»
                xmlns:xsi=»http://www.w3.org/2001/XMLSchema-in-
stance»>
  <ExecutionTag>c3b67e748b93b0bac3718f1058e12907</Execu-
tionTag>
  <ClientRequestId>2fb66b67-e251-4c09-8d61-8627b8bf9bfd</
ClientRequestId>
  <ExpectedExecutionTime>2014-01-13T22:32:30</ExpectedExe-
cutionTime>
  <SchedulerJobId>JobMOSS</SchedulerJobId>
  <SchedulerJobcollectionId>JobCollectionMOSS</Scheduler-
JobcollectionId>
  <Region>West Europe</Region>
  <Message>Message Job to execute</Message>
</StorageQueueMessage>
```

Por último, tiene propiedades opcionales que son:

- Política de reintentos.
- Acción en caso de error.

Ambas opciones se configuran de la misma forma que está explicada en Http Action.

Services Bus queue Action

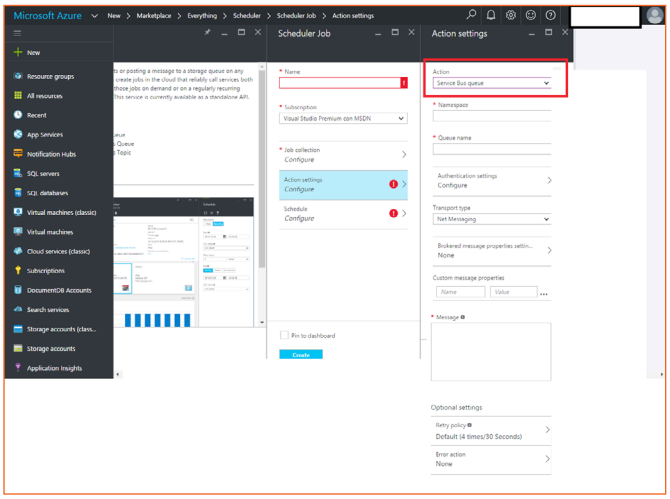


Figura 5.- Job con tipo de acción Service Bus queue.

Utilizaremos este tipo de acción cuando nuestro proceso esté alojado en un WorkerRole o tengamos un proceso On-Premise y necesitemos más potencia que Azure Storage queue por ejemplo. En https://azure.microsoft.com/en-us/documentation/articles/service-bus-azure-and-service-bus-queues-compared-contrasted/ explica qué características tiene cada una y cuando utilizarlas.

En este caso, primero necesitaremos tener creado un Azure Service Bus y una queue en este Service Bus. Una vez creado podemos introducir Namespace el namespace de

nuestro Service Bus y en Queue name el nombre de nuestra queue.

Después debemos introducir la configuración de la autenticación para ello deberemos introducir dos campos:

- SAS key name: Nombre del Shared Acces Key que previamente habremos creado en nuestro Azure Service Bus.
- SAS Key: Ser la Shared Acces Key generada.

Una vez definida la autenticación configuraremos que tipo de transporte queremos, las opciones son:

- Net Messaging
- AMQP

El siguiente paso es definir las propiedades de Brokered message, donde debemos definir:

- Content type: el content type del mensaje.
- Correlation Id: Identificador de correlación que nos servirá para filtrar los mensajes en el proceso, de forma que procese de golpe todos los mensajes de una mismo Correlation Id.
- Force persistence: Indica si el mensaje se debe guardar en base de datos o en memoria. Disponemos de tres opciones:
  - True: Se guarda en base de datos.
  - False: Mantenemos en memoria.
  - None: No aplica persistencia ninguna.
- Label: Etiqueta de la aplicación.
- Message id: Identificador del mensaje que servirá para detectar mensajes duplicados.
- Reply To: Define donde debe responder la aplicación en caso de aplicar la comunicación dúplex.
- Reply to sesión Id: El identificador de la sesión que el receptor debe marcar en el mensaje de respuesta para que el remitente pueda relacionarlo con una sesión concreta.
- Session Id: Identificador de la sesión.
- Tme to live: Tiempo de vida que tendrá el mensaje en la queue. Tiene dos opciones:
  - None: No se define TTL.
  - Set: Se define el TTL en x segundos, minutos, horas o días.

Acto seguido podemos añadir propiedades definidas por nosotros al mensaje.

Una vez realizado todos estos pasos añadiremos el Message, que será el mensaje que se enviará en la queue.

Por último, tiene propiedades opcionales que son:

- Política de reintentos
- Acción en caso de error

Ambas opciones se configuran de la misma forma que está explicada en Http Action.

Services Bus topic Action

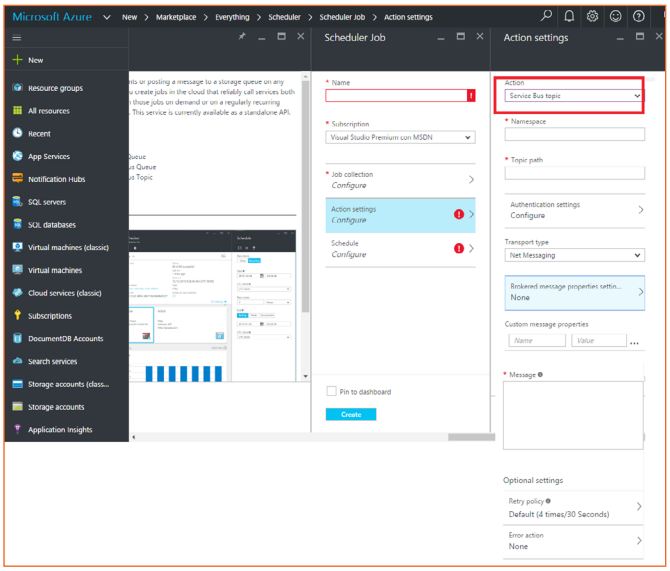


Imagen 6.- Job con tipo de acción Service Bus topic

La configuración es igual que la opción de Service Bus queue, excepto que en vez de poner el nombre de la queue pondremos el path del topic que utilizaremos.

¿Cuándo debemos utilizar un topic? A diferencia de Azure Service Bus queue donde el mensaje solo es procesado por un subscriptor, en la opción topic el mensaje será procesado por todos los subscriptores a este topic. Entonces está opción la debemos usar cuando tengamos varios procesos, ya sea en WorkerRole o en On-Premise que queremos que se ejecuten todos al recibir un mensaje concreto.

Planificación del job

Una vez introducido el nombre del Job, seleccionar la subscripción al que queremos vincularlo, vincular el Job a un Job Collection y selección el tipo de acción a realizar, solo nos queda planificar el Job.

Azure Task Scheduler nos ofrece dos opciones de planificación:

- Once: Definimos el día, hora y UTC offset que queremos que se ejecute. Solo se ejecutará una vez, la que hayamos definido.
- Recurring: Seleccionaremos esta opción cuando queramos ejecutar de forma periódica nuestro Job, para ello es necesario definir los siguientes campos:
  - Start: Fecha y hora de inicio.
  - UTC ofsset: UTC en el que deseamos que se ejecute.
  - End: Tennos tres opciones:
    - End by: Seleccionamos fecha, hora y UTC de finalización.
    - Never: Se ejecutará infinitamente.
    - Ocurrences: Cuantas veces queremos que se ejecute: 1,2,3…
    - Recur every: Cuantas veces queremos que se ejecute, aquí tenemos cinco opciones:

- Minutes.
- Hours.
- Days: Al seleccionar días nos da la opción de realizar una planificación avanzada donde podemos indicar a qué horas y minutos queremos que se lance el proceso. Por ejemplo: 10:30; 22:30. El proceso se lanzará cada día a las 10:30 de la mañana y a las 22:30 de la noche.
- Weeks: En esta opción también tenemos la opción de planificación avanzada, además de las opciones avanzadas de días nos da la opción de seleccionar que día o días de la semana queremos que se lance. Por ejemplo: lunes; viernes; 10:30; 22:30. El proceso se lanzará cada lunes y cada viernes a las 10:30 de la mañana y a las 22:30 de la noche.
- Months: Aquí también nos aparece la opción de planificación avanzada. Esta opción es la que más juego nos da, ya que nos permite planificar tareas a nivel de días del mes, a nivel del primer, segundo, tercer, cuarto o último día de la semana concreta, último día del mes…En definitiva un sinfín de posibilidades que cubren todas nuestras expectativas.

Destacar que siempre que deseemos podemos lanzar el Job de forma manual y que sigue las especificaciones ISO-8601 también en las fechas.

Alta disponibilidad

Para finalizar, Azure Task Scheduler proporciona alta disponibilidad dado que en el momento que se crea un Job en una región, automáticamente se crear una réplica en la región más próxima, de forma que si se produce algún error en la región solicitada automáticamente se ejecuta el Job replicado en la región más próxima garantizando la alta disponibilidad. Azure Task Scheduler aplica replicación geográfica para dar alta disponibilidad.

Conclusiones

Azure Task Scheduler es una forma sencilla y rápida para planificar la ejecución de nuestras tareas y poderlas modificar. La opción de reintento en caso de error, de acción correctora cuando esta suceda y de tener un historial detallado de las ejecuciones hace de este servicio de Azure uno de los más atractivos y útiles que hay.

Referencias

- http://www.quartz-scheduler.net/
- https://portal.azure.com/
- https://msdn.microsoft.com/en-us/library/azure/dn528937.aspx
- https://azure.microsoft.com/en-us/documentation/articles/scheduler-intro/
- http://fabriccontroller.net/a-complete-overview-to-get-started-with-the-windows-azure-scheduler/

ROBERT BERMEJO
Arquitecto .Net
bermejoblasco@live.com
@robertbemejo
www.robertbermejo.com

04

Power BI: Conexión con SQL y Oracle

Power BI ofrece múltiples servicios de conexión a diversos orígenes de datos. En este artículo nos centraremos en la conexión a bases de datos On Premises de SQL Server y Oracle. Como es de público conocimiento contamos con dos plataformas de desarrollo para Power BI (Power BI Desktop y Power BI Web), además del componente de refresco de información (Gateway).

Power BI Web

El componente Power BI Web, tal como lo dice su nombre es un componente accesible desde cualquier browser [1]. Este componente está orientado a la generación de Dashboards y reportes a través de la utilización de Servicios pre-establecidos a distintos orígenes de datos.

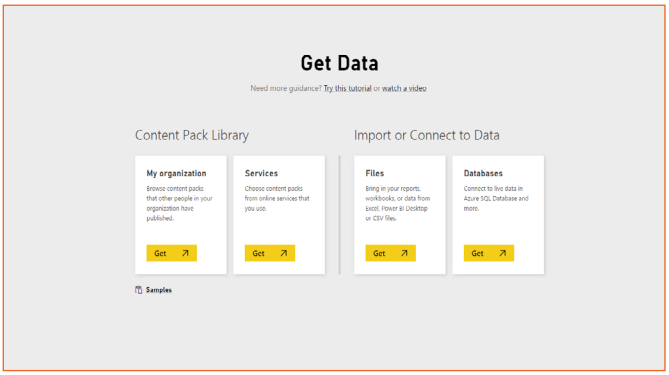


Imagen 1.- Power BI Web.

Dentro de estos orígenes se destacan los servicios para GitHub, Google Analytics, Azure, etc.

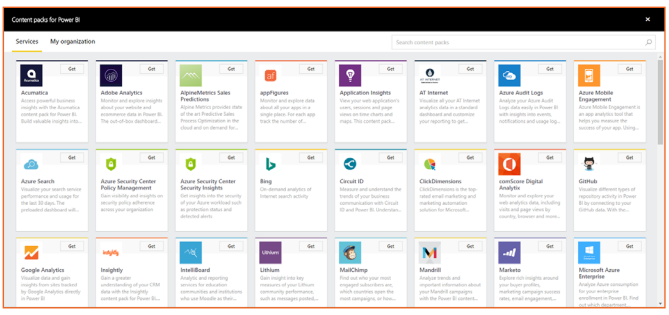


Imagen 2.- Orígenes de datos en Power BI.

Al estar orientado a los servicios, este componente no ofrece soporte para la conexión a orígenes On Premises, aunque si permite (una vez creado los reportes en Power BI Desktop) realizar modificaciones sobre los reportes y estructuras derivados de estos orígenes.

Power BI Desktop

Por contraparte, Power BI Desktop, aplicación de escritorio de Power BI, permite la conexión con sources On Premises, en los cuales se destacan SQL Server y Oracle (existe una variedad de orígenes a los cuales se puede generar conexión de datos).

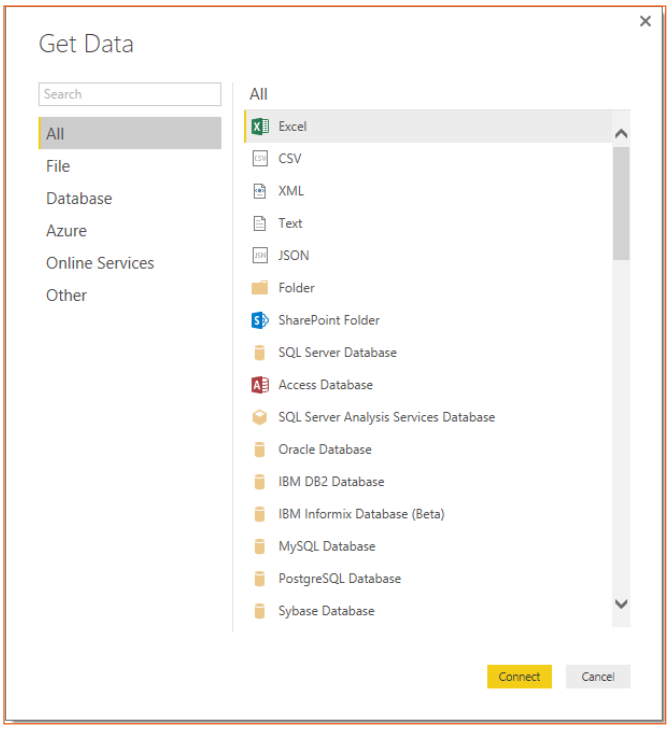


Imagen 3.- Conexiones de datos.

Debido a estas características, se detallará a continuación la interconexión de datos a través de Power BI Desktop.

Power BI ofrece múltiples servicios de conexión a diversos orígenes de datos

Conexión a SQL

La conexión a una base de datos On Premises de SQL Server [2], consiste en tres simples pasos. El primero es elegir el origen de datos, el cual podemos establecer a través de la opción Get Data-> SQL Server Database.

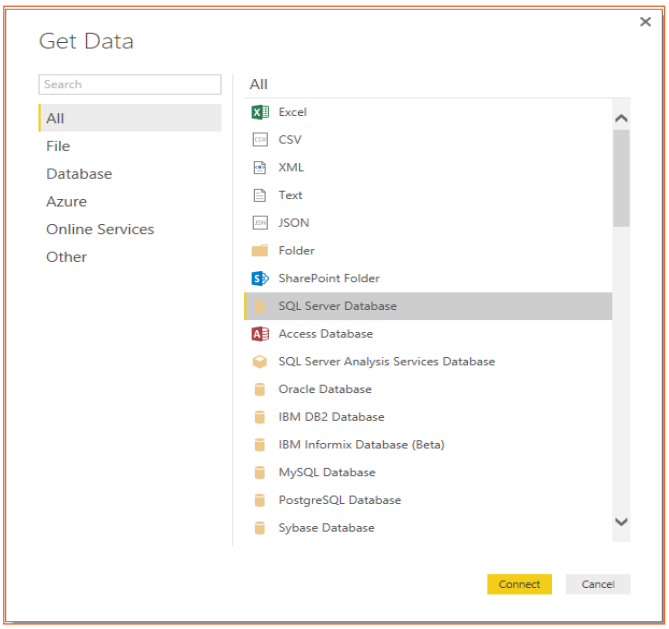


Imagen 4.- Selección del origen de tipo SQL Server Database.

En el segundo paso debemos especificar la IP o HostName del servidor de la Base de Datos y el nombre de la misma. También se deberá especificar si la importación de datos se realizará de forma Directa (DirectQuery), es decir, live streaming de datos o se creará una importación de los datos (Import) al momento de creación y cada vez que se refresque el reporte.

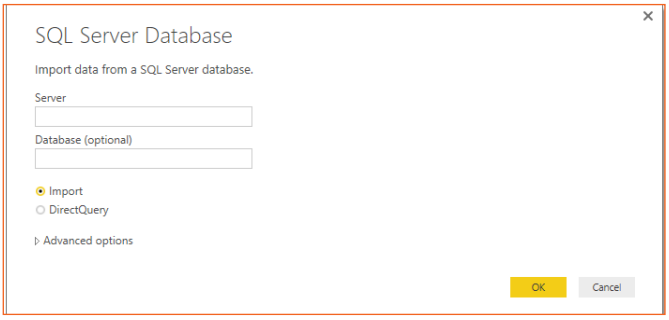


Imagen 5.- Parámetros de conexión a la BD de SQL Server.

De forma opcional, podemos realizar una consulta a la base de forma de traer la información necesaria de una manera ordenada y ya procesada. Esto lo podremos realizar utilizando la opción Advanced options y especificando la consulta en el cuadro de texto.

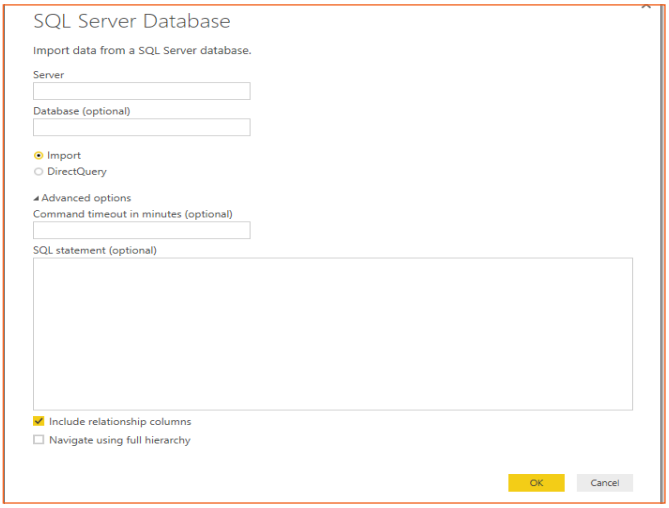


Imagen 6.- Opciones avanzadas para realizar la conexión a la BD SQL Server.

Una vez completados los datos de conexión debemos especificar las credenciales de ingreso a la base. Para ello contamos con dos opciones, a través de la utilización de Autenticación de Windows…

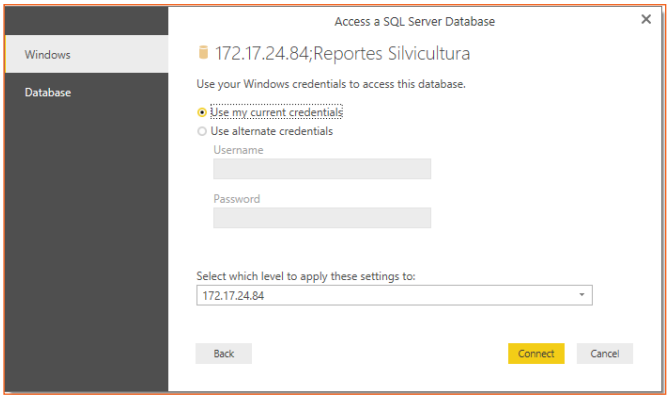


Imagen 7.- Parámetros de conexión a la BD.

… o a través de autenticación directa sobre la base de datos.

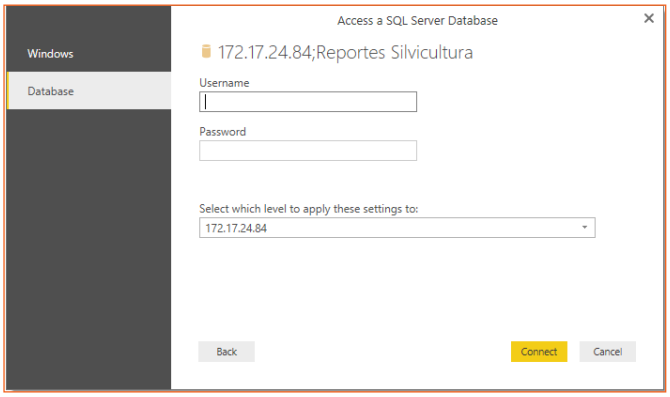


Imagen 7.- Conexión con usuario de BD.

Al completar los datos de ingreso, si los mismos son correctos, tendremos una pre-visualización de los datos de la base y sus estructuras.

Conexión a Oracle

Para realizar la conexión con una base de datos Oracle es necesario realizar ciertos pasos previos, debido a que se necesitan configuraciones específicas para poder establecer la conexión.

Como primer paso debemos descargar e instalar el Java Client[2], para ello debemos determinar que versión de Power BI tenemos instalada, si de 64 o 32 bits. Esto lo podemos obtener ingresando en Power BI Desktop, y dentro de la opción File -> Help -> About

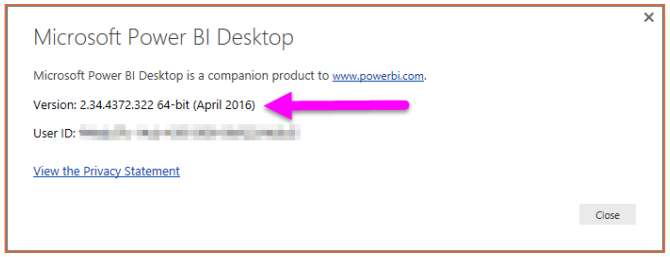


Imagen 8.- Versión de Power BI instalada.

Una vez instalado el Java Client, debemos configurar la co-

nexión a la base de datos de Oracle, esto se logra modificando el archivo TNSnames.ora (La ubicación del archivo dependerá de dónde instalamos el Java Client, dentro de la carpeta del Client se encuentra bajo la ruta …\client_1\Network\Admin). Dentro del archivo debemos especificar la conexión a través de la inserción de las siguientes entradas

```
[ALIAS] =
(DESCRIPTION =
(ADDRESS =
(PROTOCOL = TCP)
(HOST = [HOST_NAME])
(PORT = [PORT])
)
(CONNECT_DATA =
(SERVER = DEDICATED)
(SERVICE_NAME = [SERVICE_NAME])
)
)
[ALIAS] = Nombre a utilizar en el Connection string del Data Source
[HOST_NAME] = Nombre o IP del servidor que contiene la Base de Datos
[PORT] = Puerto a utilizar
[SERVICE_NAME] = Nombre del Servicio del servidor de la Base de Datos
```

Para realizar la conexión con una base de datos Oracle es necesario realizar ciertos pasos previos

Al finalizar la configuración, estaremos en condiciones de realizar la conexión a la base de datos desde Power BI Desktop. Para ello debemos dirigirnos a Get Data-> Oracle Database

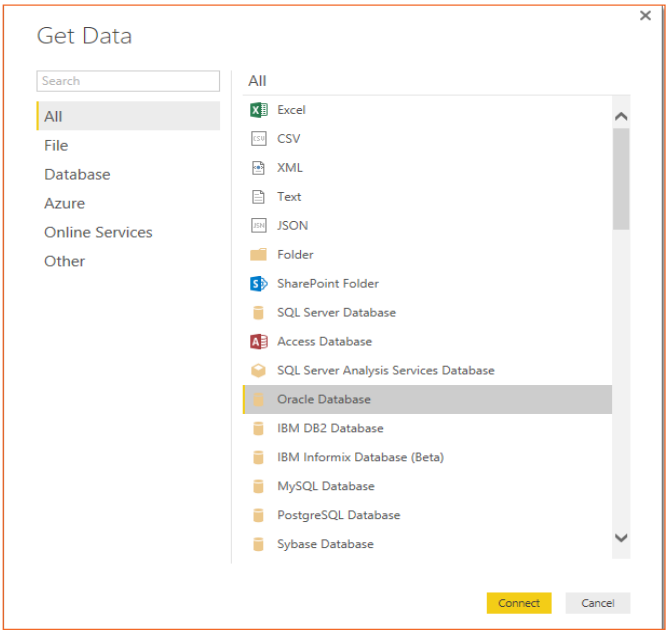


Imagen 9.- Selección del origen Oracle Database.

Para realizar la conexión debemos ingresar la IP del servidor seguido de /[ALIAS] especificado en el archivo TNSnames.ora.

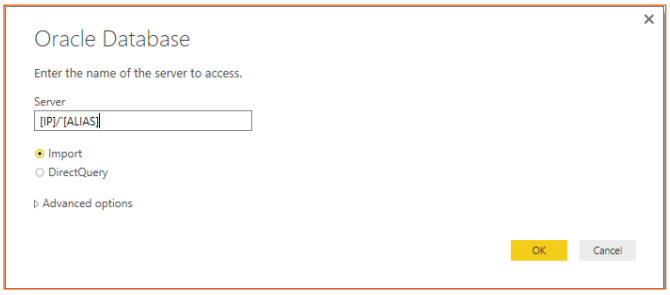


Imagen 10.- Parámetro de conexión del servidor de Oracle.

De forma análoga a SQL podremos especificar una consulta a la Base de datos en la sección Advanced options.

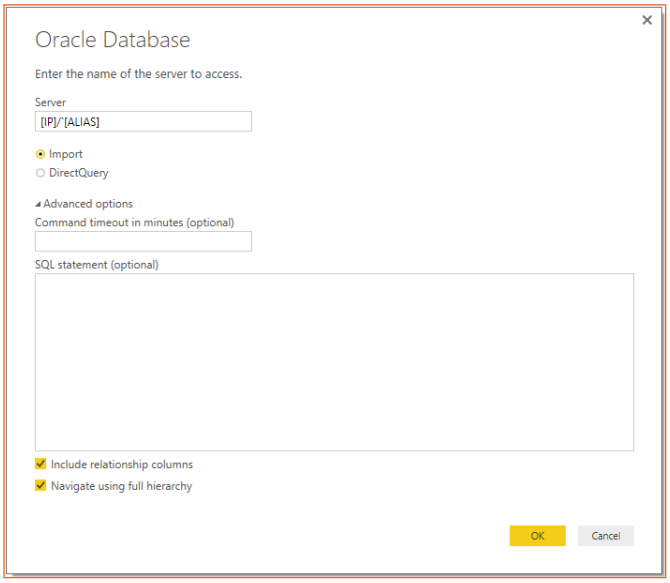


Imagen 11.- Opciones avanzadas de conexión a Oracle.

Referencias

- [1] Para ingresar a Power BI Web dirigirse a https://app.powerbi.com
- [2] En el momento sólo hay soporte para SQL Server 2012 y 2014, se está trabajando en la integración con SQL 2016
- [3] Para descargar el Java Client, ingresar a:
- 32 bits: http://www.oracle.com/technetwork/topics/dotnet/utilsoft-086879.html
- 64 bits: http://www.oracle.com/technetwork/database/windows/downloads/index-090165.html

BRUNO TORTEROLO
Software Developer en Arkano Software
Twitter: bruno_torterolo

47

Azure Information Protection

Siguiendo con el continuo crecimiento de servicios en Azure y pensando en facilitar la adopción de la nube a las empresas, Microsoft recién ha publicado un nuevo servicio para ayudarnos a proteger la información corporativa que podemos ver como una actualización de Azure Rights Management y que proviene de la adquisición de la empresa Secure Islands, este servicio se llama Azure Information Protection.

tenemos un servicio nuevo en Azure que nos permite clasificar, proteger y monitorizar los documentos de nuestra organización

Con Azure IP trabajaremos a nivel de documento para protegerlo de forma similar a como lo podemos hacer con DLP en Office 365, clasificando la información, etiquetando los documentos, añadiendo la protección necesaria en cada caso, monitorizando el acceso a la información y respondiendo ante posibles problemas de seguridad.

La protección, la monitorización y las respuestas son funcionalidades que ofrecía y ofrece Azure Rights Management, mientras que la clasificación y etiquetado son características nuevas que se agregan a este servicio.

Azure IP permite proteger el documento en origen, por parte del usuario o por mecanismos de clasificación automática, lo que permite mantener en todo momento el control de la clasificación del documento por parte del usuario, mientras que DLP aplica las políticas de protección en el momento del transporte, envío de un email o guardado del documento en una biblioteca de SharePoint, sin control por parte del usuario. La conjunción de estas dos técnicas permitirá mejorar la seguridad de la información en nuestra organización.

Azure IP permite configurar las etiquetas con la siguiente información:

- Plantilla de Azure RMS para esta etiqueta.
- Condiciones automáticas para aplicar esta etiqueta.
- Marcas visuales para aplicar al documento.

Para configurar estas etiquetas, primero tenemos que crear el servicio en una suscripción de Azure.

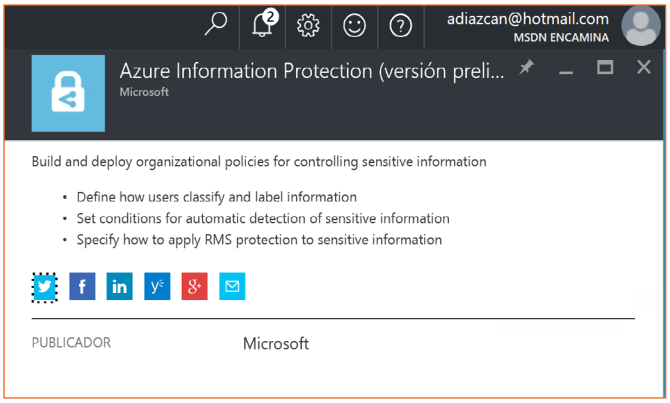


Imagen 1.- Descripción del servicio de Azure IP.

Una vez creado el servicio, tendremos un conjunto de etiquetas por defecto con las que ya podemos empezar a clasificar los documentos.

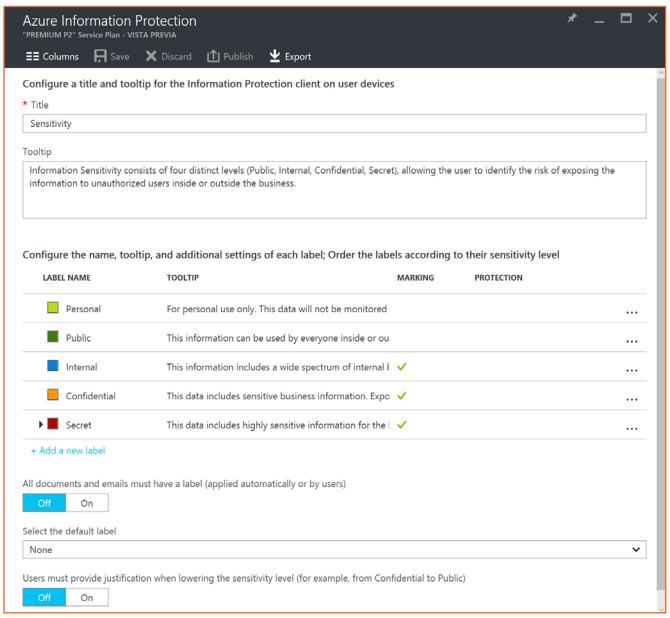


Imagen 2.- Configuración de Azure IP por defecto.

tendremos un conjunto de etiquetas por defecto con las que ya podemos empezar a clasificar los documentos

Para cada una de las etiquetas, vamos a poder configurar una descripción, un color identificativo y la plantilla de Azure RMS a aplicar.

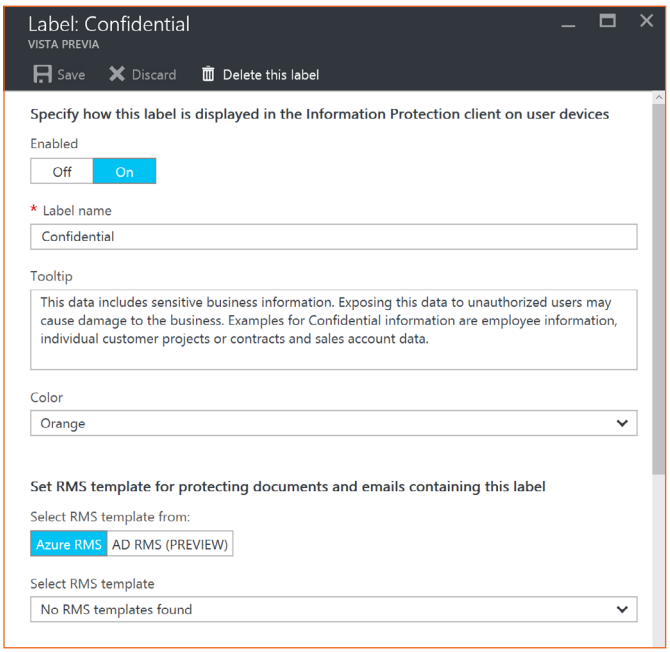


Imagen 3.- Propiedades de una etiqueta.

un usuario podrá clasificar la información de sus documentos en Office

Además de la marca visual y las condiciones de etiquetado automático.

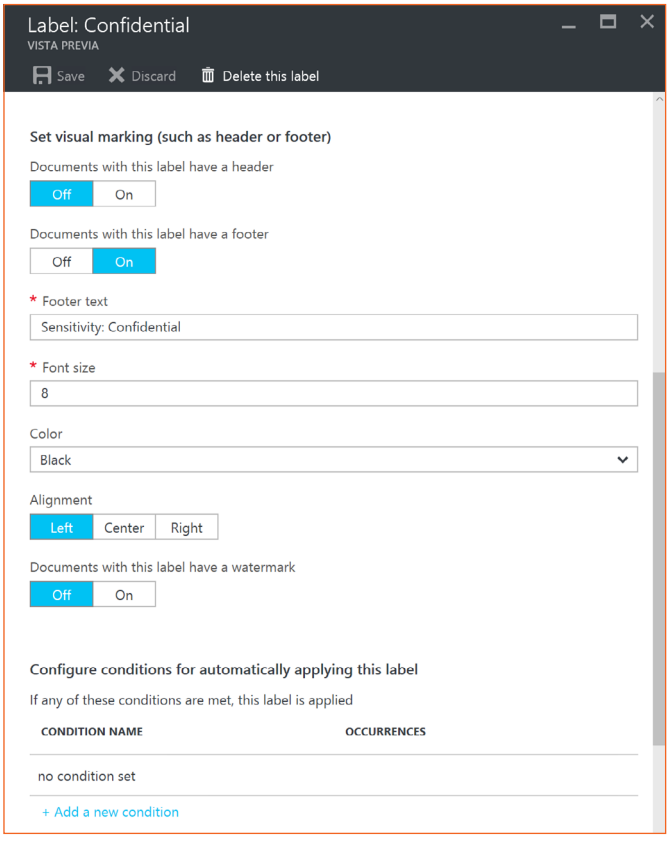


Imagen 4.- Continuación de las propiedades de una etiqueta.

Para las condiciones podemos usar reglas estándares, por ejemplo, tarjeta de crédito, o reglas personalizadas basadas en ocurrencias de texto o expresiones regulares.

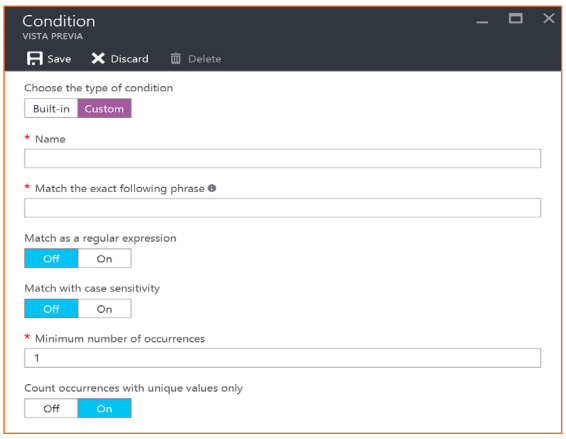


Imagen 5.- Condiciones automáticas de aplicación de una etiqueta.

Para usar el cliente de Azure IP, tenemos que instalar un complemento que se encuentra en la página de descarga https://www.microsoft.com/en-us/download/details.aspx?id=53018.

Con este complemento instalado, un usuario podrá clasificar la información de sus documentos en Office etiquetando los mismos con la clasificación necesaria o editando aquellos documentos que se hayan autoclasificado por alguna regla establecida en el servicio en Azure.

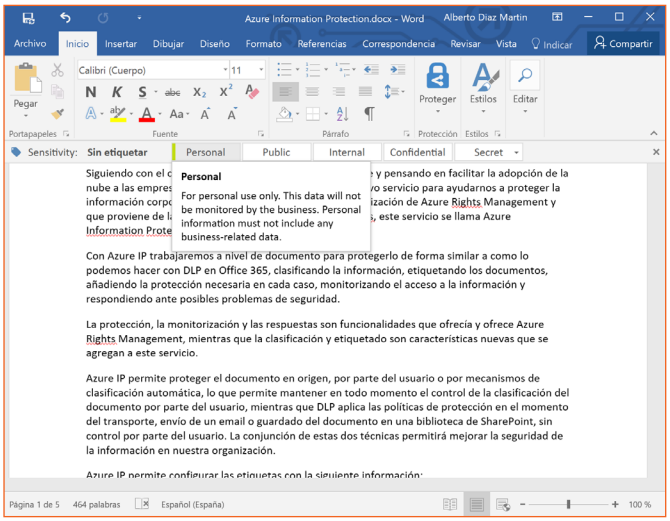


Imagen 6.- Panel de clasificación de un documento.

Por último, desde el portal de RMS vamos a poder hacer el seguimiento y monitorización de los documentos, además de poder revocar el acceso de los documentos si encontramos algún problema de seguridad.

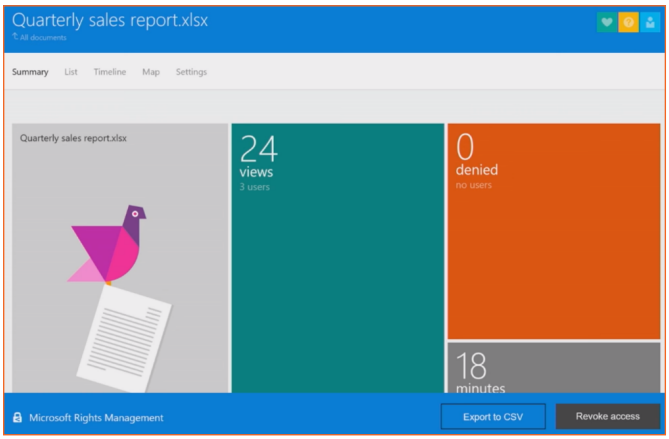


Imagen 7.- Panel de monitorización de un documento compartido.

Conclusiones

Azure IP es más que un servicio nuevo de la nube de Microsoft ya que complementa o actualiza las funcionalidades del actual Azure RMS buscando mejorar la experiencia de los usuarios a la hora de clasificar la información y complementando las funcionalidades de DLP de Office 365, todo esto para ayudar a evitar fugas de información en las organizaciones o a cumplir las políticas o leyes de los países aplicables a las empresas.

ALBERTO DIAZ MARTIN
MVP Office Servers & Services
adiazcan@hotmail.com
@adiazcan
http://blogs.encamina.com/negocios-sharepoint/
http://geeks.ms/blogs/adiazmartin

50

Nueva Experiencia de Usuario en Listas en SPO

Al igual que ha sucedido con las Bibliotecas de Documentos en SPO, Microsoft ha comenzado a liberar (para tenants de Office 365 en las que esté habilitado First Release) desde comienzos de agosto de 2016 la nueva experiencia de usuario para Listas de SPO completamente alineada con la experiencia ya disponible en Bibliotecas de Documentos. Se trata de una nueva interfaz fluida y rápida que además es responsive e incorpora integración en un solo clic con Microsoft Flow y Microsoft PowerApps. Adicionalmente, las denominadas "Listas modernas" de SPO son más "móviles" gracias a la aplicación móvil para SharePoint OnPremises y Online disponible actualmente para iOS.

Un paseo por la nueva experiencia de usuario en Listas de SPO

Para realizar un recorrido por la nueva experiencia de usuario en Listas de SPO, crearemos en un sitio de SPO una Lista de tipo personalizada denominada "Bibliografía" que almacenará información relativa a libros sobre Office 365, SharePoint y Azure:

- Desde el menú de Acciones del sitio o bien desde la nueva página de Contenidos del sitio, añadimos una aplicación de tipo "Lista personalizada" denominada "Bibliografía Recomendada" y navegamos a la misma. La primera vez que accedemos a una Lista que hace uso de la nueva experiencia, se muestra en un PopUp un resumen de las novedades que nos encontraremos en la nueva experiencia:

Imagen 1.- PopUp con el resumen de funcionalidades de la nueva experiencia de Lista.

- En el nuevo aspecto de la Lista podremos encontrar las siguientes novedades (Imagen 2):
  - Una caja de búsqueda ubicada sobre el menú vertical que facilita la búsqueda rápida de información en la lista.
  - Una barra rápida de acciones que cuenta con las siguientes opciones por defecto:
    - "Nuevo", para crear un nuevo elemento en la Lista.
    - "Edición rápida", que facilita editar varios registros de forma directa en la Lista de la misma forma que modificamos contenidos en una hoja Excel. Esta opción permite mantener la misma funcionalidad que ya conocíamos en la experiencia clásica de Listas.
    - "Exportar a Excel", que permite exportar a Excel el contenido de la Lista *(Nota: Esta opción se muestra sólo en Internet Explorer ya que a la fecha actual se sigue basando en el correspondiente control Active X)*.
    - "Flujo", que permite crear un Flujo de Microsoft Flow para la Lista. Al hacer clic en esta opción se muestra un panel en el que podemos seleccionar plantillas de Flujo disponibles por defecto, o que hayamos creado, o bien crear un Flujo desde cero.
    - "PowerApps" que permite crear una aplicación de PowerApps para visualizar información de la Lista o bien interactuar con el contenido de la Lista para añadir. Esta acción muestra de nuevo un panel para crear la PowerApp.
    - "Avisarme" que permite crear alertas en la Lista.
  - Acceso a las vistas disponibles en la Lista, crear nuevas vistas a partir de modificar la vista actual o bien acceder a la administración de las vistas creadas para la Lista.

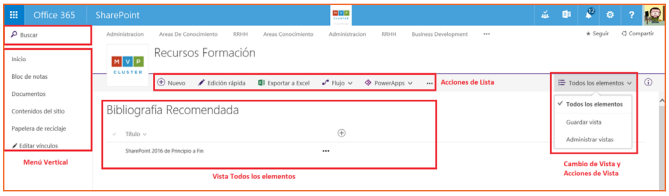


Imagen 2.- Nuevo aspecto de Listas de SPO.

desde comienzos de agosto de 2016 la nueva experiencia de usuario para Listas de SPO completamente alineada

- Creamos un nuevo elemento en la Lista a través de la opción “Nuevo”, lo que nos redirige al formulario clásico de “Nuevo elemento” para su creación. De vuelta al nuevo aspecto de la “Lista”, si seleccionamos el elemento creado veremos que:
  - Se muestra una barra de acciones específica para el elemento que contiene las siguientes acciones:
    - “Editar” que permite editar el elemento de lista a través del formulario clásico de edición de elemento.
    - “Compartir” que permite compartir el elemento a través del correspondiente diálogo. En este diálogo también podremos obtener un enlace al elemento y visualizar con quien se ha compartido.
    - “Obtener un vínculo” que muestra el mismo diálogo que para “Compartir” pero con la opción de obtener un enlace al elemento.
    - “Eliminar” que permite borrar el elemento seleccionado.
    - “Avisarme” para crear una alerta del elemento seleccionado.
  - Las mismas opciones de la barra de acciones del elemento de Lista y las siguientes acciones adicionales:
    - “Copiar campo en el Portapapeles” que permite copiar el campo “Título” en el portapapeles.
    - “Más”, opción que a su vez da acceso a las siguientes opciones:
      - “Flujo de trabajo” para agregar un Flujo “legacy” al elemento de Lista.
      - “Detalle de conformidad” que muestra una ventana en la que se visualiza la fase de retención en la que se encuentra el elemento.
  - El “Panel de propiedades” del elemento en el que podremos editar los metadatos del Elemento de lista y además visualizar el historial de actividad reciente en el elemento, si se ha compartido y con quien e información adicional como el Tipo de Contenido del Elemento, cuándo se modificó y la ruta de acceso.

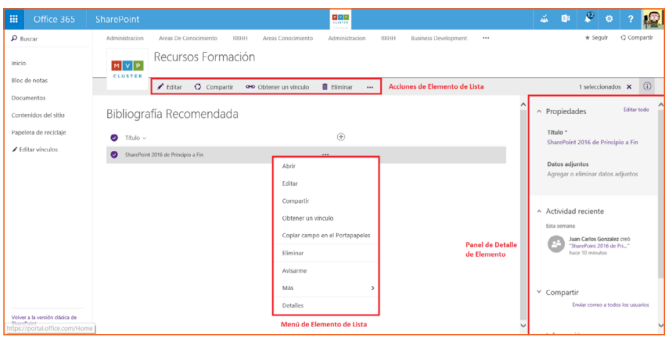


Imagen 3.- Barra de acciones y panel de detalle para un elemento.

- Una vez que hemos creado una Lista, podemos extenderla rápidamente desde la vista por defecto añadiendo nuevas columnas o bien cambiando la configuración de las columnas a mostrar o no en la vista:

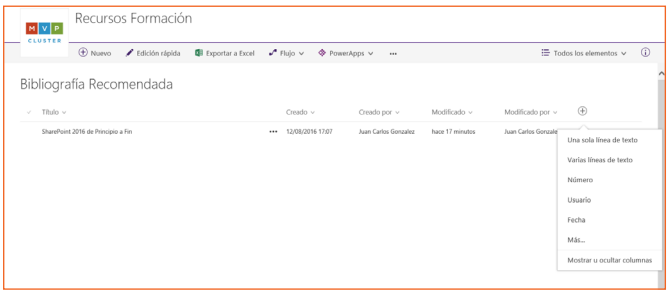


Imagen 4.- Como añadir nuevas columnas en la vista por defecto.

Por ejemplo, podemos añadir una columna de tipo elección para especificar la plataforma sobre la que trata el libro. Una vez añadida, podemos rápidamente configurar el orden de columnas y, como se indicaba antes, mostrar / ocultar otras columnas. Para que los cambios se guarden en la vista, tendremos que asegurarnos de que los mismos son guardados.

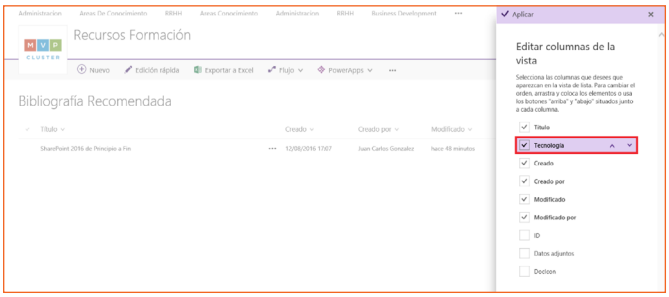


Imagen 5.- Panel de personalización de columnas de la Lista.

podemos añadir una columna de tipo elección para especificar la plataforma sobre la que trata el libro

Creación de Flujos y Aplicaciones para Listas

Junto con la nueva experiencia de usuario, una de las novedades más importantes que se incorporan en las Listas de SPO es la integración con Microsoft Flow y PowerApps *(Nota: Tanto Flow como PowerApps se encuentran actualmente en public preview)*:

- La integración con Flow permite automatizar tareas cuando se produce un suceso en una Lista (por ejemplo, se crea un elemento en la Lista) como puede ser crear información en otra Lista de otro sitio de SPO o en un sistema diferente como Microsoft Dynamics CRM Online, Salesforce o Project Online. Aunque en el momento de redacción de este artículo, Flow permite crear Flujos únicamente de tipo secuencial y no dispone de todas las posibilidades que SharePoint Designer 2013 (SPD 2013) provee para crear Flujos de Trabajo en Listas y Sitios de SPO, a futuro irá mejorando y permitiendo crear Flujos de Trabajo más complejos y que permitan dar respuesta a la necesidad de modelar procesos de negocio en Listas y Bibliotecas de Documentos de SPO.

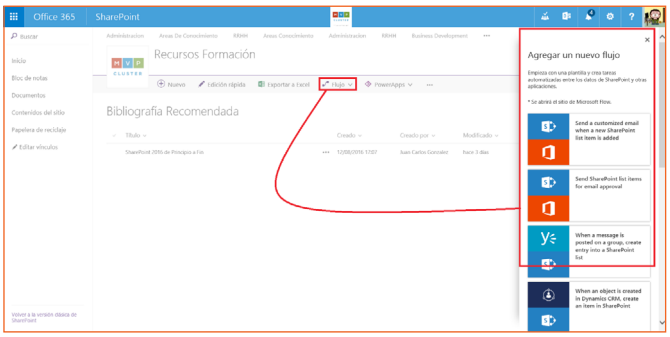


Imagen 6.- Integración de Microsoft Flow en Listas de SPO.

Cuando se hace clic en la opción "Flujo" de la barra de acciones de una Lista, se abre un panel en el que se muestran las plantillas de Flujos disponibles por defecto en Microsoft Flow o bien permite crear un Flujo desde cero.

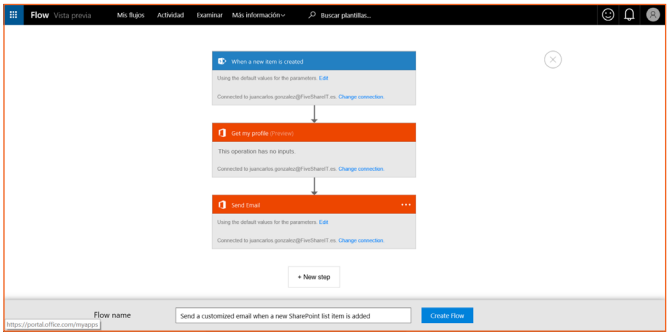


Imagen 7.- Ejemplo de Flujo por defecto para una Lista en Microsoft Flow.

- La integración con PowerApps permite crear rápidamente aplicaciones de negocio listas para ser utilizadas en cualquier dispositivo sin necesidad de escribir código. Al igual que sucede con Microsoft Flow, PowerApps permite conectar los datos de la Lista con datos de otras fuentes de datos ya sean OnPremises o cloud como Exchange, Dynamics CRM, Salesforece, Google, etc.

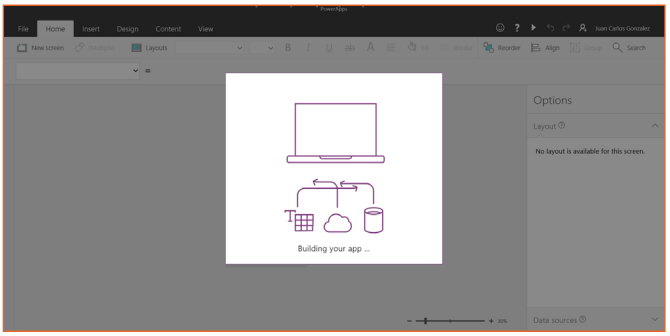


Imagen 8.- Diseñador Web de PowerApps.

Opciones de Administración de la nueva experiencia de usuario en Listas de SPO

Al igual que sucede con la nueva experiencia de usuario en Bibliotecas de Documentos en SPO, en la página de Contenidos del Sitio y en la Papelera de Reciclaje, disponemos de distintas posibilidades para habilitar o no la nueva experiencia de usuario de Listas:

- Para cada Lista, podemos acceder a la página de "Configuración de la lista" y a continuación a la opción "Configuración avanzada". Como se aprecia en la Imagen 10, podemos acceder a la administración de la biblioteca de documentos a través de las opciones de configuración globales del sitio.

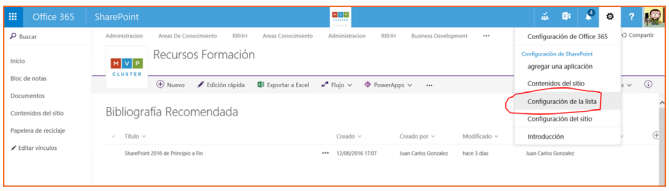


Imagen 9.- Acceso a la configuración de la lista.

En la página de "Configuración avanzada" localizaremos una nueva sección que nos permite establecer el tipo de experiencia de Lista a utilizar: "Experiencia predeterminada establecida por el administrador", "Experiencia nueva", "Experiencia clásica". La primera de las opciones es controlada por la configuración que se aplique de forma global a nivel de tenant de SPO como se verá en el siguiente punto.

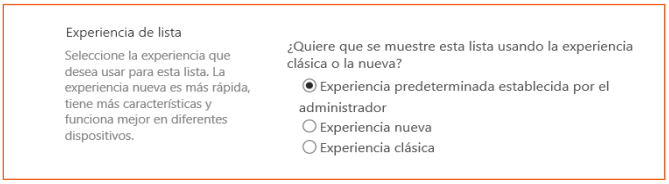


Imagen 10.- Opciones de configuración de "Experiencia de lista" en "Configuración avanzada".

- A nivel de tenant de SPO, se puede configurar de forma global la experiencia de usuario tanto para Listas, como para Bibliotecas de Documentos, la página de Contenidos del sitio y la Papelera de reciclaje. Las opciones de configuración globales están disponibles en la página de "configuración" disponible en la administración del tenant de SPO:

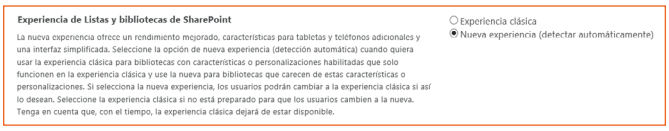


Imagen 11.- Configuración de la nueva experiencia de Lista y Bibliotecas de Documentos a nivel de tenant de SPO.

- Finalmente, la nueva experiencia puede ser habilitada/deshabilitada también de forma global a nivel de tenant de SPO haciendo uso de PowerShell tal y como se detalla en el siguiente artículo de soporte: https://support.office.com/en-us/article/Switch-the-default-experience-for-lists-or-document-libraries-from-new-or-classic-66dac24b-4177-4775-bf50-3d267318caa9

Lo que falta (o parece que falta) en la nueva experiencia de usuario de Listas

Al igual que sucede con las Bibliotecas de Documentos, la nueva experiencia de usuario de Listas desplegada por Microsoft en tenants con First Release configurado se trata de una versión preliminar de la experiencia que tendremos

finalmente desplegada en SPO. En la misma faltan elementos en los que Microsoft está trabajando o bien simplemente no estarán disponibles como, por ejemplo:

- En el momento de redacción de este artículo, no todos los tipos de Listas disponibles en SPO soportan la nueva experiencia. Como ejemplos, las Listas de tipo Calendario y Tareas por el momento no disponen de la misma.
- Para Listas existentes con un alto grado de personalización, no estará disponible la nueva experiencia. Por ejemplo, no se soportan todos los tipos acciones personalizadas que se hayan desarrollado como por ejemplo las acciones personalizadas que inyecten bloques de script.
- Soporte "completo" en la barra de navegación superior para que los distintos niveles de la misma se muestren independientemente de que se haya activado o no la característica de publicación en el Sitio.
- Posibilidad de aplicar branding a las Listas, Microsoft se ha comprometido a proporcionar mecanismos de proporcionar capacidad para personalizar las nuevas Listas, aunque por el momento se desconoce cómo se podrán hacer dichas personalizaciones.
- Soporte de varios niveles de agrupamiento, incluido de serie. En el momento en el que se dispone de columnas de tipo elección o lookup en la biblioteca se dispone de la opción de agrupar por dichas columnas a nivel de vista.

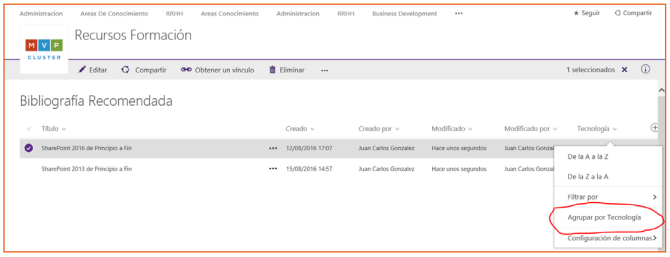


Imagen 12.- Ejemplo de agrupación en una Lista de SPO.

La integración con Flow permite automatizar tareas cuando se produce un suceso en una Lista

Conclusiones

La nueva experiencia de usuario en Listas para SPO supone un cambio radical no solo en cuanto a aspecto, sino también en cuanto a forma de trabajar colaborativamente con información contenida en las mismas. La clásica Ribbon a la que estábamos acostumbrados desaparece y es reemplazada por un menú horizontal de acciones que facilita el trabajo a nivel de Lista y de Elemento de Lista. A nivel de Elemento de Lista, el nuevo panel de edición simplifica no solo la visualización sus metadatos sino también visualizar su historial de actividad. Finalmente, la integración con Microsoft Flow y PowerApps permite extender la funcionalidad propia de las Listas con Procesos de Negocio y Aplicaciones Personalizadas que faciliten no solo visualizar la información almacenada, sino modificarla.

Referencias

- https://blogs.office.com/2016/07/25/modern-sharepoint-lists-are-here-including-integration-with-microsoft-flow-and-powerapps/

JUAN CARLOS GONZÁLEZ MARTÍN
Office Servers and Services MVP
Cloud & Productivity Advisor en MVP CLUSTER
jcgonzalezmartin1978@hotmail.com
@jcgm1978 | https://jcgonzalezmartin.wordpress.com/

04

Flujos y eventos para Project Server con Nintex

Hace tiempo que tenía ganas de hablar sobre las posibilidades que Nintex ofrece para Project Server. Y es que no solo nos permite realizar flujos de Gestión de la Demanda de manera sencilla, al igual que lo haríamos para SharePoint, sino que, además, permite lanzar flujos que respondan a diversos eventos de Project Server: cuando se ha publicado un proyecto, cuando se ha creado un recurso. Hasta la aparición de Microsoft Flow, sólo Nintex permitía enlazar con estos eventos sin la necesidad de recurrir a desarrollos con Visual Studio.

Flujos de Gestión de la demanda con Nintex

Al habilitar Nintex Workflows en una colección de sitios de Project Server aparece una nueva opción en el apartado de configuración. Desde la misma podremos tanto crear un nuevo flujo de gestión de la demanda como asociar un flujo a un evento de Project:

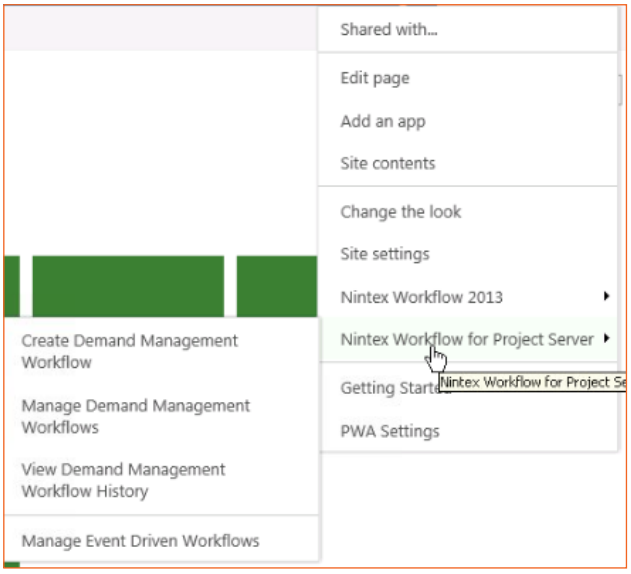


Imagen 1.- Opción para asociar/crear un flujo de Nintex para Project Server.

En el caso de un flujo de gestión de la demanda, las acciones que tenemos disponibles son las siguientes:

- Cambiar un tipo de proyecto: no nativo con flujos de SPD.
- Comparar una propiedad de un proyecto.
- Publicar un proyecto.
- Hacer una consulta a Project Server: no nativo con flujos de SPD. Se utiliza la PSI.
- Leer una propiedad de proyecto.
- Leer un grupo de seguridad: no existente con flujos de SPD.
- Indicar la etapa del proyecto.
- Indicar información de estado.
- Actualizar propiedades del proyecto.
- Actualizar una propiedad del proyecto.
- Esperar a que se proteja el proyecto.
- Esperar a que se confirme el proyecto (en la gestión de la cartera).
- Esperar a que se envíe el flujo (a la siguiente etapa)

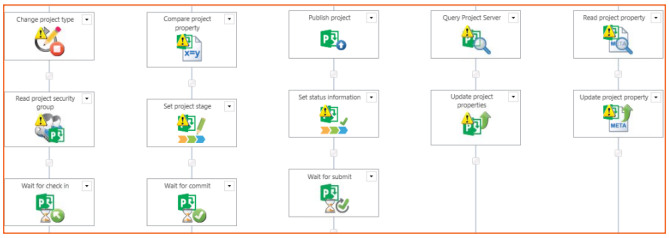


Imagen 2.- Ejemplos de acciones disponibles para Flujos de Nintex para Project Server.

Como se ha podido comprobar, la mayoría de acciones disponibles ya existen en un flujo creado con SharePoint Designer (SPD) a excepción de las tres no nativas (las cuales, además, son muy interesantes.) Pero, además, el resto de acciones de Nintex para SharePoint también están disponibles, lo que nos permite ampliar la funcionalidad de nuestros flujos integrándola con otros que se ejecuten sobre los sitios de proyecto, por ejemplo.

Este podría ser un ejemplo sencillo de flujo de Gestión de la Demanda con Nintex:

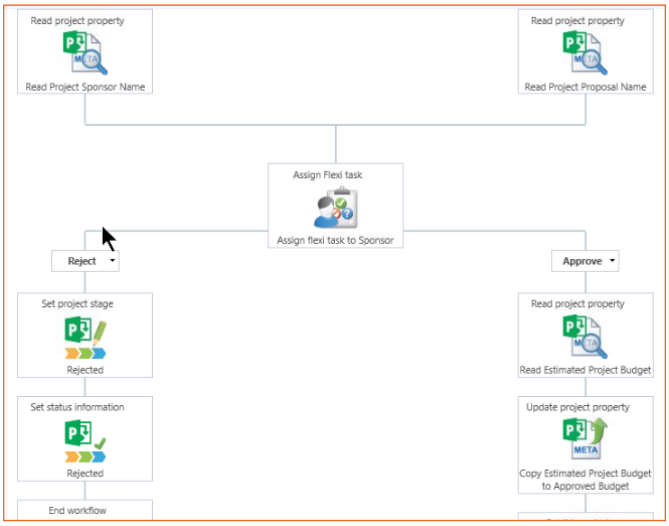


Imagen 3.- Ejemplo de flujo de gestión de la demanda con Nintex Workflow.

Al habilitar Nintex Workflows en una colección de sitios de Project Server aparece una nueva opción en el apartado de configuración

En este caso, se leen las propiedades del proyecto “Sponsor” y “Nombre de la propuesta” y se genera una tarea para el Sponsor en la que debe decidir si aprueba o no el presupuesto del proyecto. En caso negativo se rechaza el proyecto y finaliza el flujo y, en caso afirmativo, se lee la propiedad “Presupuesto estimado” y se copia en la propiedad “Presupuesto Aprobado”, continuando el flujo.

Flujos de Gestión asociados a eventos con Nintex

Como hemos comentado con anterioridad, hasta Microsoft Flow sólo Nintex ofrecía la posibilidad de sobre escribir los eventos de Project Server sin necesidad de desarrollos con Visual Studio. Esto lo hace especialmente potente ya que nos permite modificar u obtener datos de Project Server en respuesta a diferentes sucesos que suceden en el sistema.

En este caso, en la “Ribbon” aparecen las siguientes opciones, apareciendo los flujos y los eventos a los que se asocian bien diferenciados:

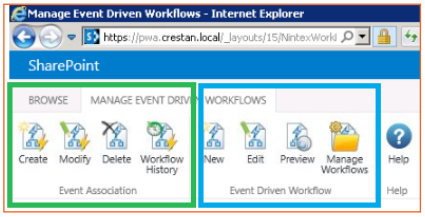


Imagen 4.- Eventos para flujos de Nintex para Project Server.

En ambos casos podemos hacer básicamente las mismas operaciones CRUD: crear, modificar y borrar tanto las acciones como los flujos. Lo primero que habría que hacer es crear un flujo para poder asociarlo a un evento. En este caso, las acciones propias de Nintex para Project Server quedan bastante reducidas:

- Hacer una consulta a Project Server.
- Obtener información del evento: por ejemplo, si el evento es el de publicación, el GUID del proyecto que se publica.
- Actualizar propiedades del proyecto.

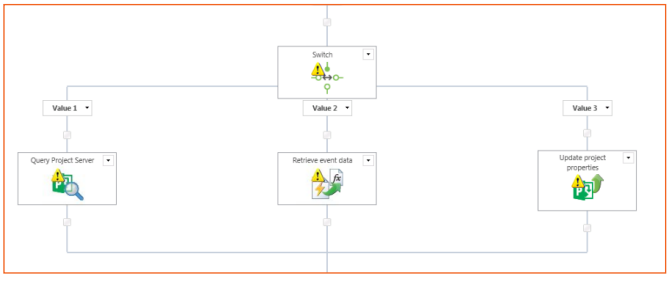


Imagen 5.- Ejemplos de acciones de flujos de Nintex para Project Server.

En la siguiente imagen podemos ver un ejemplo de flujo orientado a eventos con Nintex:

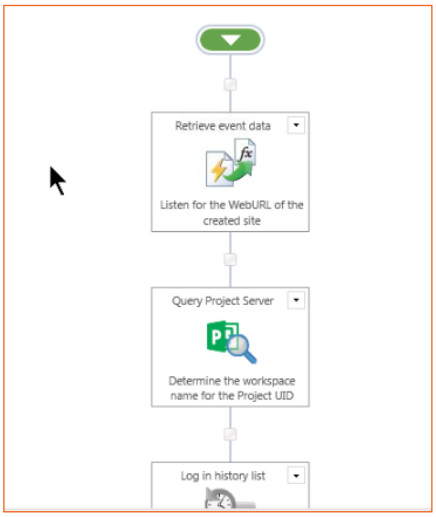


Imagen 6.- Ejemplo de flujo orientado a eventos.

Una vez hemos terminado nuestro flujo todavía nos falta asociarle un evento de Project Server que lo dispare. Para crear una nueva asociación, lo primero que hay que indicar es el tipo de evento que dispara la ejecución del flujo. Para ello, se debe seleccionar el objeto de Project Server sobre el que se desea ejecutar el workflow (Proyecto, Recurso, Calendario…):

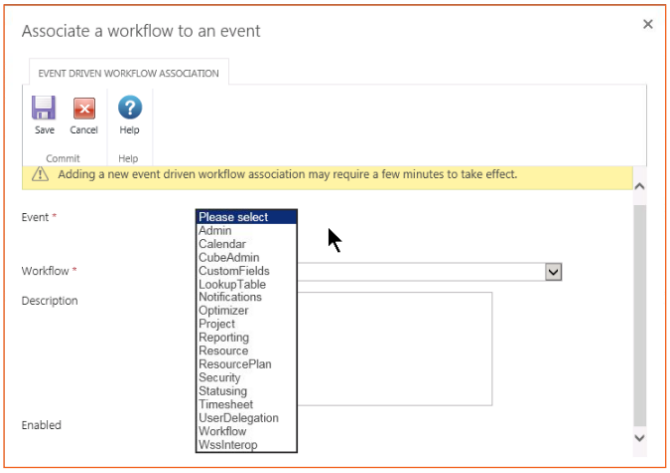


Imagen 7.- Formulario de asociación de flujo de Nintex.

Una vez seleccionado el objeto, se despliegan los diferentes tipos de eventos correspondientes al mismo:

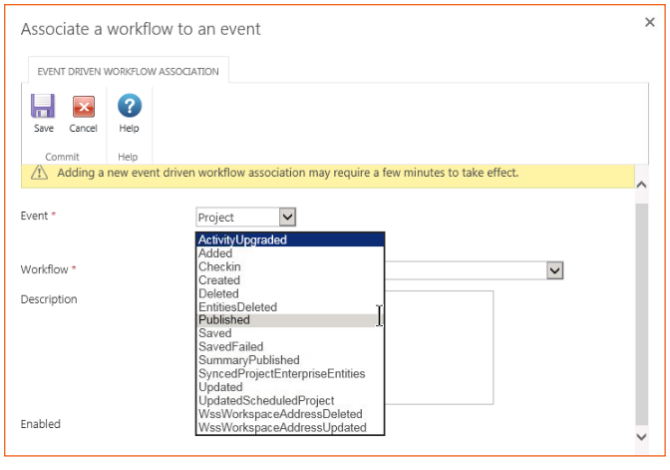


Imagen 8.- Eventos disponibles para el objeto seleccionado.

el uso de Nintex Workflow para Project Server nos permite aumentar las posibilidades que nos ofrece out-of-the-box Microsoft

Finalmente, hay que indicar el flujo que se desea lanzar cuando se ejecute dicho evento:

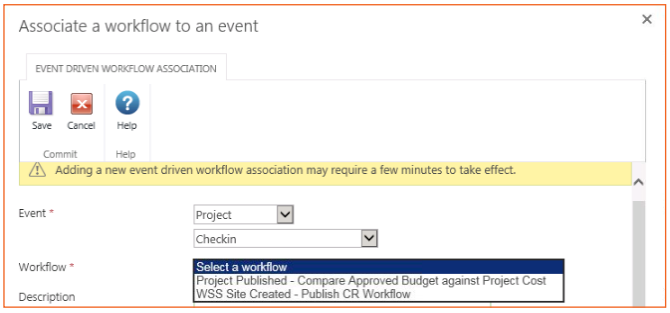


Imagen 9.- Selección del flujo a lanzar.

Conclusiones

Como se ha podido comprobar, el uso de Nintex Workflow para Project Server nos permite aumentar las posibilidades que nos ofrece out-of-the-box Microsoft para el desarrollo de flujos sobre este producto. No sólo nos facilita el diseñador de flujos de Nintex la creación de los mismos si no que, además, nos ofrece algunas acciones adicionales que sólo están disponibles con esta herramienta. Y, lo que es mejor, nos permite lanzar flujos capturando los eventos que se producen en la herramienta pudiendo así acceder a la información en tiempo de ejecución. De esta forma, se pueden realizar cálculos tras un evento y solicitar una aprobación o cambiar el valor de un campo basándonos en los mismos. O transformar y guardar la información de un proyecto o recurso tras su edición o publicación para la realización de informes.

JOSE RAFAEL GARCÍA
josex1975@gmail.com
https://projectservernotes.com/
@jrgarcia1975

04 Gestión de datos no relacionales en Microsoft Azure

Cuando una persona se adentra en el mundo de tecnologías de información se dará cuenta que, en los proyectos de software, por lo general siempre existen estos dos componentes:

- El software.
- Medios de almacenamiento.

El software puede estar escrito en cualquier lenguaje de programación de nuestra preferencia como .Net o Java. Los medios de almacenamiento pueden ser desde simples archivos planos de texto hasta estructuras más complejas como bases de datos, estos elementos se deben visualizar por separado no como una sola entidad.

El primer contacto que un desarrollador o administrador de bases de datos puede tener con respecto a la gestión de información, es con una "Base de datos relacional".

Las bases de datos relacionales son aquellas que almacenan información y la mantienen en estructuras cuadradas tipo tabla. Las empresas que se mantienen líderes en este ramo son:

Comerciales (SQL Server de Microsoft, Oracle DB de Oracle) y Open Source (MySQL, PostgreSQL).

Con la llegada de Microsoft Azure como plataforma del cómputo en la nube, podemos encontrar una gran cantidad de servicios que se ofertan, entre muchos de ellos tenemos un grupo enfocado a la gestión de información.

Microsoft Azure ofrece almacenamiento con SQL Server y Oracle DB con máquinas virtuales, además de un servicio relacional en la nube llamado SQL Azure.

Una de las muchas razones por las que la nube fue concebida es: para almacenar grandes cantidades de información. Información que difícilmente podría ser guardada y procesada en servidores locales, esta información puede ser tanto relacional como no relacional.

Las bases de datos relacionales son aquellas que utilizan mecanismos internos de almacenamiento que no estructuran la información a manera de tabla

Las bases de datos no relacionales, son aquellas que utilizan mecanismos internos de almacenamiento que no estructuran la información a manera de tabla. Su existencia se debe a necesidades de mejor performance y tiempos de respuesta que muchas veces en las bases de datos relacionales no se logran tan fácilmente.

Los servicios no relacionales que ofrece Microsoft Azure son:

DocumentDB

El servicio de DocumentDB ofrece un almacenamiento no relacional en texto plano con formato JSON. Las estructuras quedan categorizadas en contenedores llamados "Documentos".

Imagen 1.- DocumentDB.

Estas son algunas de sus características:

- JavaScript Core: El motor de ejecución de consultas de DocumentDB está basado en JavaScript completamente. Objetos ya conocidos en SQL Server como procedimientos almacenados, funciones o triggers también se encuentran en DocumentDB.
- Consultas con sintaxis SQL: DocumentDB soporta SQL tal cual como si trabajáramos desde SQL Server, por lo que al momento de extraer o procesar información, el ambiente nos será muy familiar.
- Administración supervisada: Esto significa que podemos administrar roles y permisos para usuarios en particular.
- Estructura: Todas las entidades almacenadas en DocumentDB mantienen el formato Json de manera nativa.
- Escalabilidad: DocumentDB se encuentra en la nube de Microsoft Azure, por lo que almacenamiento y rendimiento se regulan a través del tiempo de acuerdo a las necesidades llegando al orden de petabytes sin problemas.
- Tipo documental: Se utilizan elementos denominados documentos, no tablas.

El modelo de recursos por el cual funciona DocumentDB es el siguiente:

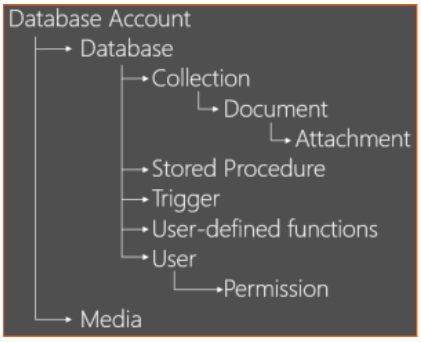


Imagen 2.- Modelo de recursos de DocumentDB.

- Data Base Account (Cuenta de base de datos)
- Se refiere a la cuenta principal con lo que se administra la base de datos DocumentDB.
- DataBase (Base de Datos)
- Es el contenedor lógico de usuarios, segmentado por colecciones.
- Collection (Colección)
- Contenedor de documentos JSON.
- Document (Documento)
- Objetos CRUD (Create, Read, Update, Delete) por colección.
- Attachments / Media (Archivos adjuntos)
- Almacenamiento binario con Blobs.
- Store Procedures, Triggers and Functions (Procedimientos almacenados, desencadenadores y funciones)
- Lógica de la aplicación que se ejecuta en la base de datos, escrito completamente en JavaScript.
- Users (Usuarios)
- Nombre de espacio lógico para alcances de permisos.
- Permissions (Permisos)
- Control de acceso a recursos específicos por medio de tokens de autorización.

El software puede estar escrito en cualquier lenguaje de programación

Azure Redis

El servicio de Redis ofrece un almacenamiento no relacional en memoria RAM, toda la información se va acumulando en memoria volátil y no en disco duro. Las maneras en que Redis almacena información son las siguientes:

- Strings
- Lists
- Hashes
- Redis Sets
- Sorted Sets
- Bitmaps
- Hyperloglogs
- Geospatial Indexes

Una de las principales limitaciones de Redis (en sí mismo) es la memoria, ya que las estructuras de datos no pueden ser mayores a esta, pero dado que Redis se encuentra en la infraestructura de Microsoft Azure, esa limitación se vuelve casi nula, los paquetes más básicos que podemos encontrar al momento de escribir este post, rondan en los 53 GB.
Ahora que nos queda más claro que es Redis, podríamos hacernos esta pregunta: ¿Por qué debería usar Redis a diferencia de una base de datos relacional?

Y bueno, la primera respuesta que podría ofrecer es: No debemos de usar Redis a menos que lo necesites, eso depende de la naturaleza de tu proyecto.

Aquí expongo algunas características de Redis:

- Redis se maneja en memoria RAM, por lo que el tiempo escritura y lectura de datos es superior que al hacerlo en un disco duro físico o virtual.
- Redis soporta replicación maestro-esclavo, por lo que los datos de un maestro pueden ser replicados a uno o varios esclavos, y estos a su vez también pueden actuar de maestros para otros esclavos.
- Los datos en Redis no siempre serán aleatorios, también pueden ser persistentes (pasarlos a disco duro).
- Estos son algunos ejemplos de proyectos que podrían usar Redis de manera ideal:
- Videojuegos.
- Proyectos que involucren mensajería instantánea.
- Software que constantemente invoque consultas transaccionales SQL.
- Aplicaciones de tiempo real.

Imagen 3.- Azure Redis.

Table Storage

El servicio de Table Storage ofrece un almacenamiento no relacional similar a una tabla de bases de datos relacionales. Sin embargo, los registros de estas tablas no tienen que tener el mismo número de columnas y tampoco necesitan de campos clave para hacer relaciones. Una sola tabla de tipo Table Storage dependiendo la arquitectura que se le dé, puede almacenar una o varias bases de datos simultáneamente.

Table Storage no es una base de datos relacional, pero es una alternativa al modelo de datos relacionales existente, estas son algunas de sus características:

- Las tablas son independientes.
- No existen llaves foráneas o joins.
- No existen índices personalizados.
- Una consulta de datos, se puede extender a más de un servidor, no limitándose a uno como normalmente existe en un modelo relacional.

Estos son los elementos que conforman una tabla:

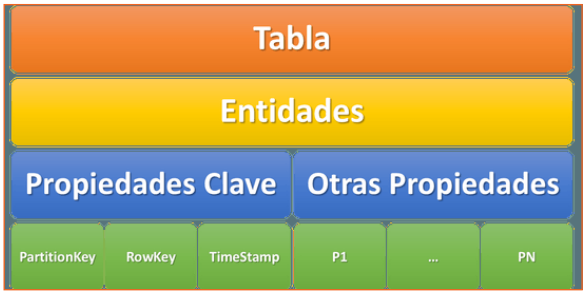


Imagen 4.- Elementos que conforman una tabla.

- Entidades: Puedes verlas como renglones o registros en una tabla.
- Propiedades: Puedes verlas como las columnas de un registro en una tabla. Siempre deben existir tres propiedades clave:
- PartitionKey: Tipo string, no mayor a 1 KB, comprende una ó varias entidades en una tabla.
- RowKey: Tipo string, no mayor a 1 KB, identifica de manera única una entidad dentro de la partición de la tabla.
- TimeStamp: Mantiene la hora en la que una entidad fue insertada o actualizada en la tabla.
- Particiones: Es una colección de entidades en una tabla que tienen el mismo PartitionKey.

El servicio de Table Storage ofrece un almacenamiento no relacional similar a una tabla

HDInsight

El servicio de HDInsight ofrece un almacenamiento no relacional dedicado al tratamiento de información a gran escala y no estructurada (que no tenga un tipo de dato definido), mejor conocido hoy en día como Big Data. Detrás de este servicio encontramos Hadoop.

Hadoop es un framework que permite el procesamiento de grandes volúmenes de datos a través de clusters. Su diseño permite empezar desde pocos hasta cientos de nodos. Hadoop es un sistema distribuido con arquitectura Maestro/Esclavo (Master/Slave), usando para almacenar información en su formato (HDFS) y algoritmos de tipo MapReduce. Clic aquí para conocer más de HDInsight: http://bit.ly/2aQ0TZ2

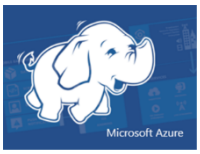


Imagen 4.- HDInsight.

Conclusiones

Como te podrás dar cuenta, los medios de almacenamiento son muy diversos. Dependiendo de las necesidades y objetivos de nuestros proyectos deberemos elegir la mejor alternativa.

Espero que este artículo te sea de utilidad y aprovecho para invitarte a seguirme por social media.

VÍCTOR MORENO
MVP Microsoft Azure
@vmorenoz | http://blogs.itpro.es/eduardocloud

52

Nosotros

Alberto Diaz

Alberto Díaz es SharePoint Team Lead en Encamina, liderando el desarrollo de software con tecnología Microsoft. Para la comunidad, ha fundado TenerifeDev (www.tenerifedev.com) con otros colaboradores, un grupo de usuarios de .NET en Tenerife, y coordinador de SUGES (Grupo de Usuarios de SharePoint de España, www.suges.es) y colaborador con otras comunidades de usuarios. Microsoft MVP de SharePoint Server desde el año 2011 y asiduo conferenciante en webcast y conferencias de tecnología de habla hispana.

Sitio Web: http://blogs.encamina.com/negocios-sharepoint/
Email: adiazcan@hotmail.com
Blogs: http://geeks.ms/blogs/adiazmartin
Twitter: @adiazcan

Fabián Imaz

Fabián Imaz, MVP de SharePoint Server trabaja en el mundo del desarrollo de software desde hace más de 10 años, teniendo la suerte de trabajar en distintas arquitecturas y tecnologías Microsoft. Pertenece a la firma Siderys, http://www.siderys.com empresa de desarrollo de Software especializada en SharePoint 2007/2010/2013 y en desarrollo de soluciones inteligentes. Desde los comienzos Fabián ha trabajado en distintitas comunidades donde organiza y promueve eventos locales para la difusión de tecnología dentro de los miembros de las mismas. Es director de la carrera SharePoint 2010 y SharePoint 2013 en Microsoft Virtual Academy, http://www.mslatam.com/latam/technet/mva2/ Home.aspx y cuenta con un sitio en CodePlex con varios desarrollos http://siderys.codeplex.com.

Sitio Web: http://www.siderysbsn.com
Email: fabiani@siderys.com.uy
Blogs: http://blog.siderys.com
Twitter: @fabianimaz

Gustavo Velez

Gustavo Velez es Ingeniero Mecánico y Electrónico; trabaja en el diseño e implementación de sistemas de IT basados en tecnologías de Microsoft, especialmente SharePoint, para Avanade (http://www. avanade.com), una compañía multinacional de IT. Propietario del sitio especializado en información sobre SharePoint en español http://www. gavd.net y autor de seis libros sobre SharePoint y sus tecnologías.

Sitio Web: http://www.gavd.net
Email: gustavo@gavd.net
Blogs: http://geeks.ms/blogs/gvelez/

Juan Carlos González Martín

Ingeniero de Telecomunicaciones por la Universidad de Valladolid y Diplomado en Ciencias Empresariales por la Universidad Oberta de Catalunya (UOC). Cuenta con más de 12 años de experiencia en tecnologías y plataformas de Microsoft diversas (SQL Server, Visual Studio, .NET Framework, etc.), aunque su trabajo diario gira en torno a las plataformas SharePoint & Office 365. Juan Carlos es MVP de Office Servers & Services y co-fundador del Grupo de Usuarios de SharePoint de España (SUGES, www.suges.es), del Grupo de Usuarios de Cloud Computing de España (CLOUDES) y de la Comunidad de Office 365. Hasta la fecha, ha publicado 9 libros sobre SharePoint & Office 365, así como varios artículos en castellano y en inglés sobre ambas plataformas.

Email: jcgonzalezmartin1978@hotmail.com
Blogs: http://geeks.ms/blogs/jcgonzalez &
http://jcgonzalezmartin.wordpress.com/

¿Desea colaborar con CompartiMOSS?

La subsistencia del magazine depende de los aportes en contenido de todos. Por ser una revista dedicada a información sobre tecnologías de Microsoft en español, todo el contenido deberá ser directamente relacionado con Microsoft y escrito en castellano. No hay limitaciones sobre el tipo de articulo o contenido, lo mismo que sobre el tipo de tecnología.
Si desea publicar algo, por favor, utilice uno de los siguientes formatos:

- Artículos de fondo: tratan sobre un tema en profundidad. Normalmente entre 2000 y 3000 palabras y alrededor de 4 o 5 figuras. El tema puede ser puramente técnico, tanto de programación como sobre infraestructura, o sobre implementación o utilización.

- Artículos cortos: Artículos cortos: Máximo 1000 palabras y 1 o 2 figuras. Describen rápidamente una aplicación especial de alguna tecnología de Microsoft, o explica algún punto poco conocido o tratado. Experiencias de aplicación en empresas o instituciones puede ser un tipo de artículo ideal en esta categoría.

- Ideas, tips y trucos: Algunos cientos de palabras máximo. Experiencias sobre la utilización de tecnologías de Microsoft, problemas encontrados y como solucionarlos, ideas y trucos de utilización, etc. Los formatos son para darle una idea sobre cómo organizar su información, y son una manera para que los editores le den forma al magazine, pero no son obligatorios. Los artículos deben ser enviados en formato Word (.doc o .docx) con el nombre del autor y del artículo.

Si desea escribir un artículo de fondo o corto, preferiblemente envíe una proposición antes de escribirlo, indicando el tema, aproximada longitud y número de figuras. De esta manera evitaremos temas repetidos y permitirá planear el contenido de una forma efectiva.

Envíe sus proposiciones, artículos, ideas y comentarios a la siguiente dirección:
revista@compartimoss.com adiazcan@hotmail.com
fabiani@siderys.com.uy jcgonzalezmartin1978@hotmail.com
gustavo@gavd.net


Comparti
MOSS