Nº43 marzo 2020

CompartiMOSS

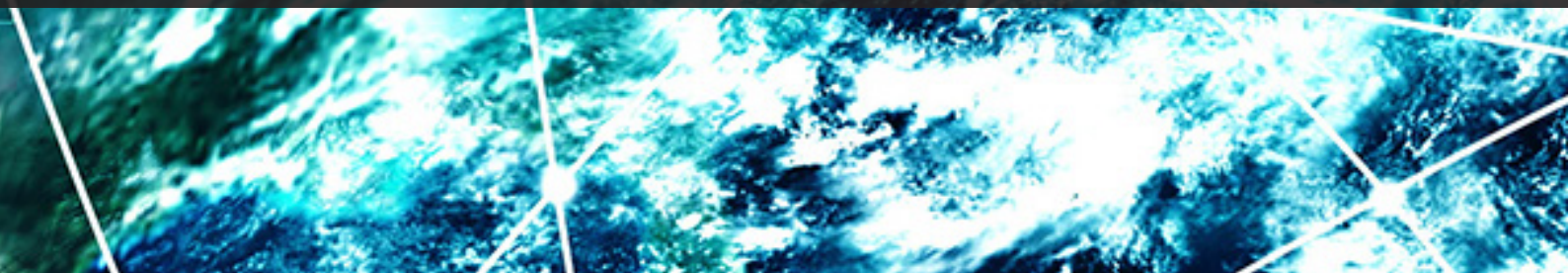
REVISTA ESPECIALIZADA EN TECNOLOGÍAS MICROSOFT

Entrevista Imanol Iza

Productivity Tips con Microsoft Teams: Canales Privados

Azure KeyVault + docker

Logic Apps y HTTP Actions, ¿Puedo hacer llamadas Async?



Contenido

Comparti MOSS

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

Un año más desde CompartiMOSS seguimos al pie del cañón comprometidos con nuestro objetivo de ser la revista de referencia en tecnologías y productos de Microsoft, aunque focalizados en Azure, Microsoft 365 y todas las plataformas asociadas (Microsoft Teams, Power Platform o SharePoint Online). En este nuevo año, en el que parece que se avecina una nueva crisis económica, Microsoft como el resto de los grandes jugadores del mercado del Software se enfocará cada vez más en el desarrollo de servicios y soluciones de Inteligencia Artificial disponibles y asentados sobre sus plataformas Cloud: Azure, Microsoft 365 y Dynamics 365.

Como siempre, este nuevo número de la revista ha sido posible gracias a la inestimable colaboración de nuestros autores cuyo apoyo y dedicación número tras número resulta fundamental. Esperamos que los artículos incluidos en el número 43 sean de vuestro agrado y desde ya nos ponemos a trabajar en el próximo número que verá la luz en junio de 2020.

El Equipo Editorial de CompartiMOSS

04

Productivity Tips con Microsoft Teams: Canales Privados

Los canales privados en Microsoft Teams están pensados para facilitar escenarios exclusivos de colaboración dentro de un Team a un subconjunto de sus integrantes. La reserva de un canal privado es tal que ni siquiera los propietarios de un Team que no formen parte del canal pueden acceder a sus conversaciones y archivos. En este artículo vamos a hacer un repaso a las características de los canales privados en cuanto a cómo se crean, como se proporciona acceso, como se administran y como se pueden extender por medio de aplicaciones.

"Los canales privados en Microsoft Teams están pensados para facilitar escenario privados de colaboración dentro de un Team a un subconjunto de sus integrantes"

Escenarios de uso de los Canales Privados

Antes de meternos en detalle con los Canales Privados de Teams, comentaremos en primer lugar algunos de los escenarios y casos de uso en los que tienen encaje:

- Un grupo de personas en un Team quieren un espacio particular para colaborar sin tener que recurrir a crear un nuevo Team.
- Un subconjunto de los integrantes de un Team quiere disponer de un entorno privado en el que hablar sobre información sensible (Presupuestos, Recursos, Estrategia, etc.).

Creación de Canales Privados

Por defecto, en un tenant de Office 365 la creación de Canales Privados está habilitada y tanto Propietarios (Owners) como Miembros (Members) de un Team pueden crear Canales Privados (Nota: El número de Canales Privados que se puede crear en un Team está limitado a 30). La creación de canales privados se puede restringir a dos niveles:

- A nivel global en el Teams Admin Center un Administrador Global o bien un Administrador de Teams puede deshabilitar la creación de Teams a nivel de política (Nota: Como alternativa al Teams Admin Center, se puede administrar la creación de Canales Privados a nivel de tenant con el cmdlet New-CsTeamsChannelsPolicy y el atributo AllowPrivateChannelCreation):

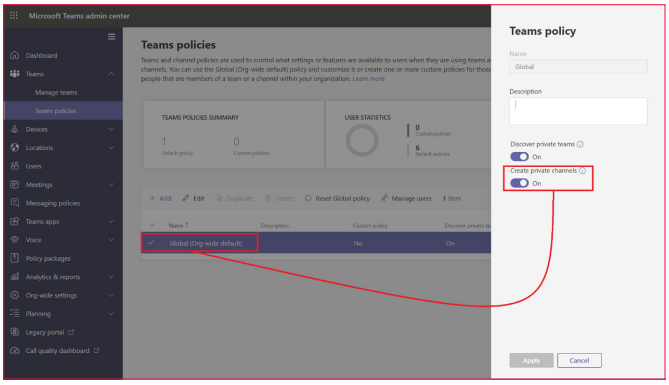


Imagen 1.- Deshabilitar la creación de Teams en el Teams Admin Center.

- A nivel de un Team concreto, un usuario de tipo propietario puede deshabilitar que los miembros pueden crear canales privados quitando los permisos correspondientes:

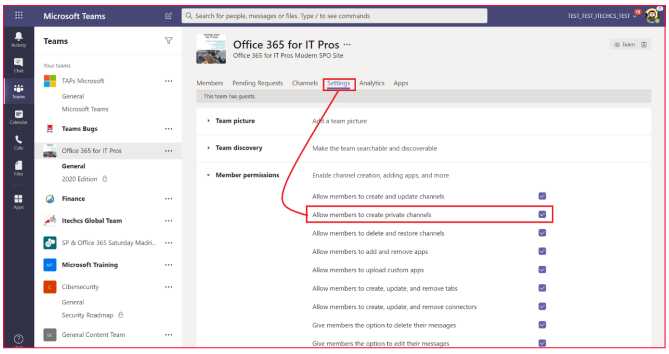


Imagen 2.- Permiso para crear canales privados para miembros de un Team.

Para crear un Canal Privado, un propietario o miembro del Team (*Nota: Los usuarios invitados no pueden crear canales privados*) únicamente tiene que seguir el siguiente proceso:

- A través de las opciones de menú a nivel de Team, hacer clic en la opción para crear un nuevo canal:

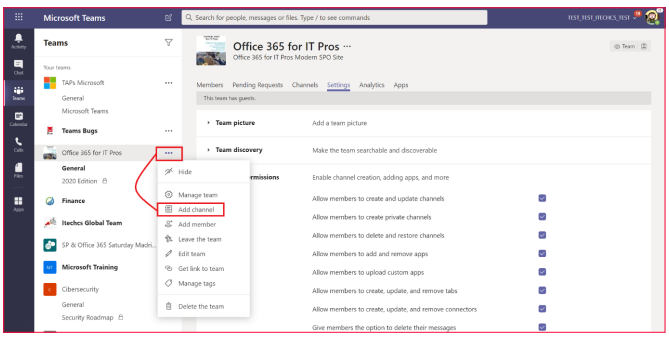


Imagen 3.- Opción para crear un nuevo canal.

- En la ventana de creación del canal, se tiene que indicar un nombre, opcionalmente una descripción y a continuación el tipo de canal a crear. En este caso, elegiremos privado como tipo de canal a crear y hacemos clic en Siguiente (Next). Hay que tener en cuenta que una vez una Canal se crea como Privado, no es posible posteriormente convertirlo en un Canal estándar.

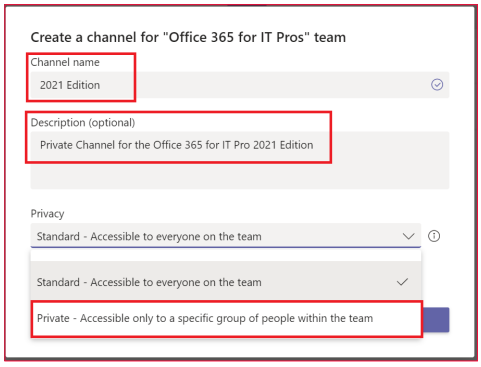


Imagen 4.- Creación de un canal privado.

- A continuación, de forma opcional podemos indicar los integrantes del Canal Privado que pueden tener tres roles al igual que sucede en un Team: Propietarios (Owners), Miembros (Members) o Invitados (Guests). Cualquier integrante que añadamos tiene que ser también integrante del Team, es decir, no es posible añadir otros usuarios que no formen parte del Team.

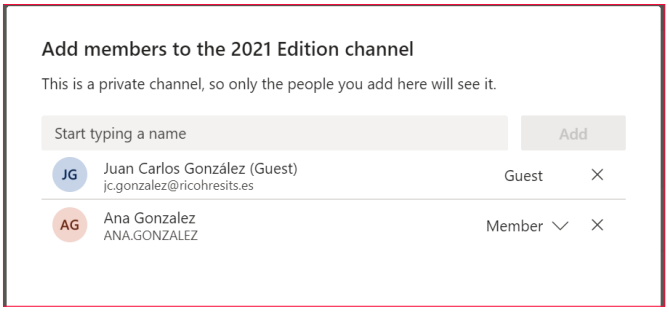


Imagen 5.- Añadiendo integrantes al Canal Privado.

Nota: El número máximo de integrantes de un Canal Privado es de 250.

- Una vez el Canal Privado se crea, aparece como un Canal más en el Team, pero identificado con un candado.

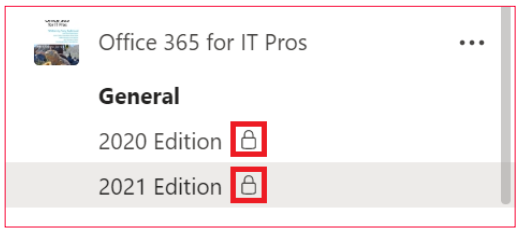


Imagen 6.- Canales Privados en un Team

"A nivel de un Team concreto, un usuario de tipo propietario puede deshabilitar que los miembros pueden crear canales privados quitando los permisos correspondientes"

Administración de un Canal Privado

Los Canales Privados, como los estándares, cuenta con sus propias funciones de administración. Para acceder a estas funciones:

- Hacemos clic en los "…" al lado del nombre del Canal Privado o bien hacemos clic con el botón derecho del ratón.
- En el menú contextual que se abre, hacemos clic sobre "Administrar canal" (Manage channel):

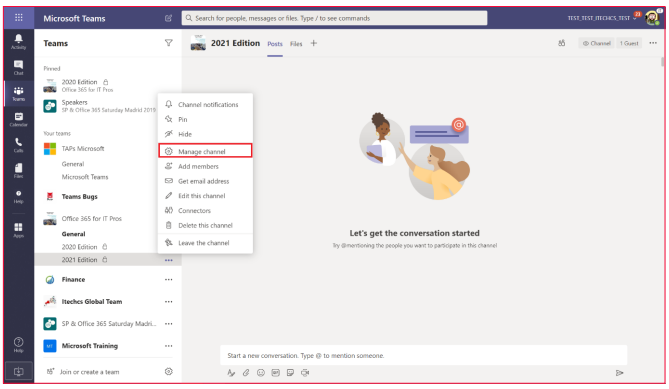


Imagen 7.- Acceso a la administración de un Canal Privado.

- La sección de Integrantes (Members) muestra todos los usuarios del Canal Privado y permite añadir nuevos usuarios.

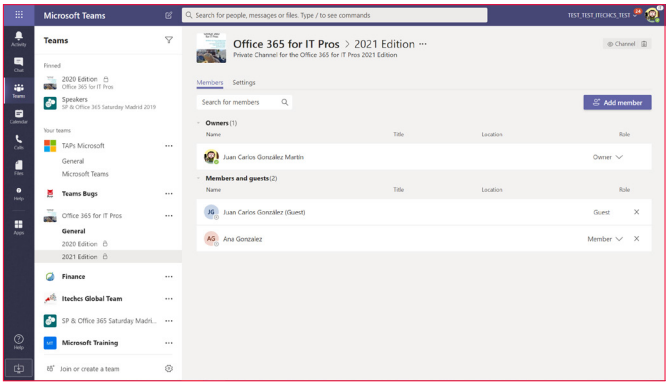


Imagen 8.- Sección de Integrantes en la administración del Canal Privado.

- La sección "Settings" muestra las configuraciones relativas al Canal Privado en cuanto a permisos de integrantes, si se permite mencionar al canal con "@" y configurar el material divertido en las conversaciones que tengan lugar en el Canal Privado.

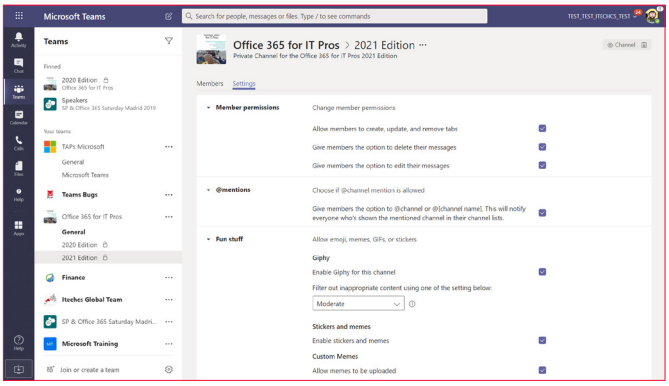


Imagen 9.- Sección de "Settings" en la administración de un Canal Privado.

Aplicaciones en Canales Privados

En los Canales Privados, al igual que en los estándares, se pueden añadir Aplicaciones de Teams. En concreto:

- Se pueden añadir Conectores y Pestañas (Tabs).
- No es posible por el momento añadir Bots o Extensiones de Mensajes.

Las Aplicaciones que se pueden añadir en un Canal Privado se tienen que haber instalado primero en el Team al que pertenece. Por ejemplo, para añadir una nueva Pestaña en un canal privado hacemos clic en la acción "+" de forma que se muestra la ventana con las Tabs que se pueden instalar en el Canal:

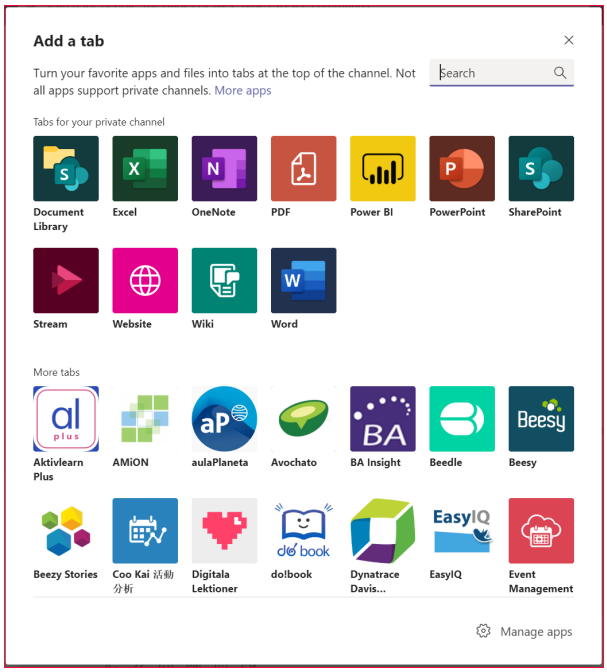


Imagen 10.- Aplicaciones que se pueden instalar en un Canal Privado.

Nota: No todos los tipos de Tabs que se pueden añadir a un Canal Estándar se pueden añadir a un Canal Privado. Por ejemplo, por el momento no es posible añadir un Plan de Planner como pestaña de un Canal Privado.

Gestión y Almacenamiento de Archivos en Canales Privados

Los Canales Privados, como los estándares, permiten que se puedan almacenar todo tipo de documentos, estructurarlos en carpetas y en general sacar partido a la integración de SharePoint Online con Microsoft Teams.

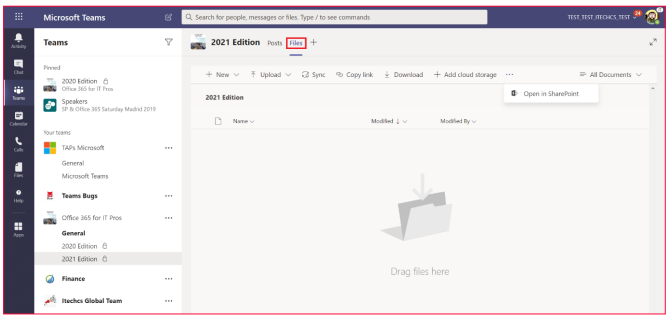


Imagen 11.- Sección de Archivos en un Canal Privado.

"Los Canales Privados, como los estándares, permiten que se puedan almacenar todo tipo de documentos, estructurarlos en carpetas y en general sacar partido a la integración de SharePoint Online con Microsoft Teams"

Ahora bien, los archivos en un Canal Privado se gestionan de forma diferente a los archivos de un Canal Estándar ya que cada Canal Privado cuenta con su propio Sitio de SharePoint que tiene las siguientes características:

- La URL del Sitio es de la forma /sites/<Nombre del Team>-<Nombre del Canal Privado>.
- Los Sitios de Canales Privados se crea con una plantilla de sitio específica: TeamChannel#0.
- Los Sitios de Canales Privados no se muestran en el Modern Admin Center de SharePoint Online.
- A los Sitios de Canales Privados se les pueden añadir nuevas Listas y Bibliotecas, pero no se soportan en cambio páginas modernas.

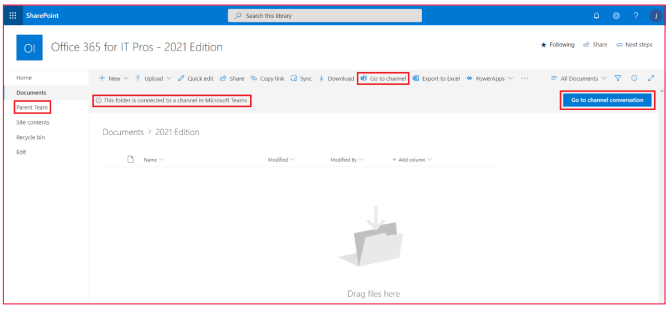


Imagen 12.- Sitio de un Canal Privado.

- Desde la Biblioteca del Canal Privado se puede acceder a las conversaciones del Canal y desde la navegación del Sitio se puede acceder al Sitio del Team al que pertenece el Canal Privado.

Conclusiones

Los Canales Privados en Microsoft Teams permite que parte de los integrantes de un Team puedan disponer de un espacio privado de colaboración y en el que trabajar con información confidencial sin riesgo a que otros usuarios del Team padre (incluyendo Propietarios) puedan acceder y visualizar su actividad. En un Team se pueden crear hasta 30 Canales Privados y en cada Canal Privado podemos tener hasta 250 integrantes. Los Canales Privados se pueden extender con Aplicaciones y cada vez que se crea un Canal Privado, se crea una Colección de Sitios para almacenamiento y gestión de la documentación del canal.

JUAN CARLOS GONZÁLEZ MARTÍN
Office Apps and Services MVP | Office 365 SME
@jcgm1978

07

.NET Core Más allá de la seguridad, autenticación y autorización en una API

Con la llegada del "nuevo" framework de .NET Core, la forma en la que se desarrollaban las aplicaciones dio un vuelco totalmente. Un framework monolítico que solo se podía ejecutar en entornos Windows, por un nuevo framework modular, multiplataforma y orientado al Cloud. A lo largo de este articulo vamos a ver que opciones nos permite para configurar y administrar la seguridad en nuestros desarrollos.

Antes de empezar a entrar en materia con el detalle, hay que tener claro los conceptos autenticación y autorización, aunque puedan ser similares no lo son, sino que se complementan el uno al otro. La autenticación es un proceso en el que un usuario proporciona credenciales que después se comparan con las almacenadas en un sistema operativo, base de datos, aplicación o recurso. Si coinciden, los usuarios se autentican correctamente y, después, pueden realizar las acciones para las que están autorizados durante un proceso de autorización. La autorización se refiere al proceso que determina las acciones que un usuario puede realizar.

Autenticación

Una vez tenemos claro que es cada cosa vamos a empezar con la autenticación. ¿Como vamos a autenticar nuestra API? En primer lugar, debemos tener claro como nos vamos a autenticar. El método más común que se está utilizando ahora mismo es utilizando JWT.

¿Qué es JWT? JSON Web Token es un estándar abierto basado en JSON propuesto por IETF para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o claims. ¿Quién nos puede generar ese token? Cualquier proveedor de identidad como pueda ser Azure, Amazon, Google, Twitter o LinkedIn (por citar algunos) disponen de un servidor de token en el que podemos autenticar a un usuario en nuestra aplicación. También es posible la implementación de un servidor que nos devuelve estos tokens en base a la lógica de negocio que nosotros consideremos oportuna. La ventaja de introducir algún proveedor de los citados anteriormente es que casi todo el mundo ya dispone de un usuario en dichas plataformas y por lo tanto muchos usuarios no les haría falta registrarse en ningún otro sistema, ni tener que recordar otra clave más.

Talk is cheap, show me the code … Toda esta parte teórica está muy bien, pero vemos ver como se le añade esta funcionalidad a nuestra aplicación de .NET Core. Cuando creamos una solución en .NET Core tiene una clase de arranque de StartUp.cs que está compuesta de dos métodos ConfigureServices y Configure. En el primer método se tienen que configurar añadir la configuración de todos los servicios/middlewares que se van a utilizar y en el segundo se le añade que lo utilice. Para el tema de la autenticación, por ejemplo, para una autenticación contra el Azure Active Directory, dentro del ConfigureServices tenemos que añadir el siguiente código

```
0 references | Adrian Diaz Cervera, 36 days ago | 1 author, 3 changes
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(sharedOptions =>
    {
        sharedOptions.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    })
        .AddAzureAdBearer(options => Configuration.Bind("AzureAd", options));
```

El método AddAzureAdBearer se inyecta como un paquete de Nuget incluido dentro de ASP.NET Core pero también en versiones anteriores se añadía como un middeware a la aplicación. El código que tiene es relativamente sencillo y lo que debe se encarga es de mapear las propiedades del servidor de seguridad con los datos que llevan en el token de cada petición. El código es el siguiente:

```
0 references | Adrian Diaz Cervera, 38 days ago | 1 author, 1 change
public static class AzureAdServiceCollectionExtensions
{
    0 references | Adrian Diaz Cervera, 38 days ago | 1 author, 1 change
    public static AuthenticationBuilder AddAzureAdBearer(this Authe
        => builder.AddAzureAdBearer(_ => { });
    2 references | Adrian Diaz Cervera, 38 days ago | 1 author, 1 change
    public static AuthenticationBuilder AddAzureAdBearer(this Authe
```

Para indicarle los valores por los que tiene que autenticar nuestra aplicación, en el servicio que utilicemos (app.setings, KeyVault) tendremos que introducir los siguientes valores, que se obtienen al dar de alta la aplicación en el Azure Active Directory:

```
"AzureAd": {
  "ClientId": "",
  "Domain": "",
  "Instance": "https://login.microsoftonline.com/",
  "TenantId": "",
  "CallbackPath": "/signin-oidc",
  "ClientSecret": "",
  "AppIDURL": "",
  "ConfigView": ""
}
```

Una vez tenemos ya se ha añadido el método en el que indicamos que podemos utilizar este tipo de autenticación, el siguiente paso es en método Configure indicarle que nuestra aplicación requiere autenticación. Para ello hay

que añadir el siguiente código:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    this.AutofacContainer = app.ApplicationServices.GetAutofacRoot();
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseHttpsRedirection();

    app.UseRouting();


    app.UseAuthentication();
    app.UseAuthorization();
```

Nota: Es bastante importante el orden en el que se añade cualquiera de estos elementos, es decir, si indicamos que usemos Authorization antes que Autenticación nuestra API no tendrá ni una cosa ni la otra. Resumiendo, el orden en el que añadimos dichos elementos el pipeline tiene su importancia.

Autorización

Esta es la parte sencilla, ahora bien, que tipos de autorización nos proporciona .NET Core. Para indicar que uno de los controladores de la API que estamos desarrollando nos bastaría con poner en el decorador del controlador el Atributo "Authorize". Algo como el siguiente código:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
[ApiVersionNeutral]
[Route("api/[controller]")]

[Authorize]
0 references | Adrian Diaz Cervera, 36 days ago | 1 author, 2 changes
public class SharedController : Controller
{
    [HttpGet(Name ="Version")]
    0 references | Adrian Diaz Cervera, 38 days ago | 1 author, 1 change
    public string Get()=> (HttpContext.GetRequestedApiVersion() != null) ?
                        HttpContext.GetRequestedApiVersion().ToString() :
                        "Not version in Request";


}
```

"La autenticación es un proceso en el que un usuario proporciona credenciales que después se comparan con las almacenadas en un sistema operativo, base de datos, aplicación o recurso"

Autorización basada en roles

Hasta este punto nada del otro jueves, pero está claro que en otras versiones de ASP.NET nos creábamos una autenticación basada en Roles. El típico Rol Provider en que se consultaba en una base de datos que rol tenía el usuario que se había autenticado y se encargaba de indicar si podía acceder a dicho método o no. Los tiempos cambian, ahora tenemos un Claim y en la misma se puede indicar que Rol/es tiene dicho usuario y con el mismo token podemos empezar a construir nuestra lógica. Como se puede comprobar el tema de rendimiento entre ambas opciones es bestial.

Ejemplo: Poner el caso que tenemos una aplicación de los Vengadores en el que a cada usuario le asignamos un Rol (el avenger al que pertenece), imaginaros que tenemos unos métodos en el que tenemos que invocar el poder de cada vengador. ¿Cómo lo haríamos?

```
[HttpGet]
[Route("featurehulk")]
[Authorize(Roles = "Hulk")]
0 references | Adrian Diaz Cervera, 36 days ago | 1 author, 1 change
public IEnumerable<string> GetHulk()
{
    return new string[] { "fuerte", "leal" };
}

[HttpGet]
[Route("featureviudanegra")]
[Authorize(Roles = "ViudaNeegra")]
0 references | Adrian Diaz Cervera, 36 days ago | 1 author, 1 change
public IEnumerable<string> GetViudaNegra()
{
    return new string[] { "habilidosa", "silenciosa", "inteligente" };
}
```

Dentro del atributo Authorize indicamos el Rol o Roles que tienen autorización para llamar a cada método. En este caso el primer método solo lo podrán invocar los usuarios que pertenezcan al Rol Hulk, mientras que en el segundo solamente lo pueden invocar los usuarios que pertenezcan al Rol Viuda Negra.

Si queréis saber cómo añadir estos roles en la aplicación del Azure Active Directory podéis seguir este enlace en el que se detallan todos los pasos para hacerlo https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-enterprise-app-role-management

El Rol del usuario es algo que también podemos acceder dentro del controlador. Igual necesitamos que el desarrollo funcione de una forma diferente dependiendo del rol que acceda, por ejemplo, no es lo mismo que cuando acceder un Administrador que cuando acceder un responsable de una delegación u otro usuario con menos privilegios. Pues esta opción también está permitida porque se dispone del método IsInRole dentro de la Claim que se envía. Este ejemplo lo podríamos hacer tal que así:

```
[HttpGet]
[Authorize]
0 references | Adrian Diaz Cervera, 36 days ago | 1 author, 1 change
public IEnumerable<string> Get()
{
    if (User.IsInRole("Hulk"))
    {
        return new string[] { "fuerte", "leal" };
    }
    else
    {
        return new string[] { "difernte" };
    }
}
```

Autorización basada en directivas

En la gran mayoría de las ocasiones es posible que con los dos tipos de autorizaciones tengamos más que suficientes, pero vamos a pensar en otros casos posibles. Imaginemos que tenemos una API en el que los métodos que tenemos diversos tipos de licenciamiento en el que dependiendo de si el usuario ha adquirido un tipo de licencia o no puede invocar a determinados métodos o no. Está claro que esta opción no es recomendable tenerla en una Claim, sino que esta consulta implica una lógica o unas determinadas acciones propias de la lógica de negocio.

Manos a la obra, una directiva Una directiva de autorización se compone de uno o varios requisitos. En primer lugar, nos vamos a crear un Requerimiento. En nuestro caso es si se le aplica licencia o no. Para ello tendremos el siguiente método:

```
using Microsoft.AspNetCore.Authorization;

4 references | Adrian Diaz Cervera, 35 days ago | 1 author, 1 change
public class LicenceRequirement : IAuthorizationRequirement
{
    1 reference | Adrian Diaz Cervera, 35 days ago | 1 author, 1 change
    public bool Licence { get; }

    1 reference | Adrian Diaz Cervera, 35 days ago | 1 author, 1 change
    public LicenceRequirement(bool licence )
    {
        Licence = licence;
    }
}
```

Una vez tenemos el Requerimiento creado, el siguiente paso es implementar el Handler que es el encargado de hacer cumplir dicho requerimiento. Para ello se tendrá que implementar una clase que implemente la interfaz IAuthorizationHandler

```
0 references | 0 changes | 0 authors, 0 changes
public class LicenceHandler : AuthorizationHandler<LicenceRequirement>
{
    0 references | 0 changes | 0 authors, 0 changes
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                   LicenceRequirement requirement)
    {

        context.Succeed(requirement);

        return Task.CompletedTask;
    }
}
```

En el método HandleRequirement deberemos de implementar la lógica de negocio que queramos aplicar a dicho método. En este caso todo lo que entra en el requerimiento es aprobado ya que no se ha añadido ninguna condición.

> ***"El método AddAzureAdBearer se inyecta como un paquete de Nuget incluido dentro de ASP. NET Core pero también en versiones anteriores se añadía como un middeware a la aplicación"***

Una vez tenemos implementado el método el siguiente paso es añadirlo en el arranque de la aplicación para que tengamos dicha Política y utilizarla cuando dentro del controlador o método que consideremos oportuno. Para ello en el ConfigureService tendremos que añadir este código:

```
// Add all of your handlers to DI.
services.AddSingleton<IAuthorizationHandler, LicenceHandler>();
services.AddAuthorization(options =>
{
    options.AddPolicy("Licence", policy =>
        policy.Requirements.Add(new LicenceRequirement(true)));
});
```

Ahora casi como en el resto de los atributos en el controlador, ponemos Policy junto con el nombre de la política en nuestro caso Licence. Y en caso de que el usuario tenga licencia podrá lanzar dicho método.

```
[HttpGet("{id}", Name = "Get")]
[Authorize(Policy = "Licence")]
0 references | Adrian Diaz Cervera, 35 days ago | 1 author, 2 changes
public string Get(int id)
{
    return "value";
}
```

Resumen

La seguridad es algo lo suficientemente importante como para tenerlo en cuanta desde el minuto uno que empezamos a realizar un desarrollo. ASP Net Core es un framework lo suficientemente bien pensado, y con una comunidad enorme que lo sustenta como para tener multitud de opciones para poder añadir seguridad en base a los requerimientos que se necesiten.

La construcción de una API es algo lo suficientemente importante como para tener claro cada uno de los elementos de los que se compone: Seguridad, Nomenclatura, versionado, testing … a lo largo de siguientes artículos iré desgranando algunos de dichos aspectos.

El código fuente mostrado en el ejemplo está disponible en este repositorio de GitHub https://github.com/AdrianDiaz81/NetCorenf-BCN-2020

Happy coding!!

ADRIÁN DIAZ CERVERA
Architect Software Lead at Encamina
MVP Office Development
http://blogs.encamina.com/desarrollandosobresharepoint
adiaz@encamina.com @AdrianDiaz81

10

Entrevista Imanol Iza

Me llamo Imanol Iza, nací en San Sebastián en el año 82. Aunque soy vasco de raíces y echo mucho de menos mi tierra, mi hogar actual es Madrid desde hace 13 años.

Precisamente esos son los años que llevo trabajando en tecnologías Microsoft, siempre en temas relacionados con SharePoint desde sus orígenes, .Net, C# y a día de hoy Office 365 y Azure.

Actualmente trabajo en NECSIA como Director de Operaciones encargándome del buen funcionamiento de los proyectos y equipos del área de Transformación Digital.

Desde hace algo mas de un año soy MVP de Microsoft y colaboro en las comunidades aportando mi granito de arena. También me encargo de coordinar junto con mis compañeros el grupo de usuarios de SharePoint de Madrid "MadPoint" y algunos eventos importantes como el "SharePoint Saturday Madrid" o el "O365 Developer Bootcamp"

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

Desde bien pequeño siempre tuve claro que quería dedicarme a la tecnología. Siempre he sido una persona muy inquieta y curiosa, recuerdo aquellas tardes en el ático de un amigo, montando nuestras primeras placas electrónicas.

Quizás eso fue lo que me impulsó a hacer Ingeniería técnica Electrónica. Aunque la vida da muchas vueltas y finalmente dejé la carrera para realizar un grado superior de Informatica. ¡La mejor decisión que pude tomar! ¡Era lo que verdaderamente me gustaba!

Tras unas prácticas, decidí marchar un año a Madrid para coger experiencia. Tuve la gran suerte de caer en Medianet Software (actualmente Sngular), donde conocí grandes compañeros que me ayudaron un montón. Aprendí muchísimas cosas y empecé a introducirme en tecnologías Microsoft.

Ese mismo verano conocí a una Vicalvareña de ojos azules que me enamoró e hizo que echara raíces en Madrid hasta el día de hoy. El resto de los años hasta la actualidad ha sido un viaje apasionante lleno de experiencias en diferentes compañías y compañeros de viaje que me han ayudado a ser la persona y el profesional que soy hoy en día.

¿Cuáles son tus principales actividades tecnológicas hoy en día?

Mi día a día es muy variado, me encargo de liderar las operaciones, coordinando y gestionando el trabajo del equipo. Llevo el control de las cuentas y las oportunidades participando en la fase de preventa técnica, diseñando soluciones sobre Office 365 y Azure.

Por otro lado, trato de contribuir en las comunidades técnicas siempre que puedo, aportando mi granito de arena y compartiendo mis conocimientos con los demás.

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

Mi prioridad es mi familia. Tengo una hija de un año y disfruto mucho viéndola crecer. Nos encanta pasar tiempo juntos toda la familia, bien sea dando un paseo o viendo una serie de Netflix con manta y palomitas.

¿Cuáles son tus hobbies?

Tengo dos hobbies principales. La natación y la fotografía. Desde muy pequeño he sido siempre nadador federado,

y es una de mis grandes pasiones. Me encanta hacer travesías de aguas abiertas en el mar en verano. Otra de mis grandes pasiones es la fotografía. Me encanta plasmar en imágenes momentos únicos y poder compartirlos con los demás.

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

Estamos viviendo un momento fantástico tecnológicamente. Los avances que se han realizado son asombrosos y los próximos años serán muy prometedores. Veremos como la IA y los servicios contextuales nos facilitarán las tareas cotidianas y se integrarán en nuestras vidas de forma natural. Los servicios Cloud serán sin duda la base de todas las tecnologías y la eclosión de los dispositivos IoT en nuestras vidas marcará la diferencia en un futuro no muy lejano.

IMANOL IZA
Cloud Operations Manager, NECSIA | Microsoft MVP
@imaiza
Imanol.iza@gmail.com
www.imanoliza.com

12

Clasificación de textos con AI Builder y Power Automate

Introducción

En AI Builder existen cuatro tipos de modelos que podemos elegir según nuestras necesidades:

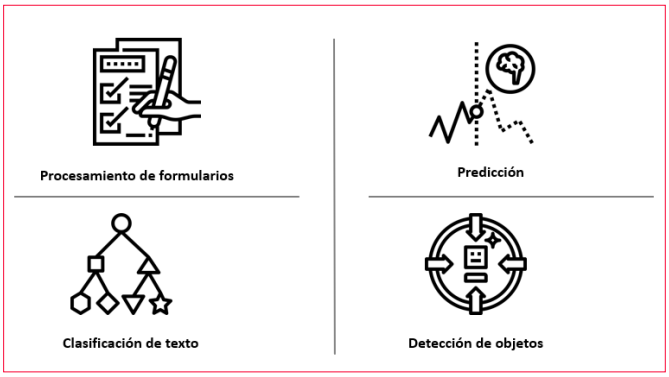


Imagen 1.- Tipos de modelos personalizados de IA.

- Procesamiento de formularios: Modelo capaz de extraer pares clave-valor de un documento. Por ejemplo, para un modelo de factura de un proveedor específico, podríamos extraer la fecha, los conceptos y/o el importe total.
- Predicción: Modelo capaz de detectar predicciones en base a patrones históricos de resultados. Este modelo puede ser especialmente útil en los casos que queramos obtener respuestas de tipo sí/no o aprobar/suspender. Un ejemplo de este modelo sería el de analizar si una persona asistirá o no a un evento de comunidad.
- Clasificación de texto: Con este modelo podemos clasificar cualquier tipo de contenido textual (correos electrónicos, documentos, posts en redes sociales, etc.), de forma que podemos asignarlo a un conjunto de etiquetas o metadatos en base un aprendizaje previo. Esto nos podría servir, por ejemplo, para enrutar una petición o queja realizada en Twitter hacia un departamento específico de nuestra compañía a partir del texto del mismo.
- Detección de objetos: Este modelo de IA es capaz de detectar los objetos que aparecen en una imagen en base a un aprendizaje previo. Esto nos podría ser útil, por ejemplo, para fotografiar una máquina que se encuentra en una fábrica, reconocerla y obtener sus especificaciones técnicas que se encuentran almacenadas en un documento en SharePoint.

Cabe destacar que, exceptuando en el caso del modelo de Predicción, actualmente estos tipos de modelos de IA se encuentran en versión preliminar.

Caso práctico: Uso del modelo de Clasificación de Texto

La mejor manera de ver cómo nos pueden ayudar estos tipos de modelos de IA es el de verlos en acción con un caso práctico. Imaginemos que queremos mejorar la atención a los pacientes en un hospital a partir de sus valoraciones sobre la atención recibida. Para ello, podríamos seguir el siguiente esquema:

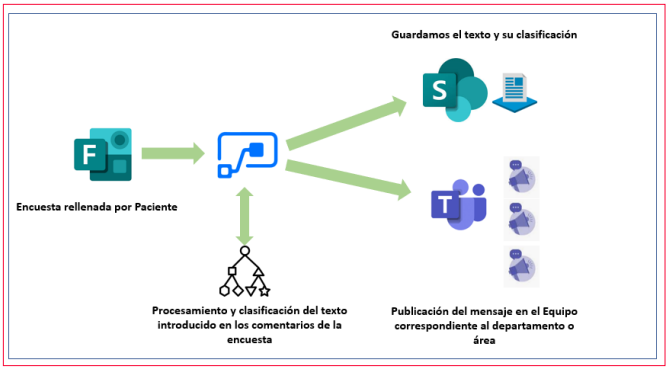


Imagen 2.- Flujo para la clasificación de valoraciones de pacientes y publicación en el Equipo correspondiente.

El objetivo es que, a partir de los comentarios introducidos en la encuesta, la misma llegue al departamento o equipo al que le puede interesar para llevar a cabo las acciones que considere necesarias.

> ***"AI Builder nos permite integrar de forma clara y sencilla soluciones de inteligencia artificial a nuestras aplicaciones y/o automatizaciones de procesos"***

Creación del modelo

Antes de crear nuestro flujo en Power Automate, deberemos crear y entrenar un modelo de IA de clasificación de texto. Para este caso práctico hemos utilizado los datos de ejemplo que nos proporciona Microsoft para hacer algunos laboratorios en AI Builder[1]. Concretamente vamos a usar el fichero pai_healthcare_feedbacks.csv, que contiene textos y etiquetas con las deberemos entrenar al modelo:

1 https://github.com/microsoft/PowerApps-Samples/tree/master/ai-builder

pai_name	pai_tags	pai_text
573	Care	During my stay I was completely ignored. The staff failed to pick up on me aspirating and having a UTI, I also had pneumonia.
572	Care,Facilities	Despite in a room with many physios treating many patients I was made to feel that I had personal, sensitive attention.
571	Care	I WAS QUITE ILL AFTER MY OPERATION AND THE CARE I RECEIVED WAS VERY GOOD
570		THE STAFF: DOCTORS, NURSES AND CLEANERS WERE ALL VERY PROFESSIONAL AND TRIED TO HELP ALL THEY COULD EVEN WHEN VE
569	Care	the doctors/nurses and staff on SAU & ward 1 where I was treated with respect and dignity by all.
568	Facilities	The ward seemed clean and I regularly saw cleaners about working diligently.
567	Care	All the Staff were exceptional polite and friendly
566	Care	The service I received was first class. The physiotherapist was polite, good humoured, helpful and kept me informed of what she was
565	Care	very proffesional very gentle given answers to questions
564	Care	there were a couple of outstanding individuals amongst the nursing team and the medical staff treated me with courtesy, respect ar
563	Care	Nurses were very caring and professional despite the fact they were very busy.
562	Care	doctors working here agreed that the cleniness, respect and treatment of most patients was barbaric.
561		Friendly helpful staff.
560	Care	Once again, I'm hugely impressed by the level of care I received, and wouls just like to say thanks :)
559		The staff on the ward (from the doctors and nurses, to the assistants, and the cleaning staff) are absolutely amazing - friendly, carin
558	Care	my husband was treated with great care at the emgermency and then transferred very quickly to observation were he stayed for a v
556	Care	I was amazed at the level of service provided, the timeliness of my journey through a well orchestrated system and the attention, cc
555		friendlyness of all staff
554	Care	I was seen fairly quickly, treated respectfully and my condition and next steps were well explained.
553	Check-in	Receptionist was friendly and helpful and arranged fracture clinic appointment promptly.

Figura 3.- Fichero de aprendizaje para el modelo de clasificación de texto.

Como podemos ver, disponemos de una columna (pai_tags) con las etiquetas que queremos asignar sobre el texto (pai_text). Cabe tener en cuenta que, aunque el texto está en inglés, actualmente se soportan otros idiomas, como el español[2]. Cabe mencionar algunos aspectos importantes sobre el fichero en el que se basará el aprendizaje:

- Número de entradas: Actualmente, para poder entrenar al modelo necesitamos 100 o más entradas de texto por etiqueta.
- Idioma: Todas las entradas de texto deberían estar en el mismo idioma.
- Distribución de valores: Debería haber el mismo número de entradas para cada una de las etiquetas. Obviamente el aprendizaje es mucho más difícil si hay muchas etiquetas de un tipo y pocas de otro y, en definitiva, los resultados obtenidos tendrán una baja fiabilidad.

Para crear el modelo, desde nuestro entorno[3] de Power Platform, seleccionaremos el tipo de modelo que queremos crear, y en el que deberemos indicar un nombre para el mismo:

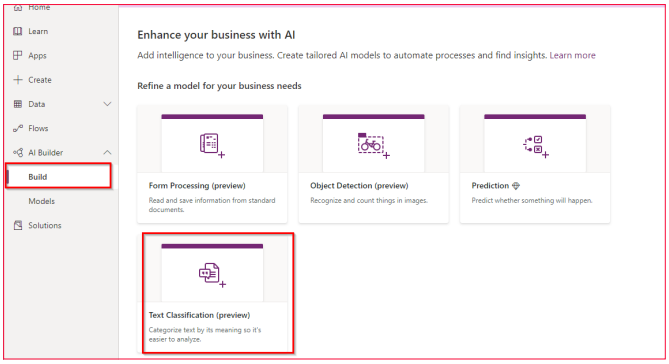


Imagen 4 .- Creación del modelo de clasificación de texto.

Seguidamente el sistema nos preguntará en qué entidad del Common Data Service y columna se encuentra el texto a analizar, así como el campo que contiene la etiqueta (nos preguntará por el carácter que separa a cada una de ellas). Es decir, antes de poder crear el modelo deberemos importar[4] nuestros datos (el fichero Excel mostrado previamente) en una nueva entidad del Common Data Service.

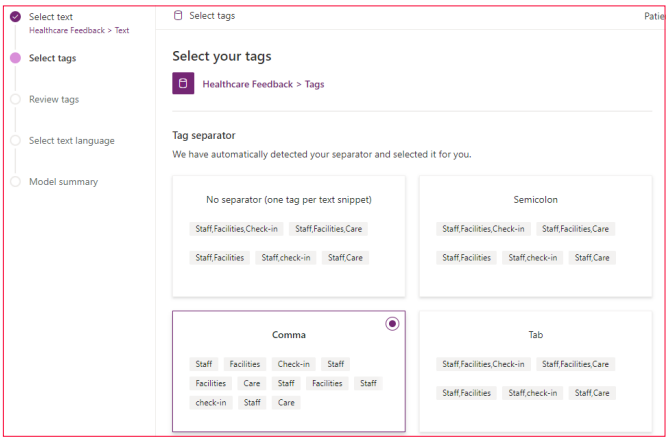


Imagen 5 .- Selección del campo y separador de etiquetas.

Finalmente seleccionaremos el idioma en el que están los textos y procederemos a entrenar al modelo, proceso que puede tardar más o menos en función del volumen de los datos de aprendizaje (en nuestro caso, éste finalizó en 3 minutos aproximadamente). Cuando el modelo ha finalizado el entrenamiento, éste se puede probar antes de publicarlo, tal y como se muestra en la siguiente figura, donde hemos introducido nosotros el texto:

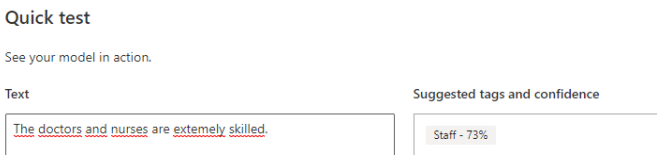


Imagen 6 .- Prueba del modelo de clasificación de texto creado.

Podemos comprobar como el resultado de analizar la frase "The doctors and nurses are extremely skilled" nos devuelve por resultado la etiqueta "Staff", con una confiabilidad del 73% en el resultado obtenido (cuanto mejor esté entrenado el modelo, mayor confiabilidad obtendremos en el resultado).

Uso del modelo en Power Automate

Como indicamos previamente, implementaremos un flujo que se ejecutará cuando un paciente rellene la encuesta que se le enviará cuando haya finalizado su atención o estancia en el centro, que podría tener el siguiente aspecto:

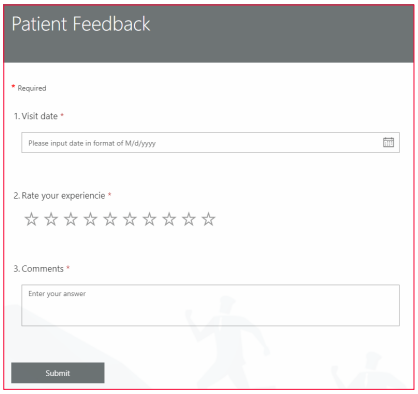


Imagen 7.- Encuesta de satisfacción para el paciente.

Un elemento muy importante a tener en cuenta: Para poder utilizar los modelos de IA desde Power Automate, de-

2 https://docs.microsoft.com/en-us/ai-builder/before-you-build-text-classification-model

3 https://docs.microsoft.com/es-es/power-platform/admin/environments-overview

4 https://docs.microsoft.com/es-es/powerapps/developer/common-data-service/import-data

beremos crear nuestros flujos desde una nueva solución[5]. El aspecto del flujo que crearemos es el siguiente:

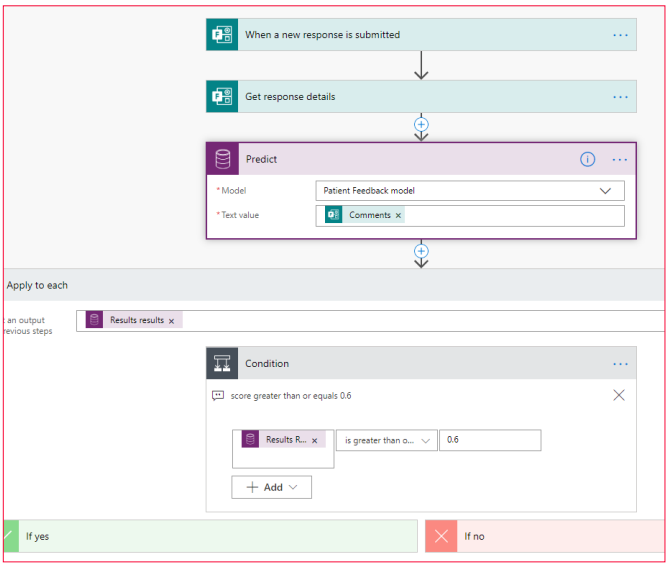


Imagen 8 .- Flujo de clasificación de comentarios.

En definitiva, lo que hacemos es obtener los datos de la respuesta a la encuesta y enviarlos a nuestro modelo de IA de clasificación de texto (en este ejemplo, llamado Patient Feedback model). Concretamente, enviaremos el texto introducido en el campo Comments de la misma.

> ***"El objetivo es que, a partir de los comentarios introducidos en la encuesta, la misma llegue al departamento o equipo al que le puede interesar para llevar a cabo las acciones que considere necesarias"***

Después de realizar el análisis del texto, el modelo nos devuelve el resultado: las etiquetas y la confiabilidad en el resultado, que no deja de ser un valor entre 0 y 1, y donde tendremos en cuenta los valores más cercanos a este último (en nuestro caso solo admitimos aquellos resultados donde el score es superior al 60%). Si el resultado es positivo, guardaremos el elemento en una lista de SharePoint y enviaremos un mensaje de Teams al equipo correspondiente, tal y como se puede comprobar en esta figura:

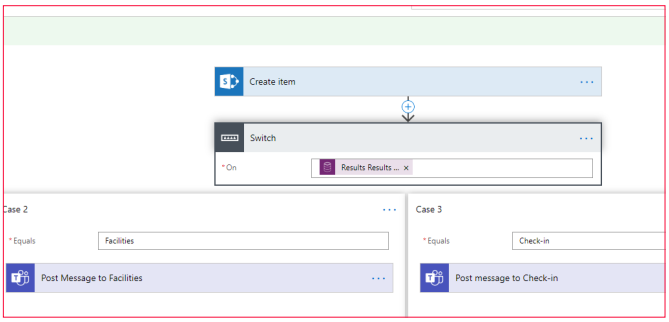


Imagen 9 .- Segunda parte del flujo de clasificación de comentarios.

Donde Facilities y Check-in son algunas de las etiquetas que se incluían en el fichero Excel, mediante el cual se ha entrenado el modelo de IA de clasificación de texto que hemos creado.

Conclusión

En este artículo hemos introducido al lector en el uso de los modelos de AI Builder, y concretamente en el de clasificación de texto. Mediante un ejemplo práctico hemos visto cómo podemos dotar de inteligencia a un proceso de feedback de forma rápida y sencilla, hecho que nos permite mejorar nuestra respuesta a determinadas situaciones.

La apuesta de Microsoft hacia el uso de AI en todos nuestros ámbitos (personal y laboral) es ya indudable, y el hecho de que se facilite el acceso y su uso permitirá agilizar y mejorar muchos de los procesos que se llevan a cabo actualmente en distintos sectores.

Ahora únicamente nos queda practicar, probar y, en definitiva, apostar por ellos. ¿Te subes al tren de la IA en la Power Platform?

FERRAN CHOPO GARCIA
IT Consultant & Trainer
ferran@ferranchopo.com
@fchopo
http://www.ferranchopo.com

5 https://docs.microsoft.com/es-es/powerapps/maker/common-data-service/create-solution

15

Configurar la Autenticación en Power Virtual Agent

Como sabréis Microsoft Power Virtual Agent es el nuevo miembro de la familia de la Power Platform. Microsoft ha creado una herramienta fantástica para crear interfaces conversacionales de una forma rápida y fácil. Esta plataforma está construida sobre Azure Bot Service y permite una extensibilidad muy grande utilizando toda la potencia de Power Platform.

"Microsoft Power Virtual Agent es el nuevo miembro de la familia de la Power Platform. Microsoft ha creado una herramienta fantástica para crear interfaces conversacionales de una forma rápida y fácil"

Aquí es donde radica todo su valor, los usuarios de negocio podrán crear una interfaz conversacional a medida dotándole de un flujo de conversación adaptado al contexto del usuario. Podrán integrar el ChatBot con las" Flow Actions". Acciones de llamada a Power Automate que nos permiten extender la lógica e integrar con sistemas externos mediante API. Si aun así, estas acciones no cubren todas las necesidades podrán agregar "Bot Framework Skills" a sus ChatBots para dotarles de funcionalidad extra.

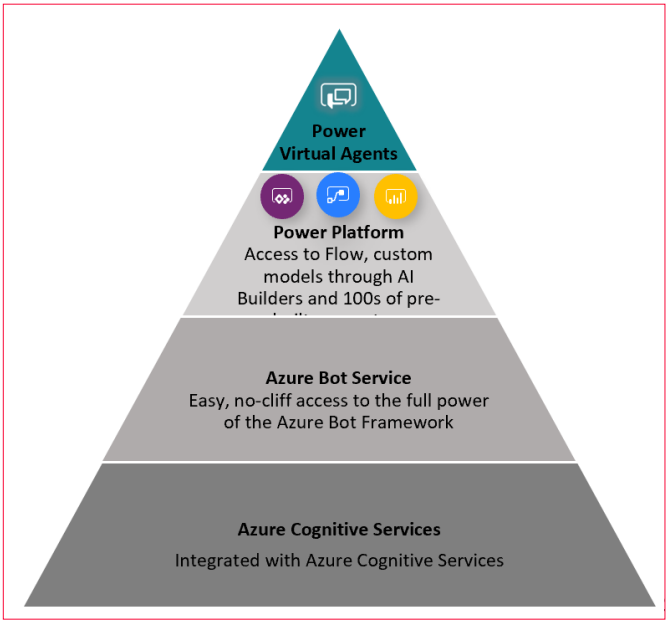


Imagen 1.- Power Virtual Agents Stack.

Una de las ventajas principales que tiene Power Virtual Agent es la Autenticación. La documentación existente sobre este tema es escasa y difusa. De modo que voy a tratar de explicar paso a paso como configurar la autenticación de Power Virtual Agent y espero que sirva de guía para los demás. Antes hay que decir que únicamente podemos configurar un único endpoint de autenticación por Bot y que Power Virtual Agents es compatible con cualquier proveedor de identidad que cumpla con el estándar OAuth2.

Para configurar la autenticación de Power Virtual Agent:

- Dirígete a la sección "Manage" y selecciona "Authentication" https://powerva.microsoft.com/#/manage/authentication.
- Veras un formulario vacío. Según la documentación de Microsoft estos son los valores que debemos introducir: https://docs.microsoft.com/en-us/power-virtual-agents/configuration-end-user-authentication

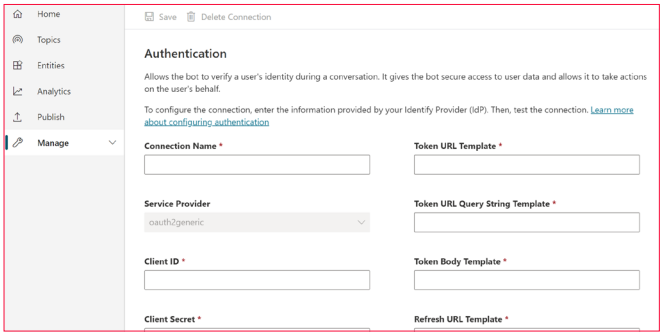


Imagen 2.- Formulario para configurar la autenticación.

- Para ello necesitamos registrar una aplicación en el AAD. Dirígete a https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps y inicia un nuevo registro de aplicación. Define un nombre y elige la opción de "Multitenant".

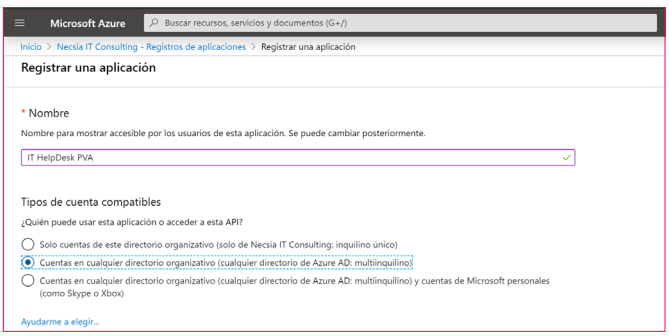


Imagen 3.- Aplicación de Azure AD a registrar.

- Una vez la tengas creada debes configurar la URI de redirección:
  - Selecciona en el menú “Autenticación”.
  - Agrega plataforma “Web”.
  - Escribe https://token.botframework.com/.auth/web/redirect
  - Selecciona la opción de “Access tokens”.
  - Selecciona “Configurar”.

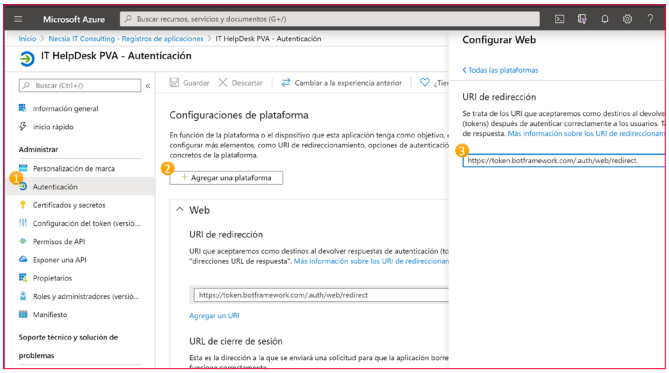


Imagen 4.- Configuración de URL de Redirección.

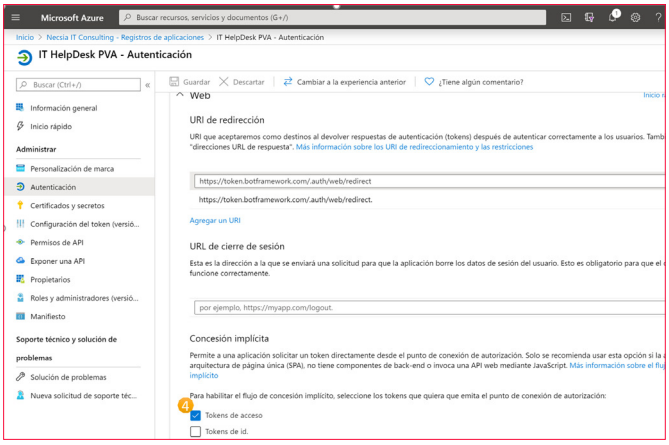


Imagen 5.- Configuración Tokens de acceso.

- Lo siguiente que debes hacer es dar permisos de Graph para la App. Para ello selecciona “API Permissions” en el menú lateral:
  - Añade un permiso.
  - Elige “Microsoft Graph”.
  - Elige la opción de “Delegated permisión”.
  - Selecciona la opción “openid”.
  - Selecciona “Agregar permiso”.

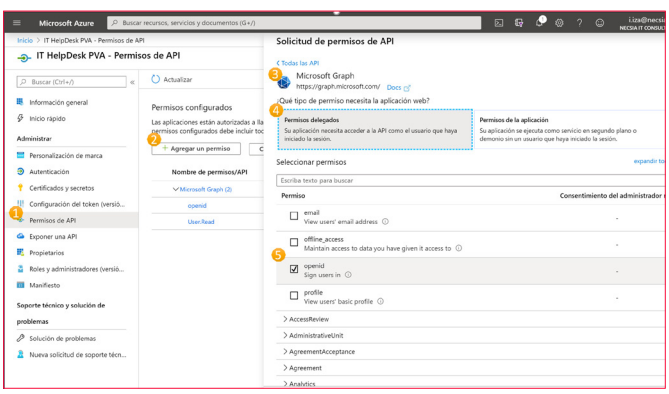


Imagen 6.- Configuración de permisos en el Graph.

- Por último, dirígete a “Certificados y secretos” y crea un nuevo secreto de cliente:

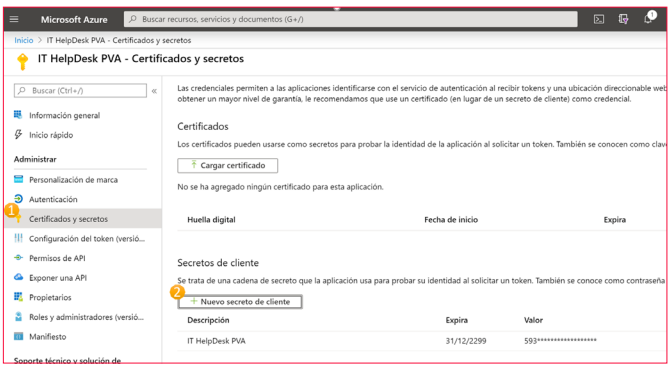


Imagen 7.- Creación de secretos de cliente.

- Con esto estaría la aplicación registrada en el AAD. Ahora debes copiar los valores del Client ID y Client Secret en la configuración de Autenticación de Power Virtual Agent y listo.

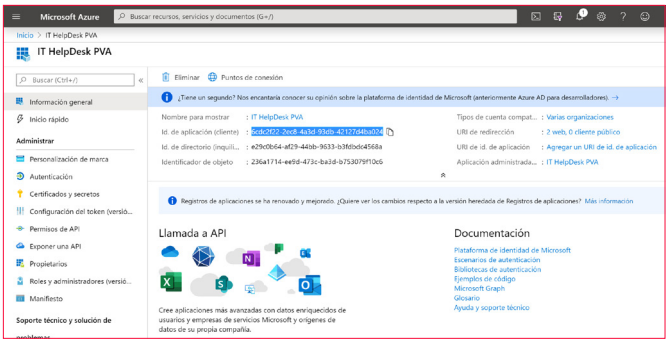


Imagen 8.- Copiando el ClientID y ClientSecret para el PVA.

"Únicamente podemos configurar un único endpoint de autenticación por Bot y que Power Virtual Agents es compatible con cualquier proveedor de identidad que cumpla con el estándar OAuth2"

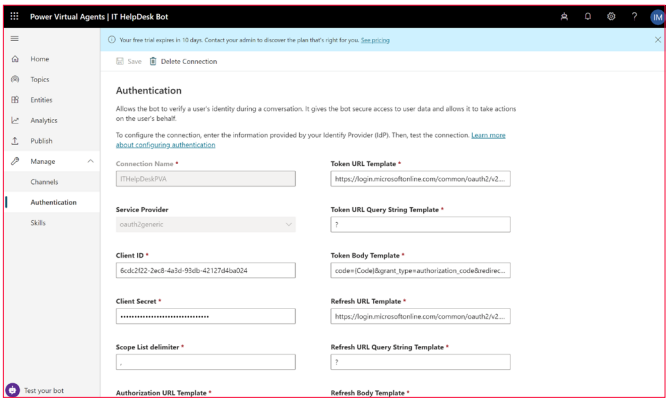


Imagen 9.- Configuración del ClientID y ClientSecret en el PVA.

- Agregamos una acción de Autenticación al flujo de conversación. Cuando lo ejecutes y el ChatBot solicite autenticación, te saldrá un recuadro de Login que pedirá un One Time Code. Introdúcelo en el chat de conversación y el Bot procederá a Autenticarse.

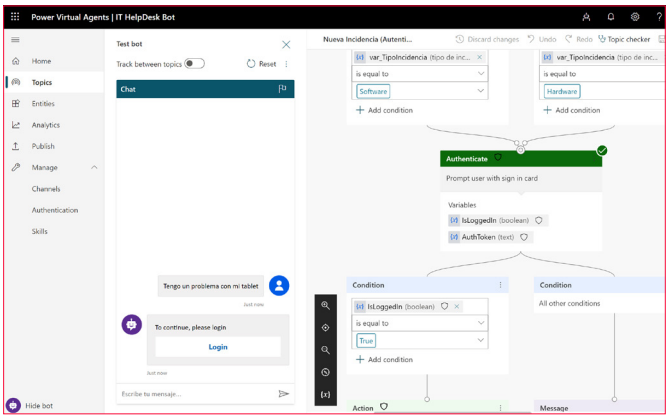


Imagen 10.- Añadiendo la acción de autenticación en el PVA.

"Cuando lo ejecutes y el ChatBot solicite autenticación, te saldrá un recuadro de Login que pedirá un One Time Code. Introdúcelo en el chat de conversación y el Bot procederá a Autenticarse"

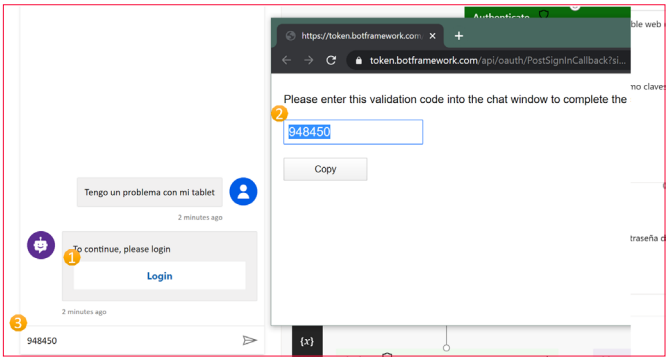


Imagen 11.- OneTime Code para logarse en el PVA.

- Una vez este autenticado, verás dos variables que devuelve el Bot, y que podemos usar después:
  - IsLoggedIn – Booleano devuelve si el usuario esta logado o no.
  - AuthToken – Bearer Token generado para el usuario.

Eso es todo, espero que os haya servido de ayuda para no volveros locos con la autenticación en Power Virtual Agent.

IMANOL IZA
Cloud Operations Manager, NECSIA
Microsoft MVP
@imaiza
Imanol.iza@gmail.com
www.imanoliza.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.

19

Uso de Funciones de Azure con Flow y Power Apps - Parte 3: Usando OpenAPI en Power Apps

Flow es la implementación que Microsoft ha creado para integrar el motor de flujos de Azure, Logic Apps, en Office 365. A su vez, Power Apps es un intento para facilitar la creación de aplicaciones sin necesidad de programación, que pueden ser utilizadas por sí mismas, o integradas en SharePoint Online. Ambos sistemas, aunque fáciles de utilizar y bastante poderosos en cuanto a funcionalidad, carecen de la flexibilidad para agregar nuevas posibilidades de cálculo y procesamiento. Esta falta de los dos sistemas se puede solucionar por medio de las Funciones de Azure.

"Microsoft Office Flow es el motor de flujos de trabajo creado por Microsoft en base a Azure Logic Apps"

Esta es la tercera parte de una serie de tres artículos:

- Como crear Funciones de Azure para que puedan ser utilizadas por Flow y PowerApps (CompartiMOSS No. 41, http://www.compartimoss.com/revistas/numero-41/uso-de-funciones-de-azure-con-flow-y-powerapps-part-1)
- Usando Funciones de Azure con Office Power Automate (CompartiMOSS No. 42, http://www.compartimoss.com/revistas/numero-42/uso-de-funciones-de-azure-con-powerautomate-y-powerapps-parte-2-usando-openapi-en-powerautomate).
- Usando Funciones de Azure con Power Apps (CompartiMOSS No. 43).

Introducción

Microsoft Office Flow es el motor de flujos de trabajo creado por Microsoft en base a Azure Logic Apps. Los flujos que se pueden crear están totalmente basados en componentes estándar que ofrecen una lógica interna de trabajo (loops, estamentos, etc.) y conexión a otros sistemas (Exchange, SharePoint y muchos otros conectores, internos y externos a Microsoft). El principal problema de esta forma de trabajo es que no se puede crear ("programar") nueva funcionalidad dentro del sistema mismo. Lo mismo se puede decir de Power Apps: aunque ofrece conectividad con muchos otros tipos de sistemas, no es posible definir capacidades de cálculo dentro de la aplicación misma.

Para solucionar el problema, ambos sistemas permiten la utilización de "OpenAPI", un estándar internacional que fue creado por un consorcio de industrias que se propuso unificar la forma cómo se describen los APIs de REST, creando un formato de descripción neutral y no controlado por cualquier proveedor comercial (https://www.openapis.org). OpenAPI está basado a su vez en "Swagger", una manera conocida desde hace mucho tiempo para describir APIs de REST.

Aunque REST (REpresentational State Transfer) es una forma unificada para crear y utilizar APIs por medio de internet, la forma de usarlo es más un Framework que un estándar. Por tal motivo, los APIs de REST, como varían de uno a otro, se deben describir mediante una definición de OpenAPI para que otros sistemas "entiendan" como usar el API. Esta definición (OpenAPI) contiene información sobre qué operaciones están disponibles en una API y cómo se deben estructurar sus datos de solicitud y respuesta. Haciendo que Power Automate y Power Apps puedan utilizar OpenAPI hace que, a su vez, los dos sistemas estén abiertos a usar cualquier clase de funcionalidad proporcionada por cualquier tipo de sistemas externos.

Por su lado, las Funciones de Azure proporcionan toda la infraestructura técnica para poder crear funcionalidad "serverless", es decir, que los desarrolladores se pueden enfocar en crear el código que se necesita, sin necesidad de ocuparse de servidores, redes, rendimiento bajo carga, etc. Funciones de Azure se pueden programar en una variedad de idiomas de programación (CSharp, PowerShell, Python, etc.), utilizando Visual Studio, Visual Studio Code o directamente desde un navegador en el sitio de diseño de Funciones del portal de Azure. El problema, a su vez, con Funciones es que la definición Swagger que generan no es OpenAPI ni utilizable directamente por Power Automate o Power Apps.

Microsoft ha publicado parches para poder crear Funciones de Azure que puedan ser utilizadas directamente desde Power Automate y Power Apps, pero es requerido que tanto Azure como Office 365 utilicen el mismo Directorio Activo, lo que generalmente no es el caso en aplicaciones Enterprise. La forma para solucionar el problema es utilizar el Azure API Management, un servicio de Azure que permite exponer APIs al mundo externo por medio de OpenAPI.

En este tercer artículo de la serie de tres, se indica como modificar la definición OpenAPI de la función de Azure, que es expuesta al mundo externo por medio del servicio de Azure de API Management, para que pueda ser utilizada desde Power Apps. El ejemplo a continuación es un sistema de reserva de salas de conferencias en una empresa:

empleados pueden reservar una sala de conferencias desde una Lista de SharePoint; un Power Automate acoplado a la Lista pasa los datos de la reserva a la Función de Azure que calcula si la reservación es posible, o si el sitio está ocupado y retorna una indicación al respecto al flow, el que retransmite la información a SharePoint y al usuario. Todo el sistema de reserva se puede utilizar también desde una aplicación de Power Apps, como se muestra en este artículo. El algoritmo para determinar si una sala está ocupada o no, no es implementado, solamente se ha simulado por medio de un generador random. Pero el ejemplo indica claramente el potencial que ofrece la combinación de los cuatro sistemas.

Modificar la definición de OpenAPI

Si se intenta utilizar el conector creado en el segundo artículo de la serie con la definición OpenAPI de la función, Power Apps lo rechaza con un error:

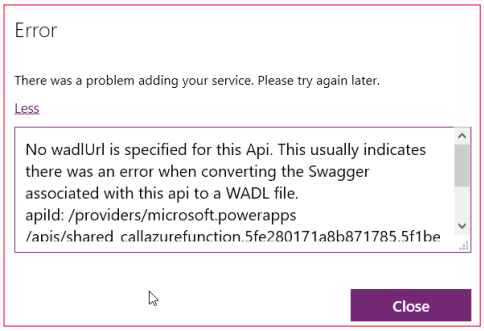


Imagen 1.- Error en el conector basado en la definición de OpenAPI de la función de Azure.

El error es bastante críptico y no da ninguna indicación sobre el problema con la definición de OpenAPI. Para poder usar el conector en Power Apps, es indispensable solucionar primero el problema, como se indica a continuación.

1.– Abra el sitio de Power Apps desde el portal de Office (https://portal.office.com), o desde el portal de Power Apps mismo (https://powerapps.com) y lóguese con credenciales que tengan permisos suficientes para crear y utilizar aplicaciones.

2.– Expanda el menú de "Data" (menú vertical al lado izquierdo) y haga clic sobre "Custom connectors". Use el botón con el icono de un lápiz al lado del conector creado en el punto 6 del segundo artículo de la serie (conectores son iguales para Power Automate y Power Apps). Esto abre la configuración del conector en forma de edición. Haga clic sobre "3. Definition".

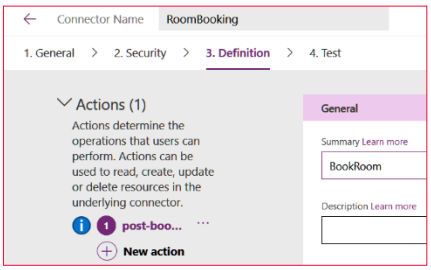


Imagen 2.- Modificar la definición de OpenAPI.

3.– Primero haga clic sobre el botón de elipse al lado derecho del método "get-" y use "Delete". Este método no es utilizado y, además, contiene errores que impiden usarlo en Power Apps.

4.– Luego seleccione el método "post-" y desplace la ventana hacia abajo, hasta la sección de "Response". Note que no hay una respuesta definida porque el OpenAPI exportado desde Azure API Management no detecta que haya una respuesta desde la función, y no crea una respuesta en la definición (esto hace que la definición no sea conforme al estándar de OpenAPI, y que Power Apps rechace el conector). Para solucionar el problema, use el botón de "+Add default response", acepte los valores por defecto y haga un update del conector. El resultado es una respuesta por defecto definida en el conector:

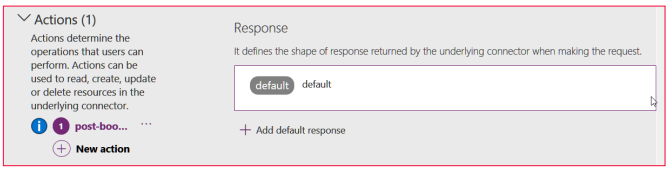


Imagen 3.- Conector con la definición de OpenAPI modificada.

"Si se intenta utilizar el conector creado en el segundo artículo de la serie con la definición OpenAPI de la función, Power Apps lo rechaza con un error"

Crear la aplicación de Power Apps

5.– Desde el portal de Power Apps haga clic sobre "Apps" y sobre "+Create an app" - "Canvas" - "Start with a blank canvas" - "Phone layout" para crear una nueva Power App.

6.– Agregue dos labels ("Room Name" y "Persons"), dos text boxes ("txtRoomName" y "txtPesons"), un botón ("btnGetRoom") y un label ("lblResult") para mostrar el resultado.

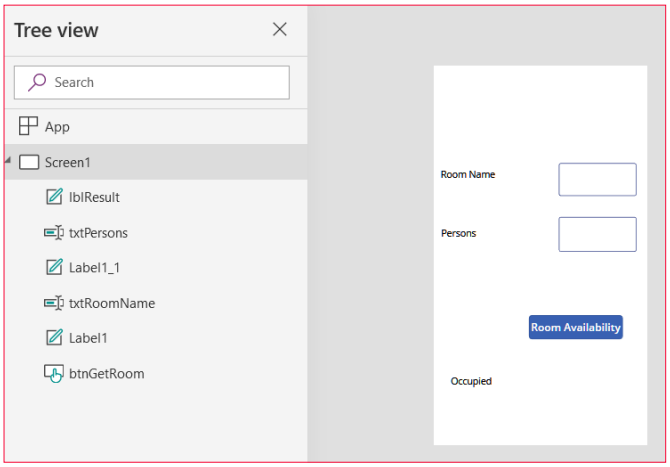


Imagen 4.- La aplicación de PowerApps.

7.– Agregue el conector. Seleccione la pestaña de "View" (menú horizontal superior) y "Data sources". En la nueva ventana use "+Add data source", busque el conector y haga clic sobre él. Si las modificaciones han sido correctas, el conector aparecerá en la lista de "Data" sin mostrar errores:

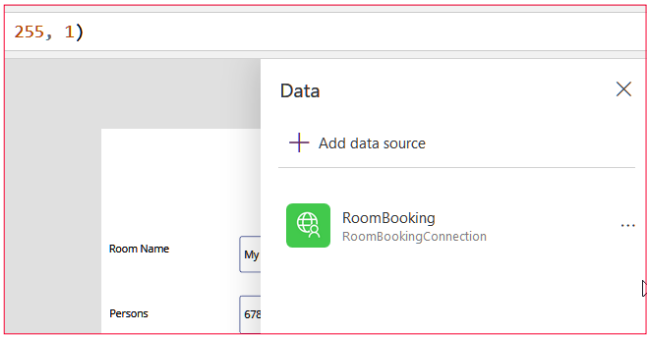


Imagen 5.- El conector en la aplicación de PowerApps

8.– Cierre la ventana de "Data" y haga clic sobre el botón de la aplicación. En el método "OnSelect" utilice el siguiente fragmento de código:

```
UpdateContext({RoomAvailable: RoomBooking.postbookroom({roomname:txtRoomName.Text, persons:txtPersons.Text})})
```

"Para darle más flexibilidad y capacidad de interacción con otros sistemas, Microsoft Power Automate y Power Apps pueden utilizar conectores a procedimientos externos. Azure Functions es el método ideal para crear esa funcionalidad"

En este fragmento se crea una variable llamada "RoomAvailable" por medio del método "UpdateContext". A la variable se le asigna el resultado de llamar el método "RoomBooking.postbookroom", que utiliza dos parámetros de entrada: "txtRoomName.Text" para "roomname" y "txtPesons.Text" para "persons". Note que "RoomBooking" es el nombre del conector y "postbookroom" el nombre de la operación (sin "-") que indica la definición del conector. De igual forma, "roomname" y "persons" son los dos parámetros de entrada que el OpenAPI ha definido en el QueryString de entrada del conector.

9.– Haga clic sobre el label para mostrar los resultados en la aplicación, y en su propiedad de "Text" seleccione "RoomAvailable". Esta es la variable que el botón crea y asigna un valor cuando la función de Azure es llamada

10.– Use el "App Checker" (esquina superior derecha) para comprobar que la aplicación no tiene errores

11.– Para testear el funcionamiento de la aplicación, use el botón de "Preview the app" en la esquina superior derecha, o use F5. Inserte dos valores en las dos casillas de texto y use el botón. El resultado deberá aparecer en el label de resultados:

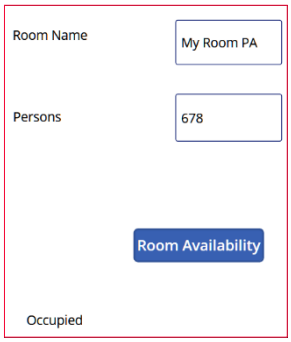


Imagen 6.- La aplicación funcionando .

La aplicación hace una llamada REST al Azure API Management, y este a su vez llama a la función. El funcionamiento de la función se puede seguir abriendo la función misma y su ventana de Logs:

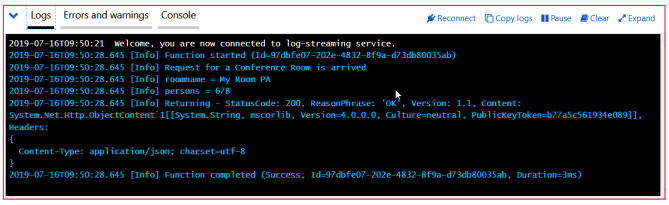


Imagen 7.- La Función de Azure trabajando con los datos de la aplicación .

12.– Note que también es posible crear una aplicación de Power Apps para la Lista de SharePoint utilizada en el segundo artículo. Pero en ese caso, la aplicación no es más que un reemplazo de la interfaz por defecto de la Lista en SharePoint, que lo que hace es crear un elemento en la Lista, el que dispara el flujo acoplado a ella. En este articulo hemos visto como crear una aplicación de Power Apps que puede funcionar totalmente independiente de SharePoint y Power Automate.

Conclusión

Para darle más flexibilidad y capacidad de interacción con otros sistemas, Microsoft Power Automate y Power Apps pueden utilizar conectores a procedimientos externos. Azure Functions es el método ideal para crear esa funcionalidad. En esta serie de tres artículos se indica como crear funciones de Azure y hacerlas funcionar bajo el OpenAPI estándar (primer artículo), como conectar Power Automate con la función (segundo articulo) y como puede utilizar Power Apps la misma función (tercer articulo).

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

22

CRM, Dynamics 365, CDS y Power Apps

Las soluciones de negocio están de moda. Una vez consolidada la ola del salto al Cloud, ya casi nadie la discute. Es el momento de apuntar hacia el nuevo hito. Ahora mismo buena parte de la industria ve la oportunidad de negocio en las soluciones de negocio. Creo que es la evolución natural, ya que los equipos de I+D de los diferentes fabricantes tienen más y mejor control de las nuevas soluciones, lo que permite implementar soluciones de forma muchísima más acelerada.

"Según muchas consultoras internacionales, Microsoft se encuentra en la punta de lanza de la innovación y desarrollo de estas nuevas plataformas que permiten mejorar procesos de negocio mediante la implementación de plataformas tecnológicas"

Según muchas consultoras internacionales, Microsoft se encuentra en la punta de lanza de la innovación y desarrollo de estas nuevas plataformas que permiten mejorar procesos de negocio mediante la implementación de plataformas tecnológicas. A pesar de que considero que Microsoft dispone de una de las plataformas mas solventes y abiertas en cuanto a las aplicaciones de negocio, creo los cambios de nombres y la aparición de nuevas soluciones, puede provocar algo de confusión en algunos casos.

Un poco de historia

Microsoft ha entrado en el mundo del CRM en 2003 con la compra de una empresa que disponía de una solución de este tipo. A partir de allí ha tenido una evolución mantenida durante muchos años hasta su llegada a la versión cloud. Durante este tiempo, la evolución de la solución ha llevado dos caminos que siempre fueron unidos en cada nueva release:

- Funcionalidades relacionadas con propiedad intelectual (IP), como por ejemplo requerimientos típicos de departamentos de marketing, de ventas o de atención al cliente. Estas funcionalidades permiten que dichos departamentos dispongan de una serie de funcionalidades de forma estándar que les permite trabajar desde el día uno.
- Funcionalidades relacionadas con la plataforma, que permiten extender la misma, y crear procesos de negocio propios a cada tipo de cliente. En este punto vemos funcionalidades como integraciones con toda la plataforma de Office365, herramientas para implementar automatismos o validaciones configurables, procesos de negocio a golpe de click.

Con la compra del "CRM", se adquiría toda la solución, que permitía aprovecharse de todo lo descrito, tanto de la IP como de la plataforma.

Separación de la IP - Plataforma

En muchas implementaciones tradicionales de Dynamics, los clientes no hacen uso de las funcionalidades derivadas de la IP, sino que utilizaban simplemente la plataforma para implementar procesos de negocio nuevos, nada que ver con los procesos estándar que traía la solución. De esta manera muchos hemos utilizado durante años en proyectos la plataforma de Dynamics 365 for Customer Engagement como un acelerador para la implementación de cualquier tipo de desarrollo o para digitalizar cualquier tipo de proceso de negocio.

Microsoft tomó nota de esto y comenzó a separar estos dos mundos, de cara a conseguir que incluso existan roadmaps diferentes entre uno y otro. Esto ha permitido que requerimientos de evolución de IP, generen nuevas funcionalidades de plataforma, y así existan sinergias entre ambas. Se seguía estando en el mismo mundo del Dynamics 365 for Customer Engagement (que era la plataforma), y la cual disponía de Apps como Sales, Marketing, Customer Service, Field Service, etc.

El reflejo técnico de este enfoque fue que todas estas Apps, se convirtieron en Soluciones que se instalaban sobre la plataforma, al igual que cualquier App o vertical que se puede conseguir en el Appsource. La "única" diferencia es que estas Apps las licenciaba y desarrollaba directamente Microsoft.

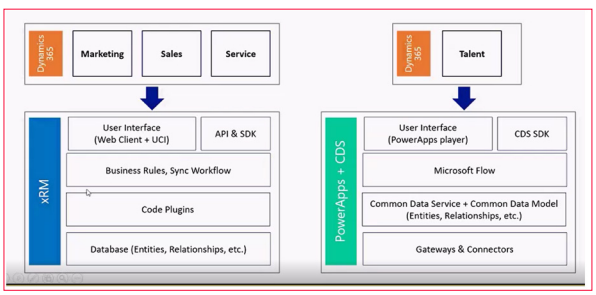


Imagen 1.- Esquema de soluciones Dynamics 365.

PowerApps + CDS

Microsoft identificó claramente que las plataformas de soluciones negocio y "low code" serían el siguiente escalón al cual se dirigía la industria. De esta manera comenzó a crear una nueva plataforma que llamó PowerApps, con un sitio donde crear entidades y campos que llamo Common Data Service (CDS 1.0). Inicialmente permitía crear aplicaciones rápidas con muchos conectores, y la posibilidad de crear entidades y campos. El rápido crecimiento de esta plataforma llevó a que sean necesarias nuevas funcionalidades más "Enterprise ready", como por ejemplo despliegues entre entornos, una SDK potente, portales de administración gestión avanzada de licencia y posibilidades de extensibilidad y personalización mucho más avanzadas.

En este punto es donde Microsoft decide detener este desarrollo y mirar hacia la plataforma de Dynamics 365 for Customer Engagement. Allí encontró que había muchos elementos comunes y así se tomó la decisión de unificar estos dos mundos:

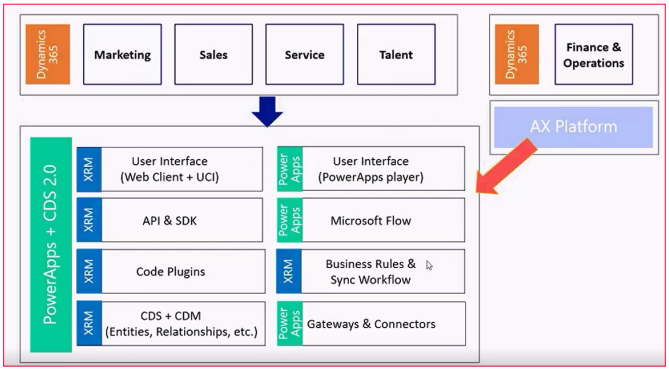


Imagen 2.- Unificación de los dos mundos en Dynamics 365.

Con esta decisión, todos los consultores de Dynamics 365 (CRM) se convirtieron en expertos en el nuevo CDS, ya que en realidad se tratan de la misma solución. Con esta unión nos quedan algunos temas algo duplicados. Por ejemplo, tenemos dos formas de crear aplicaciones, las Canvas Apps, y las Model Driven Apps:

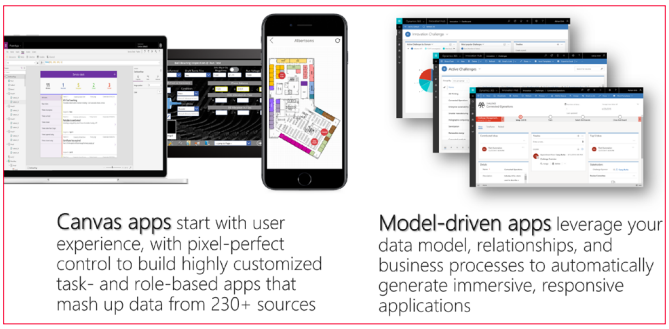


Imagen 3.- Tipos de Apps que se pueden crear.

También disponemos de los "antiguos" Workflows de Dynamics 365, y los nuevos Power Automate. Adicionalmente se ha añadido a todo el set de soluciones conocidos como PowerPlatform (que incluye Power BI, Power Apps, y Power Automate).

CDS Platform

Finalmente, hoy en día lo conocido como Dynamics 365 for Customer Engagement pasó a llamarse CDS Platform. Cuando alguien crea un entorno de CDS con su base de datos, se refiere a la plataforma de "CRM" tradicional pero solamente con las funcionalidades de extensibilidad de la plataforma y unas pocas (mínimas) entidades, como por ejemplo Cuenta, Contacto, actividades y poco más. Cuando creamos un entorno de Power Apps nos pregunta si queremos crear una base de datos. Esto quiere decir, un entorno de CDS.

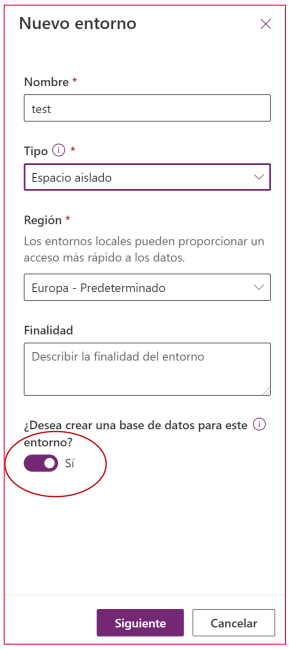


Imagen 4.- Creación de un entorno PowerApps.

Y una vez que tenemos ese entorno, podemos añadirle o no Apps de Dynamics 365.

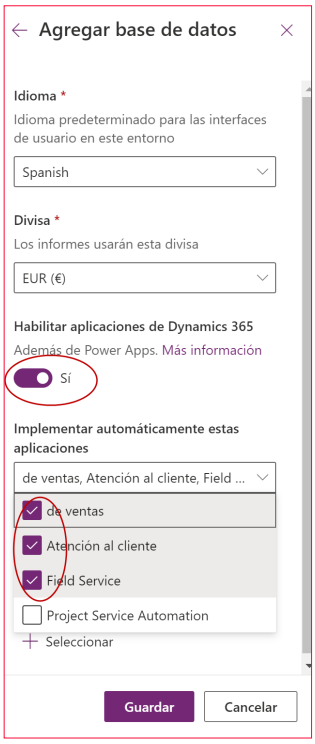


Imagen 5.- Configuración de Apps a añadir.

> ***"El reflejo técnico de este enfoque fue que todas estas Apps, se convirtieron en Soluciones que se instalaban sobre la plataforma, al igual que cualquier App o vertical que se puede conseguir en el Appsource"***

Hoy en día es posible adquirir la plataforma por separado de las soluciones de Dynamics 365 CE, pero estas siguen siendo construidas sobre la misma plataforma. Esto nos permite comprar otras licencias de "PowerApps" para que en el mismo entorno convivan las Apps de Dynamics 365 con nuestras propias Apps personalizadas. Adicionalmente, al crear un portal (Power Apps Portal), lo que realmente se crea es un entorno de CDS con las Apps que anteriormente se llamaban Dynamics 365 Portals. Finalmente, aquí es donde podemos ver que un entorno por ejemplo con Dynamics 365 for Sales, lo que en realidad quiere decir es que se dispone de un entorno de CDS, con la App de Sales instalada.

El futuro del CDS y Power Apps

La evolución del CDS y las Power Apps, está provocando una auténtica revolución tecnológica en Microsoft. Nuevas soluciones como Talent o el nuevo Project ya están implementadas desde el inicio y de forma nativa con CDS, y ojalá venga más. Para seguir la evolución de Dynamics 365 CE y de la plataforma CDS ya disponemos de dos documentos por separado, uno para el CDS y otro para Dynamics 365:

- https://docs.microsoft.com/en-us/dynamics365-release-plan/2020wave1/
- https://docs.microsoft.com/en-us/power-platform-release-plan/2020wave1/

Sin duda se nos vienen tiempos emocionantes en el mundo de las aplicaciones de negocio y en especial a los que trabajamos con Dynamics 365 y la Power Platform.

DEMIAN RASCHKOVAN
Microsoft Dynamics CRM MVP

25

Azure KeyVault + docker

En este articulo quiero explicaros como podemos desplegar nuestro entorno de desarrollo de una forma fácil y segura. Para ello vamos a usar docker para poder crear un contenedor con nuestra aplicación “API NET Core 3.1” que conecta contra nuestra base de datos Azure SQL de forma segura sin tener que exponer nuestras credenciales. Docker es una plataforma para desarrolladores y sysadmins (utilizando la filosofía DevOps) que nos permite desarrollar, desplegar y ejecutar aplicaciones en contenedores de una forma fácil y sencilla.

"Azure KeyVault nos permite centralizar el almacenamiento de los secretos de aplicación. Esto nos permite controlar su distribución"

Para ello vamos a empezar creando nuestra API en .NET Core 3.1, pare ello usaremos el siguiente comando:

```
dotnet new API -n Azuretraining
```

Esto nos creara la estructura de nuestro proyecto con el nombre que le hemos asignado «Azuretraining» tal y como podemos observar en la siguiente imagen:

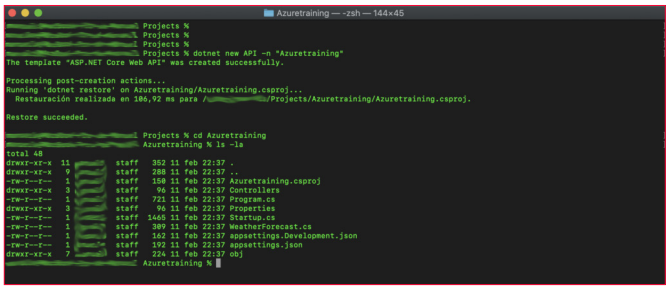


Imagen 1.- Estructura del proyecto tras el lanzamiento del comando anteriormente mencionado.

Ahora vamos a agregar el nuget para poder trabajar con SQL:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.InMemory
```

Incorporaremos una nueva clase al modelo, para este ejemplo definiremos un modelo de ejemplo llamado «Courses» dentro de nuestra carpeta Models:

```
namespace Azuretraining.Models
{
  public class Course
  {
    public long Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public int Capacity {get; set;}
    public double Qualification {get; set;}
    public string Modality {get; set;}
    public string Category {get; set;}
    public bool IsComplete { get; set; }
  }
}
```

Una vez tenemos nuestro modelo incorporaremos el contexto de base de datos, será la clase principal que coordina la funcionalidad de Entity Framework para un modelo de datos. Para ello agregaremos a nuestra carpeta Models un nuevo fichero llamado «CourseContext.cs» con el siguiente formato:

```
using Microsoft.EntityFrameworkCore;

namespace Azuretraining.Models
{
  public class CourseContext : DbContext
  {
    public CourseContext(DbContextOptions<CourseContext>
options)
      : base(options)
    {
    }

    public DbSet<Course> Courses { get; set; }
  }
}
```

Ahora deberemos de modificar nuestro fichero «Startup.cs» agregando las referencias necesarias para usar los servicios:

```
using Microsoft.EntityFrameworkCore;
using Azuretraining.Models;
```

Y modificaremos nuestra función «ConfigureServices» para que tenga el siguiente aspecto:

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddDbContext<CourseContext>(opt =>
    opt.UseInMemoryDatabase("CourseList"));
  services.AddControllers();
}
```

Esto nos proporcionará poder usar una BD en memoria para realizar unas primeras pruebas antes de atacar a nuestra BD en la nube. Una vez lo tenemos todo preparado vamos a agregar los Nugets necesarios para poder hacer el scaffolding y generar de forma automática nuestro Controller con las siguientes instrucciones:

```
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet tool install --global dotnet-aspnet-codegenerator
dotnet aspnet-codegenerator controller -name CoursesController -async -api -m Course -dc CourseContext -outDir Controllers
```

Veremos que en la carpeta Controller nos ha generado el archivo «CoursesController.cs» donde tendremos definida todas las acciones de nuestra API. Ahora ejecutaremos nuestra aplicación y en un explorador introducimos la siguiente URL: https://localhost:5001/api/courses.

Imagen 2.- Resultado de la invocación de la API al controlador Courses.

Ahora vamos a modificar nuestra aplicación para que conecte directamente con nuestra BD de Azure SQL. Para ello previamente deberemos haber creado nuestra instancia, podemos usar Azure CLI para poder hacerlo como se muestra en el siguiente ejemplo:

```
az sql server create --subscription "NOMBRE DE LA SUSCRIPCIÓN" --name trainginappDB --resource-group TrainingApp --location "West Europe" --admin-user "NOMBRE DE USUARIO" --admin-password "PASSWORD"

az sql server firewall-rule create --subscription "NOMBRE DE LA SUSCRIPCIÓN" --resource-group TrainingApp --server trainginappdb --name AllowAllIps --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0

az sql db create --subscription "NOMBRE DE LA SUSCRIPCIÓN" --resource-group TrainingApp --server trainginappdb
--name TrainingApp --service-objective S0
```

Una vez tenemos creada nuestra instancia de BD en Azure SQL, vamos a preparar nuestra solución para «dockerizar», para ello generaremos un fichero. Dockerfile con el siguiente contenido:

```
# https://hub.docker.com/_/microsoft-dotnet-core
 FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
  WORKDIR /app

 # copy csproj and restore as distinct layers
  COPY *.csproj ./
  RUN dotnet restore

 # copy everything else and build app
  COPY . ./
  RUN dotnet publish -c release -o out --no-restore

 # final stage/image
  FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
  WORKDIR /app
  COPY --from=build /app/out .
  ENTRYPOINT ["dotnet", "azuretraining.dll"]
```

Y un fichero «.dockerignore» en nuestra solución con el siguiente contenido:

```
# directories
**/bin/
**/obj/
**/out/

# files
Dockerfile*

**/*.md
```

Para nuestra cadena de conexión usaremos Azure KeyVault para poder proteger nuestro «secretos», para ello iremos al portal de Azure y crearemos un nuevo Azure KeyVault. Una vez creado vamos a Secrets -> Generate/Import como se puede apreciar en la siguiente captura:

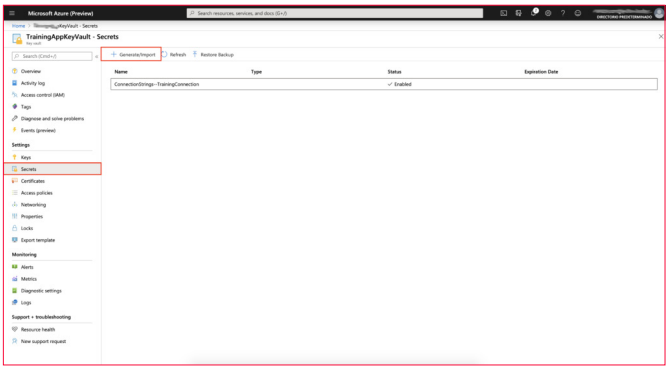


Imagen 3.- Azure KeyVault en el portal de Azure.

En la siguiente pantalla deberemos de indicar que es una entrada manual, le damos un nombre a nuestro secreto, en este caso «ConnectionStrings–TrainingConnection» esto se debe a que en nuestro fichero «appsettings.json» tenemos la definición de nuestro ConnectionStrings de la siguiente forma y para que el KeyVault pueda insertar el valor en tiempo de ejecución debemos de separarlos con «–» el nombre concatenando la relación padre-hijo:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    },
    "ConnectionStrings": {
      "TrainingConnection": ""
    }
  },
  "AllowedHosts": "*"
}
```

Imagen 4.- Contenido del fichero appsettings.json

Ahora añadimos la cadena de conexión hacia nuestro Azure SQL que nos facilita cuando creamos el servicio, como se puede apreciar en la siguiente captura:

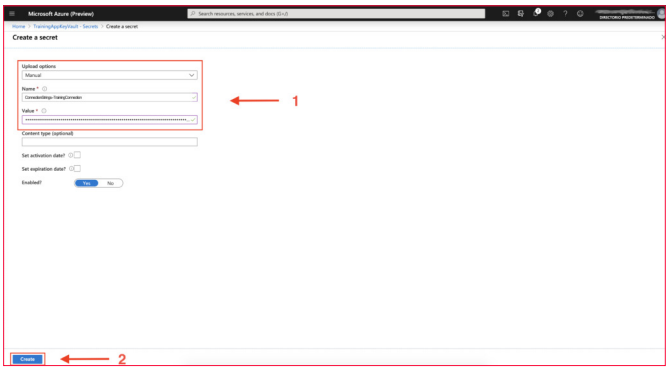


Imagen 5.- Alta de un secreto en Azure KeyVault desde el portal de Azure.

"Una vez que ya tenemos nuestro KeyVault para poder proteger nuestros «secretos» vamos a modificar nuestro proyecto para poder usarlo, para ello necesitaremos añadir los siguientes paquetes Nuget"

Una vez que ya tenemos nuestro KeyVault para poder proteger nuestros «secretos» vamos a modificar nuestro proyecto para poder usarlo, para ello necesitaremos añadir los siguientes paquetes Nugets:

```
dotnet add package Microsoft.Azure.KeyVault
dotnet add package Microsoft.Azure.Services.AppAuthentication
dotnet add package Microsoft.Extensions.Configuration.AzureKeyVault
```

Modificaremos nuestro archivo «Program.cs» añadiremos los imports necesarios:

```
using Microsoft.Azure.KeyVault;
using Microsoft.Azure.Services.AppAuthentication;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Configuration.AzureKeyVault;
```

Sustituiremos el método IHostBuilder para poder obtener la información de nuestro KeyVault y asignarlo el siguiente formato:

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
  Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((ctx, builder) =>
    {
      var keyVaultEndpoint = GetKeyVaultEndpoint();
      if (!string.IsNullOrEmpty(keyVaultEndpoint))
      {
          var azureServiceTokenProvider = new AzureServiceTokenProvider();
        var keyVaultClient = new KeyVaultClient(
          new KeyVaultClient.AuthenticationCallback(
              azureServiceTokenProvider.KeyVaultTokenCallback));
        builder.AddAzureKeyVault(
            keyVaultEndpoint, keyVaultClient, new DefaultKeyVaultSecretManager());
      }
    })
    .ConfigureWebHostDefaults(webBuilder =>
    {
      webBuilder.UseStartup<Startup>();
    });
static string GetKeyVaultEndpoint() => Environment.GetEnvironmentVariable("KEYVAULT_ENDPOINT");
```

Ahora agregaremos en el environment (todo esto lo hacemos para que la acción se realice en tiempo de ejecución), para ello nos fijaremos que en la última linea de nuestro «Program.cs» indicábamos obtener de la variable «KEYVAULT_ENDPOINT» en ella declararemos la URL de nuestro Azure KeyVault, esta información la deberemos de añadirla a nuestro fichero «launch.json» con el siguiente formato:

```
{
  // Use IntelliSense to find out which attributes exist for C# debugging
  // Use hover for the description of the existing attributes
  // For further information visit https://github.com/OmniSharp/omnisharp-vscode/blob/master/debugger-launchjson.md
  "version": "0.2.0",
  "configurations": [
    {
      "name": ".NET Core Launch (web)",
      "type": "coreclr",
      "request": "launch",
      "preLaunchTask": "build",
      // If you have changed target frameworks, make sure to update the program path.
      "program": "${workspaceFolder}/bin/Debug/netcoreapp3.1/trainingapp.courses.dll",
      "args": [],
      "cwd": "${workspaceFolder}",
      "stopAtEntry": false,
      // Enable launching a web browser when ASP.NET Core starts. For more information: https://aka.ms/VSCode-CS-LaunchJson-WebBrowser
      "serverReadyAction": {
        "action": "openExternally",
        "pattern": "^\\s*Now listening on:\\s+(https?://\\S+)"
      },
      "env": {
        "ASPNETCORE_ENVIRONMENT": "Development",
          "KEYVAULT_ENDPOINT": "https://NOMBREDENUESTROKEYVAULT.vault.azure.net/"
      },
      "sourceFileMap": {
        "/Views": "${workspaceFolder}/Views"
      }
    },
    {
      "name": ".NET Core Attach",
      "type": "coreclr",
      "request": "attach",
      "processId": "${command:pickProcess}"
    }
  ]
}
```

Por último, sustituiremos en nuestro fichero «Startup.cs» la conexión de la BD en memoria por la conexión hacia nuestro Azure SQL con la configuración que hemos preparado en los pasos anteriores, y quedará de la siguiente forma:

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<CourseContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("TrainingConnection")));

    services.BuildServiceProvider().GetService<CourseContext>().Database.Migrate();

    services.AddApplicationInsightsTelemetry();
    services.AddControllers();
}
```

Imagen 6.- Contenido del fichero Startup.cs.

Este primer servicio es la conexión que realizamos para conectar con nuestra «Cadena de conexión» securizada:

```
services.AddDbContext(options =>
        options.UseSqlServer(Configuration.GetConnectionS-
tring("TrainingConnection")));
```

Este segundo servicio nos permitirá crear las tabla y estructura iniciales en caso de que no lo tengamos:

```
services.BuildServiceProvider().GetService().Database.Mi-
grate();
```

Ahora lanzamos nuestra aplicación y vemos que nos ha funcionado correctamente:

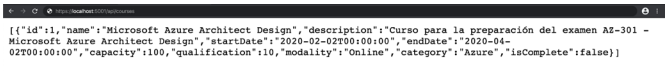


Imagen 7. Resultado de la ejecución de la API.

ATENCIÓN: Como hemos podido ver hasta aquí lo único que hemos hecho es indicar la url de nuestro Azure KeyVault para poder recuperar la información de la cadena de conexión, pero el «truco» es que si no estamos logados en nuestro azure CLI en local no podremos usarlo y nos devolverá el siguiente error:

```
Startup.cs(34,13): warning ASP0000: Calling 'BuildServicePro-
vider' from application code results in an additional copy of sin-
gleton services being created. Consider alternatives such as
dependency injecting services as parameters to 'Configure'. [/
Users/msanchez/Projects/Azuretraining/Azuretraining.csproj]
    Unhandled exception. System.ArgumentNullExcep-
tion: Value cannot be null. (Parameter 'connectionString')
            at Microsoft.EntityFrameworkCore.Utilities.Check.
NotEmpty(String value, String parameterName)
      at Microsoft.EntityFrameworkCore.SqlServerDbContex-
tOptionsExtensions.UseSqlServer(DbContextOptions-
Builder optionsBuilder, String connectionString, Action1
sqlServerOptionsAction)  at Azuretraining.Startup.<Configu-
reServices>b__4_0(DbContextOptionsBuilder options)
in /Users/msanchez/Projects/Azuretraining/Startup.cs:line
32  at Microsoft.Extensions.DependencyInjection.EntityFra-
meworkServiceCollectionExtensions.<>c__DisplayClass1_02.
b__0(IServiceProvider p, DbContextOptionsBuilder b)
      at Microsoft.Extensions.DependencyInjection.EntityFra-
meworkServiceCollectionExtensions.CreateDbContex-
tOptions[TContext](IServiceProvider applicationService-
Provider, Action2 optionsAction)  at Microsoft.Extensions.
DependencyInjection.EntityFrameworkServiceCollectionEx-
tensions.<>c__DisplayClass10_01.b__0(IServiceProvider p)
      at Microsoft.Extensions.DependencyInjection.Service-
Lookup.CallSiteRuntimeResolver.VisitFactory(Factory-
CallSite factoryCallSite, RuntimeResolverContext context)
      at Microsoft.Extensions.DependencyInjection.Service-
Lookup.CallSiteVisitor2.VisitCallSiteMain(ServiceCallSite
callSite, TArgument argument)  at Microsoft.Extensions.De-
pendencyInjection.ServiceLookup.CallSiteRuntimeResolver.
VisitCache(ServiceCallSite callSite, RuntimeResolverContext
context, ServiceProviderEngineScope serviceProviderEngine,
RuntimeResolverLock lockType)  at Microsoft.Extensions.
DependencyInjection.ServiceLookup.CallSiteRuntimeRe-
solver.VisitScopeCache(ServiceCallSite singletonCallSite,
RuntimeResolverContext context)  at Microsoft.Extensions.
DependencyInjection.ServiceLookup.CallSiteVisitor2.Vi-
sitCallSite(ServiceCallSite callSite, TArgument argument)
 atMicrosoft.Extensions.DependencyInjection.ServiceLookup.
CallSiteRuntimeResolver.VisitConstructor(ConstructorCall-
Site constructorCallSite, RuntimeResolverContext context)
      at Microsoft.Extensions.DependencyInjection.Service-
Lookup.CallSiteVisitor2.VisitCallSiteMain(ServiceCallSite
callSite, TArgument argument)  at Microsoft.Extensions.De-
pendencyInjection.ServiceLookup.CallSiteRuntimeResolver.
VisitCache(ServiceCallSite callSite, RuntimeResolverContext
context, ServiceProviderEngineScope serviceProviderEngine,
RuntimeResolverLock lockType)  at Microsoft.Extensions.
DependencyInjection.ServiceLookup.CallSiteRuntimeRe-
solver.VisitScopeCache(ServiceCallSite singletonCallSite,
RuntimeResolverContext context)  at Microsoft.Extensions.
DependencyInjection.ServiceLookup.CallSiteVisitor2.Vi-
sitCallSite(ServiceCallSite callSite, TArgument argument)
      at Microsoft.Extensions.DependencyInjection.Servi-
ceLookup.CallSiteRuntimeResolver.Resolve(Service-
CallSite callSite, ServiceProviderEngineScope scope)
      at Microsoft.Extensions.DependencyInjection.Servi-
ceLookup.DynamicServiceProviderEngine.<>c__Dis-
playClass1_0.b__0(ServiceProviderEngineScope scope)
      at Microsoft.Extensions.DependencyInjection.Service-
Lookup.ServiceProviderEngine.GetService(Type serviceType,
ServiceProviderEngineScope serviceProviderEngineScope)
      at Microsoft.Extensions.DependencyInjection.Service-
Lookup.ServiceProviderEngine.GetService(Type serviceType)
            at Microsoft.Extensions.DependencyInjec-
tion.ServiceProvider.GetService(Type serviceType)
            at Microsoft.Extensions.DependencyIn-
jection.ServiceProviderServiceExtensions.
GetService[T](IServiceProvider provider)
            at Azuretraining.Startup.ConfigureServices(I-
ServiceCollection services) in /Users/msan-
chez/Projects/Azuretraining/Startup.cs:line 34
            at System.RuntimeMethodHandle.InvokeMe-
thod(Object target, Object[] arguments, Signatu-
re sig, Boolean constructor, Boolean wrapExceptions)
            at System.Reflection.RuntimeMethodInfo.In-
voke(Object obj, BindingFlags invokeAttr, Bin-
der binder, Object[] parameters, CultureInfo culture)
      at Microsoft.AspNetCore.Hosting.ConfigureServicesBuil-
der.InvokeCore(Object instance, IServiceCollection services)
            at Microsoft.AspNetCore.Hosting.Configure-
ServicesBuilder.<>c__DisplayClass9_0.g__Star-
tup|0(IServiceCollection serviceCollection)
      at Microsoft.AspNetCore.Hosting.ConfigureServicesBuil-
der.Invoke(Object instance, IServiceCollection services)
      at Microsoft.AspNetCore.Hosting.ConfigureServicesBuil-
der.<>c__DisplayClass8_0.b__0(IServiceCollection services)
            at Microsoft.AspNetCore.Hosting.GenericWe-
bHostBuilder.UseStartup(Type startupType, Host-
BuilderContext context, IServiceCollection services)
            at Microsoft.AspNetCore.Hosting.GenericWe-
bHostBuilder.<>c__DisplayClass12_0.b__0(Host-
BuilderContext context, IServiceCollection services)
            at Microsoft.Extensions.Hosting.Host-
Builder.CreateServiceProvider()
      at Microsoft.Extensions.Hosting.HostBuilder.Build()
 at Azuretraining.Program.Main(String[] args) in /Users/msan-
chez/Projects/Azuretraining/Program.cs:line 19
```

Este error nos dará si intentamos ejecutar nuestro contenedor de docker, para solventarlo deberemos de aplicar un «work around» que nos permita poder trabajar sin problemas y a la vez que subimos el código a cualquier repositorio de código no tengamos que mostrar nuestras cadenas de conexión o información sensible. Para ello lo que vamos a hacer es añadir un nuevo fichero llamado docker-compose.yml con la siguiente composición:

```
version: "3.7"

networks:
  azuretraining.services.network:
    driver: bridge

services:
  azuretraining.services.courses:
    container_name: Azuretraining.Services
    build:
     context: ../
     dockerfile: ./Azuretraining.Dockerfile
    ports:
```

```
    - "8001:80"
  networks:
    - azuretraining.services.network
  volumes:
    - ~/.azure:/root/.azure
  environment:
        - KEYVAULT_ENDPOINT=https://NOMBREDENUESTROKEYVAULT.vault.azure.net/
```

En nuestro docker-compose hemos definido la estructura de ejecución de nuestro servicio, en este caso solo tenemos un contenedor, donde le indicamos el network, puerto, nombre del contenedor, etc. En este caso lo más importante son las propiedades volumes y environment. En el environment agregaremos nuestra url del Azure KeyVault, y en volumes lo que vamos a hacer es crear un volumen compartido donde copiaremos nuestra carpeta local de Azure para que podamos hacer sin ningún problema login con Azure CLI. Lo más importante es que, aunque esta carpeta se suba no compromete nuestra seguridad pues no tiene nada vinculante.

"En nuestro docker-compose hemos definido la estructura de ejecución de nuestros servicio, en este caso solo tenemos un contenedor, donde le indicamos el network, puerto, nombre del contenedor, etc"

Ahora modificaremos nuestro fichero .dockerfile para incluirle el Azure CLI y que podamos consumir la conexión hacia nuestro Azure KeyVault desde nuestro contenedor:

```
# https://hub.docker.com/_/microsoft-dotnet-core
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# copy everything else and build app
COPY . ./
RUN dotnet publish -c release -o out --no-restore

# final stage/image
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1

# install azure cli
ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update \
  && apt-get -y install --no-install-recommends apt-utils dialog 2>&1 \
  #
  # Verify git, process tools, lsb-release (common in install instructions for CLIs) installed
    && apt-get -y install git openssh-client iproute2 procps apt-transport-https gnupg2 curl lsb-release \
    && echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ $(lsb_release -cs) main" > /etc/apt/sources.list.d/azure-cli.list \
  && curl -sL https://packages.microsoft.com/keys/microsoft.asc | apt-key add - 2>/dev/null \
  && apt-get update \
  && apt-get install -y azure-cli;


WORKDIR /app
COPY --from=build /app/out .
ENTRYPOINT ["dotnet", "azuretraining.dll"]
```

Por último, solo nos queda lanzar el siguiente comando para ejecutar nuestra aplicación en local «dockerizada» y «securizada»:

```
docker-compose up
```

De esta forma tendremos nuestro proyecto completamente securizado pudiendo trabajar de forma fácil y sencilla, sin preocuparnos de que subamos información sensible a nuestro repositorio de código.

Conclusiones

Pienso que Azure KeyVault nos permite centralizar el almacenamiento de los secretos de aplicación. Esto nos permite controlar su distribución. Una de las grandes ventajas es que reduce en gran medida las posibilidades de que se puedan filtrar por accidente los secretos. Al no tener que almacenar información de seguridad en las aplicaciones elimina la necesidad de que esta información sea parte del código.

MANUEL SÁNCHEZ RODRÍGUEZ
Manuss20@gmail.com
@manuss20
https://manuss20.com

30

Usando el nuevo endpoint de Presencia en MS Graph API desde SPFx

El pasado mes de diciembre, el equipo de Microsoft Graph, anunció la preview de un nuevo Endpoint de Presencia, que como podéis imaginar, nos va a proporcionar la información de Presencia de uno o varios usuarios. Podéis ver el anuncio en el siguiente vinculo:

https://developer.microsoft.com/en-us/graph/blogs/microsoft-graph-presence-apis-are-now-available-in-public-preview/

"El nuevo Endpoint de Presencia nos va a proporcionar la información de Presencia de uno o varios usuarios"

Con este nuevo endpoint, podemos mostrar información de presencia en otras aplicaciones de negocio, o, como veremos en el ejemplo de este artículo, en un sitio de SharePoint, usando un WebPart desarrollado con el SharePoint framework (SPFx). Podéis ver el resultado de esto en la siguiente imagen:

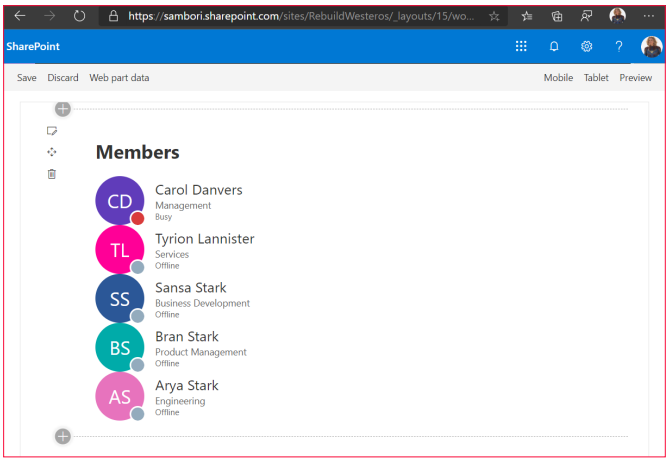


Imagen 1.- Ejemplo de uso del EndPoint de presencia.

Llamando el Endpoint de Presencia desde Postman

Antes de entrar en detalle con el código de nuestro WebPart, vamos a ver cómo tenemos que invocar el endpoint de presencia, y qué tipo de información nos va a retomar. Para ello, vamos a utilizar Postman.

Existen 2 endpoint de Presencia, el primero de ellos, nos va a dar la información de presencia de un usuario concreto, o nuestro propio usuario:

```
HTTP

GET /me/presence
GET /users/{id}/presence
```

El segundo, seguramente el que más utilizaremos en nuestras aplicaciones, nos va a permitir sacar la información de presencia de un listado de usuarios.

```
HTTP

POST /communications/getPresencesByUserId
```

Para usar Postman con la MS Graph API, os recomiendo que utilicéis la colección de postman que el equipo de producto mantiene en el siguiente repositorio de GitHub:

https://github.com/microsoftgraph/microsoftgraph-postman-collections

Una de las peticiones de dicha colección, nos permitirá sacar un AccessToken para poder llamar a los diferentes endpoints, en nuestro caso los de presencia. En el readme del proyecto, tenéis como podéis configurar el entorno de Postman, para poder obtener un AccessToken para vuestra Tenant de Office 365. Básicamente, necesitaréis registrar una App en vuestro Azure Active Directory, crear un Secret, y dar permisos a la app para el endpoint de presencia:

API / Permissions name	Type	Description	Admin Consent Requir...	Status
Microsoft Graph (3)				
Presence.Read	Delegated	Read user's presence information	Yes	Granted for
Presence.Read.All	Delegated	Read presence information of all users in you...	Yes	Granted for
User.Read	Delegated	Sign in and read user profile	-	Granted for

Una vez obtenido el AccessToken, ya podemos llamar al endpoint de presencia.

"La respuesta con la información de presencia consiste en el Identificador del usuario, y dos campos más: Availability y Activity"

Request / Response Información de Presencia del usuario logado

Para obtener la información de presencia del usuario logado, haremos:

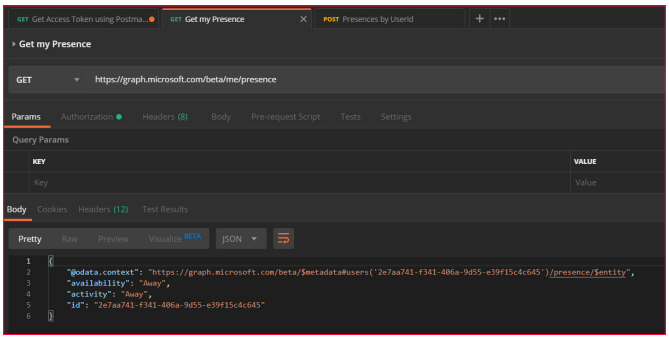


Imagen 2.- Como obtener la información de presencia del usuario logado.

Como podéis ver en la imagen anterior, la respuesta con la información de presencia consiste en el Identificador del usuario, y dos campos más:

- Availability: este es el típico: Away, Busy, etc. Según la documentación, los posibles valores a día de hoy son: Available, AvailableIdle, Away, BeRightBack, Busy, BusyIdle, DoNotDisturb, Offline, PresenceUnknown.
- Activity: Información adicional a la availability del usuario. Posibles valores: Available, Away, BeRightBack,Busy, DoNotDisturb, InACall, InAConferenceCall, Inactive,InAMeeting, Offline, OffWork,OutOfOffice, PresenceUnknown,Presenting, UrgentInterruptionsOnly.

Request / Response Información de Presencia varios usuarios

Para obtener la información de presencia de un listado de usuarios, debemos hacer un POST al endpoint /beta/communications/getPresencesByUserId, y enviar en el body la lista con los IDs de los usuarios.

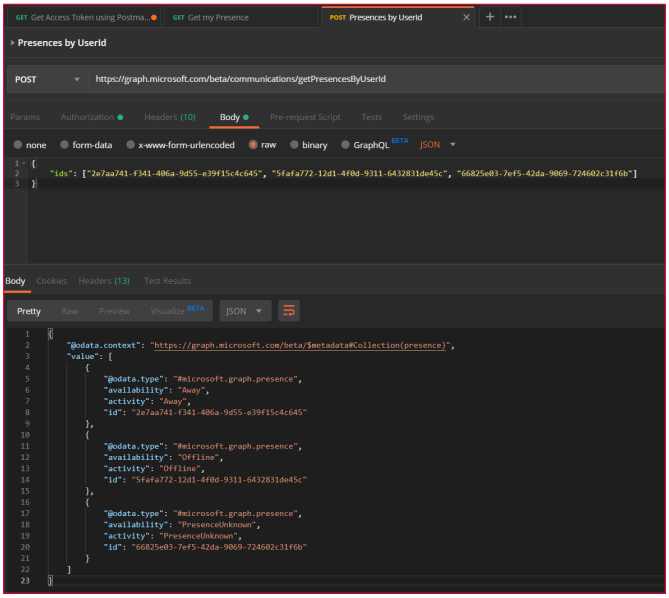


Imagen 3.- Respuesta con la información de presencia de varios usuarios.

Si nos fijamos en la respuesta, vemos que obtenemos la misma información que con el endpoint anterior, pero para cada uno de los usuarios solicitados.

Nota: si en el listado de usuarios incluimos algún usuario que no existe en el directorio, lo que obtenemos como información de presencia es la availability y activity con valor "PresenceUnknown".

Invocando al endpoint getPresencesByUserId desde un WebPart SPFx

Ahora que ya tenemos claro el tipo de petición que debemos hacer para obtener la presencia de varios usuarios, y el tipo de información devuelta que tendremos, ya podemos llamar al endpoint desde SPFx, y mostrar sus resultados.

Para ello, al ser endpoint de MS Graph API, haremos uso del propio MSGraphClient proporcionado por el framework SPFx. Como lo que queremos es sacar la presencia de todos los usuarios que forman parte de un site de SharePoint, primero necesitamos sacar los usuarios de dicho site, también llamando a Graph.

Antes de nada, en el OnInit del WebPart, utilizaremos el objeto msGraphClientFactory para obtener el MSGraphClient

```
protected onInit(): Promise<void> {
  return new Promise((resolve, reject) => {
    this.context.msGraphClientFactory.getClient().then(client => {
      this._graphHttpClient = client;
      resolve();
    }).catch(error => {
      console.log(error);
      reject(error);
    });
  });
}
```

Dicho cliente será pasado al componente de React, que hará todo el trabajo.

Primero obtendremos los miembros del site con el siguiente método:

```
private async _getMembers(): Promise<Dictionary<IPerson>> {
  const endpoint: string = `https://graph.microsoft.com/beta/groups/${this.props.groupId}/members?$select=id,displayName,department`;
  const response: any = await this.props.graphHttpClient.api(endpoint).get();
  const graphResponse: any = response.value;

  let peopleDictionary: Dictionary<IPerson> = {};
  graphResponse.map(user => {
    const person: IPerson = {
      id: user.id, displayName: user.displayName, department: user.department
    };
    peopleDictionary[person.id.toString()] = person;
  });

  return peopleDictionary;
}
```

Básicamente llamamos a Graph, y el resultado lo metemos en un diccionario, para posteriormente, actualizar cada usuario del diccionario con su información de presencia.

Con cada usuario, componemos un array de IDs, y llamamos el endpoint para sacar su información de presencia:

```
private async _getMembersPresense(users: string[]): Promise<Dictionary<IPerson>> {
  const endpoint: string = `https://graph.microsoft.com/beta/communications/getPresencesByUserId`;
  const response: any = await this.props.graphHttpClient.api(endpoint).post({
    "ids": users
  });
  const graphResponse: any = response.value;

  let peopleDictionary: Dictionary<IPerson> = {};
  graphResponse.map(user => {
    const person: IPerson = {
      id: user.id, displayName: '', department: '', availability: user.availability, activity: user.activity
    };
    peopleDictionary[person.id.toString()] = person;
  });

  return peopleDictionary;
}
```

Finalmente, en el render del componente React, utilizamos el control Persona del Office fabric UI (https://developer.microsoft.com/en-us/fabric#/controls/web/persona), que nos permite renderizar detalle de usuarios, y que además ya tiene algo para poder sacar la típica bolita con la availabili-

ty del usuario sobre su avatar:

```
members = membersArray.map(member => {
  return (
    <Persona
      size={PersonaSize.size72}
      text={member.displayName}
      secondaryText={member.department}
      tertiaryText={member.activity}
      presence={this._fromPresenceAvailabili ToPersonaPresence(member.availability)} />);
});
}

return (
  <Stack>
    <h1>Membe
    {members}
  </Stack>
);
}
```


Finalmente, la función _fromPresenceAvailabilityToPersonaPresence, hace un simple mapeo entre la información de availability que viene de la API, y las opciones que ofrece el componente Persona (que a día de hoy no tienen un mapeo uno a uno)

"Básicamente llamamos a Graph, y el resultado lo metemos en un diccionario, para posteriormente, actualizar cada usuario del diccionario con su información de presencia"

```
private _fromPresenceAvailabilityToPersonaPresence(availability: string): PersonaPresence {
  switch (availability) {
    case 'Busy':
    case 'BusyIdle':
      return PersonaPresence.busy;

    case 'Available':
    case 'AvailableIdle':
      return PersonaPresence.online;

    case 'Away':
    case 'BeRightBack':
      return PersonaPresence.away;

    case 'Offline':
      return PersonaPresence.offline;

    case 'DoNotDisturb':
      return PersonaPresence.dnd;

    default:
      return PersonaPresence.none;
  }
}
```

Tenéis todo el ejemplo completo en el repositorio de GitHub del PnP

https://github.com/SharePoint/sp-dev-fx-WebParts/tree/master/samples/react-members-with-presence

¡Hasta el próximo artículo!

LUIS MAÑEZ
Cloud Architect en ClearPeople LTD
@luismanez
https://medium.com/inherits-cloud

33

Entrevista Siderys

Nos especializamos en la adopción y desarrollo de soluciones sobre SharePoint, Project y Office 365. Desde hace más de 2 años trabajamos en proyectos de transformación digital y colaboración dentro de organizaciones locales e internacionales. Desde Montevideo, Uruguay, hemos ejecutado decenas de proyectos en varias partes del mundo y siempre con el mismo objetivo, ayudar a nuestros clientes a mejorar sus procesos día a día.

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

Nacimos como una pequeña fábrica de desarrollo de software en el año 2007, vendíamos nuestras habilidades de programación a otras firmas consultoras y en el interín buscábamos especializarnos en alguna tecnología. Después de varios años de trabajo decidimos especializarnos en SharePoint dado que veníamos ejecutando varios proyectos y veíamos que la plataforma crecía ano tras año.

¿Cuáles son las principales actividades tecnológicas hoy en día?

En Siderys desarrollamos proyectos a medida para nuestros clientes, proporcionamos servicios de adopción y transformación digital dentro de una organización. Buscamos que nuestros clientes rápidamente puedan obtener valor de las herramientas y plataformas en la nube que hoy hay a disposición de ellos, logrando transformar procesos.

¿Cuáles son las principales actividades no tecnológicas hoy en día?

Buscamos fomentar la integración interna, si bien somos una consultora con pocas personas, dos veces al año realizamos alguna actividad que nos permita compartir algunos momentos de distracción. Comúnmente se hace un asado (carne a la parrilla), jugamos a las cartas y charlamos.

¿Cuáles son las actividades que realiza en la comunidad técnica?

Uno de nuestros directores trabaja activamente en las comunidades tecnológicas y actividades que buscan distribuir el conocimiento tecnológico. Desde los primeros años de la empresa, Fabián Imaz ya se encontraba inmerso en organización de eventos, charlas y demás actividades que pudieran crear un espacio para compartir y conectar a las personas.

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

La tecnología está creciendo cada año más rápido. Vemos bits en todos lados y en todos los dispositivos, esto está haciendo que las personas estén más conectadas, informadas e incluso realicen sus labores diarias más rápido de que lo que lo hacían años atrás. Este es el camino hacia donde vamos, los bits como una herramienta para mejorar continuamente, ayudando a personas y empresas a ser cada día más eficientes. No creemos que exista un límite, creemos que hay muchas cosas todavía por ser inventadas/creadas y la tecnología será la base de todas ellas.

34

Logic Apps y HTTP Actions, ¿Puedo hacer llamadas Async?

Primero debemos analizar el problema de Http Time Out, Logic App tiene más potencia de la que parece. Es de todos sabido, que Azure Logic App es una herramienta potente, que no se ciñe solo a procesos de integración, sino que podemos usarlo como verdaderos orquestadores, que complementen de una forma óptima a nuestras queridas Azure Functions.

Pero en la práctica, los desarrolladores optan siempre por ir a Azure Functions o Durable Functions, porque parece que los flujos se complican demasiado o bien porque no se sienten cómodos. Nos llevaría muchas líneas demostrar toda la potencia de Logic App en realidad, pero vamos a intentar ver al menos uno de los problemas posiblemente más bloqueantes para mucha gente, y es ¿Qué hago cuando Logic App llama a procesos muy pesados por HTTP y que exceden los 2 minutos límite de tiempo? ¿Qué hago con las llamadas async por HTTP en Logic App, como las controlo?

Llamadas a procesos pesados y llamadas Async, patrón Pollling

En el caso general, hacer una llamada a una API externa o a otro sistema desde Logic App no supone ningún problema, simplemente debemos hacer uso de la action HTTP y obtenemos el resultado de la petición.

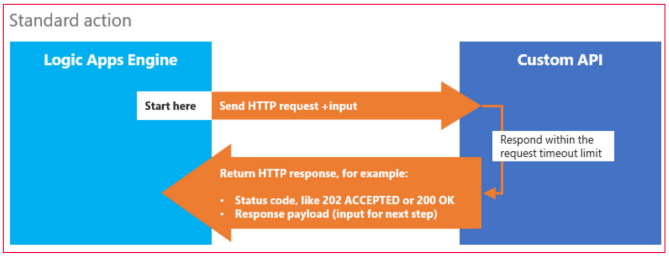


Imagen 1.- Standar action Logic App Http.

Pero claro, ¿Qué sucede si la petición tarda más de 2 minutos en responder? Si acudimos a limitaciones de Logic App, expuestas por Microsoft, nos queda muy claro que una petición de salida Http no puede superar nunca los 120 segundos, y por ello debemos bien implementar un patron “async polling pattern” o bien anidar otras LogicAapp, con el coste y la complejidad que esto conlleva.

Pues bien estamos ante el primer gran motivo para irnos a Azure Function, pero claro si estas leyendo esto seguro que eres una persona con mucha inquietud, y te preguntarás ¿Cómo es posible algo tan sencillo no se pueda resolver con Logic App y además si una Logic App tiene una duración máxima de 90 días?

Pues si se puede resolver, y como hemos ido anticipando la primera gran opción es implementar un flujo que siga el patrón Polling que vemos en la siguiente imagen:

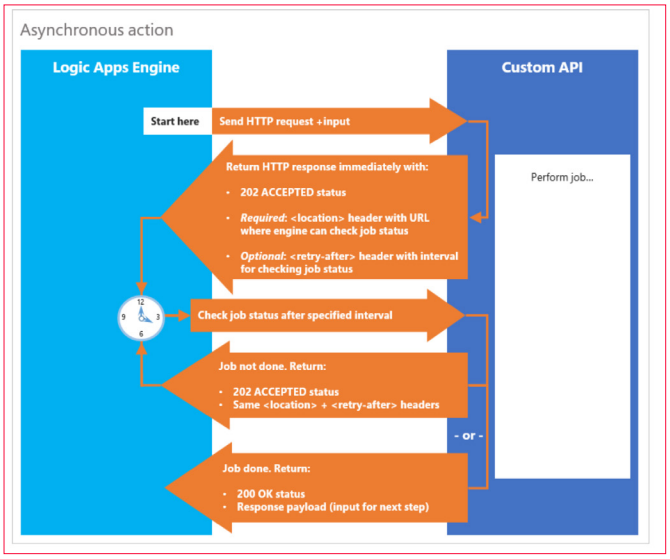


Imagen 2.- Polling pattern.

Antes de entrar a “destripar” el diagrama anterior, debemos pensar en el escenario ideal para poner en practica este ejemplo. Por ejemplo, pensemos en un sistema de “Compra online”, en la cual hay un servicio que aprovisiona la compra, pero que tarda varios minutos, incluso horas en tramitar nuestro pedido. Evidentemente aquí ya no es un problema de Async o no, es un problema de que el proceso en background puede tardar no se pongamos 15 minutos, tiempo suficiente para despreciar por ejemplo una Azure Function con un plan de consumo básico.

"Azure Logic App es una herramienta potente, que no se ciñe solo a procesos de integración, sino que podemos usarlo como verdaderos orquestadores, que complementen de una forma óptima a nuestras queridas Azure Functions"

Siguiendo ahora sí, el diagrama anterior, nuestra Logic App va a realizar una petición HTTP al sistema de Compra, y

este le va a devolver una cabecera de respuesta 202 confirmando que se ha tramitado la petición, e incluso un tiempo o intervalo de tiempo estimado de respuesta al pedido.

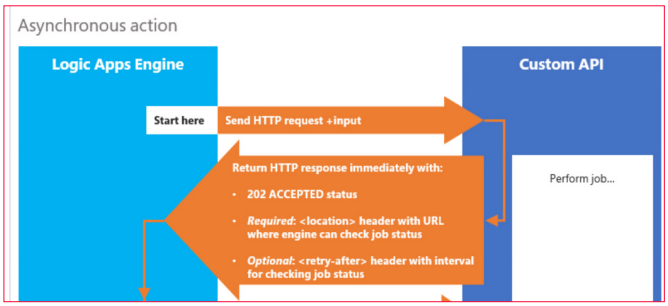


Imagen 3.- Http 202 accepted.

A partir de aquí en la Logic App, deberemos implementar un "check status", que podemos hacer con un do until y un timer para ir ganando tiempo en el proceso. Recordar que tenemos hasta 90 días por flujo, y si o si queremos capturar que el pedido es correcto, porque necesitamos anotar esta transacción en nuestro sistema.

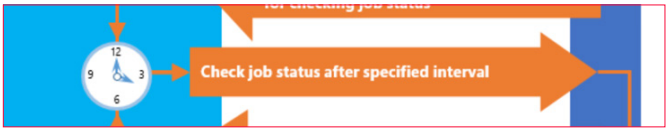


Imagen 4.- Polling timer.

Pongamos que hemos configurado el intervalo para que cada 5 minutos, se evalué la petición, nuestra Logic App hará peticiones HTTP al sistema de Compras preguntando por ejemplo con un Identificador de pedido, para comprobar si ha concluido o no la transacción, en este caso podemos obtener dos posibles resultados:

- Trabajo pendiente con un 202.
- Trabajo finalizado con 200 y un cuerpo de respuesta.

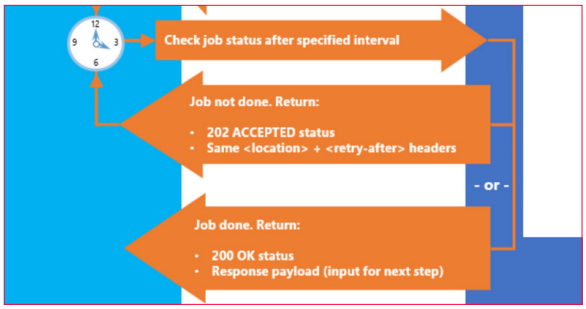


Imagen 5.- Return 200 ok.

"con un simple timer y un poco de ingenio ya no solo podemos soportar peticiones asíncronas que igual para eso es mejor ir por código y Azure Function"

Como veis, con un simple timer y un poco de ingenio ya no solo podemos soportar peticiones asíncronas que igual para eso es mejor ir por código y Azure Function, sino que podemos implementar procesos duraderos en el tiempo, que se queden esperando y reciban lo antes posible la transacción, pudiendo anotar en todos los sistemas necesarios la operación y dejarlos así sincronizados. He de adelantar que este diagrama habla a muy bajo nivel de como se implementa un patrón Polling, que no es más que un sistema de "check status" cada cierto tiempo, mediante un temporizador, como veremos en el siguiente ejemplo, las Acciones HTTP de Logic App y de Response ya tienen implementado este patrón de base y podemos hacer uso de el sin tener que implementarlo a mano, aunque es bueno saberlo si tenemos un sistema que no acepta este patrón.

Ejemplo, Logic App con respuesta Async y Logic App con

patrón Polling

Vamos a simular el patrón Polling anteriormente definido, con dos Logic App, de forma que una Logic App va a devolver una respuesta de forma Async, y tendrá activado el patron polling de respuesta y que desde ahora será "Response Worflow", y una segunda Logic App que va a replicar exactamente el comportamiento del temporizador que hemos visto anterior mente y que vamos a llamar "Request Polling Workflow".

Implementar Response Workflow

Vamos a implementar un proceso que permita simular un proceso de más de 2 minutos, y provocar así que se implemente un patron polling en los procesos que lo instancien o no podrán obtener el dato de salida.

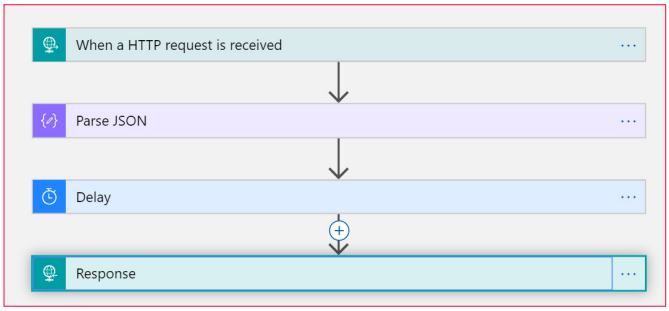


Imagen 6.- Response workflow.

Este workflow contará de cuatro pasos, que serán los siguientes:

1.– Desencadenador por Http:

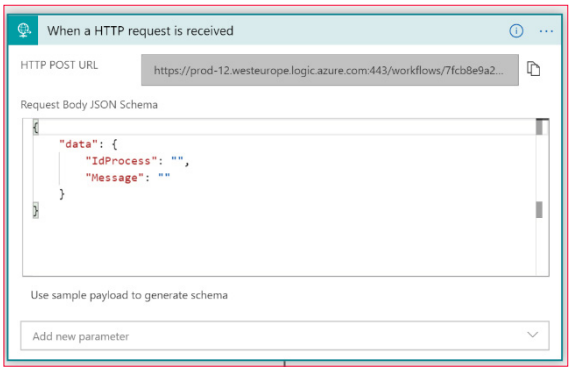


Imagen 7.- Http Triger Logic App.

2.– Mapeo de body de entrada, a un JSON entendible por Logic App con "Parse Json":

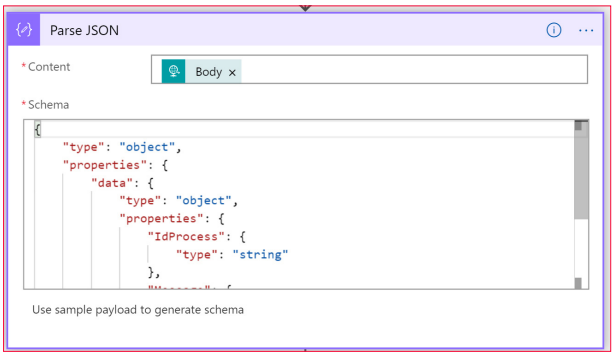


Imagen 8.- Parse JSON Logic App.

3.– Delay de 3 minutos, para simular un time out en los procesos que consuman a este proceso.

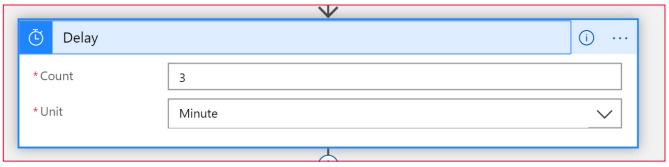


Imagen 9.- Timer.

4.– Acción de respuesta que tenga activa el modo Async para poder implementar el patron polling.

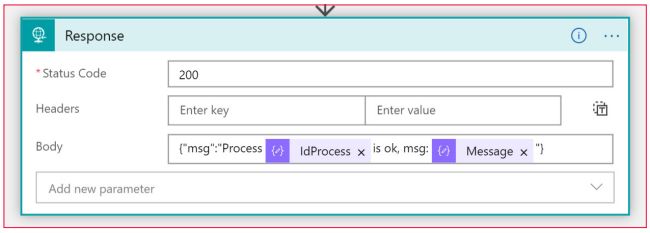


Imagen 10.- Response – Action.

Esta acción responderá un 200 ok pasados los 3 minutos, con el IDProceso de entrada y el mensaje obtenido en la petición. Para poder configurar la salida como asíncrona deberemos pulsar en las propiedades de la respuesta y marcar la opción "asynchronus response", para que devuelva en el acto un 202. Junto con el 202 Accepted, el proceso va a devolver una URL en una cabecera llamada "Location", que va a permitir al sistema que ha realizado la petición obtener el resultado final del proceso pasados los 3 minutos de delay.

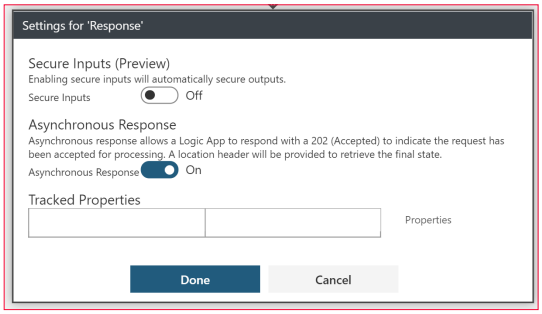


Imagen 11.- Async process.

Para comprobar que la Logic App, va a permitir preguntar a cualquier sistema por el "status" del proceso, podemos realizar una petición vía Postman a la Logic App que acabamos de crear. En la caja de descadenador por HttpTrigger, podemos copiar el campo "HttpPost Url". Debemos crear una petición post en Postman tal cual se ve en la siguiente imagen:

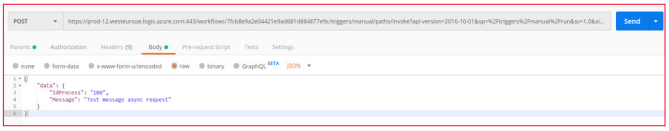


Imagen 12.- Testing Postman Workflow.

Si todo va como debe nos deberá devolver un 202 Accepted, y una serie de cabeceras entre la que tenemos la cabecera Location, que contiene la URL para hacer seguimiento del proceso.

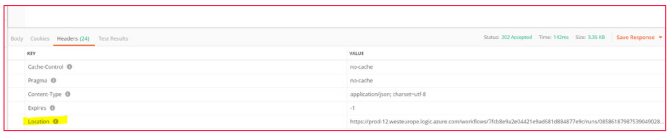


Imagen 13.- Location Header.

Pasados los 3 minutos de duración del Timer de nuestra Logic App, podemos hacer una petición GET a la URL que contenía la cabecera Location, y obtendremos el resultado que esperábamos, devolviendo el id de proceso 100, y el mensaje de Test que le pasamos al inicio de la prueba.

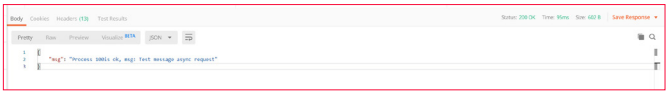


Imagen 14.- 200 Response.

Implementar Request Polling Workflow

Una vez tenemos un sistema que acepta llamadas asíncronas, vamos a implementar este patron polling en una Logic App, para simular esa espera de más de 2 minutos y solventar así el problema que tenemos con los procesos pesados. Debemos recordar que una Logic App tiene una duración máxima de 90 días por lo que es el límite que nos ofrece el servicio para implementar estos procesos. En la definición del proceso hablamos de crear un timer que evaluara el proceso asíncrono para ver si había terminado o no la petición, cada cierto tiempo. He de decir que lo podemos hacer, y es un proceso 100% optimo, pero también es un patrón que ya nos implementa la acción HTTP de Logic App, y que se encarga de ir consultando por la cabecera Location cada cierto tiempo por nosotros.

Vamos a crear un proceso con HttpTrigger, sencillo que reciba el mismo json en el body que en el caso anterior, así se lo enviaremos a la Logic App que creamos antes.

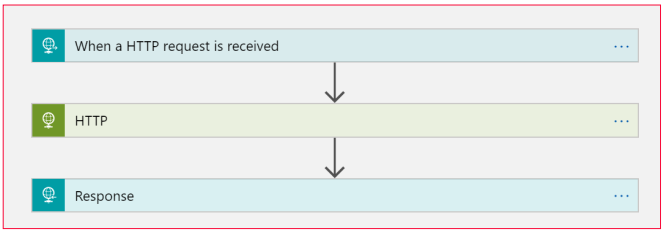


Imagen 15.- Request polling workflow.

Si vemos el workflow con perspectiva no hace más que hacer una petición a la Logic App anterior, que devuelve

un 202 en la primera petición, y que mediante el patrón polling de las llamadas asíncronas se va a esperar a que termine el otro workflow para devolver el resultado.

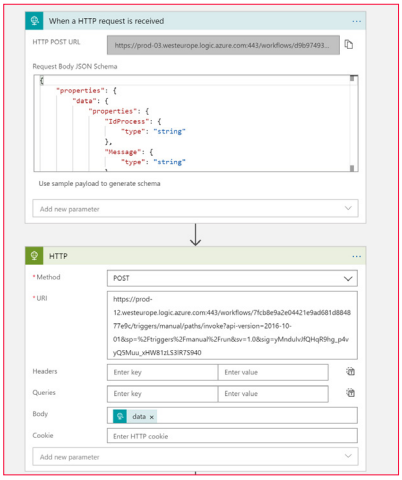


Imagen 16.- Http Polling action.

Debemos recordar que la petición Http a nuestra Logic App Response Workflow tiene un retardo de 3 minutos, por eso debemos indicarle en las propiedades que cumple un patrón asíncrono. Si no es así no esperará a que termine el proceso.

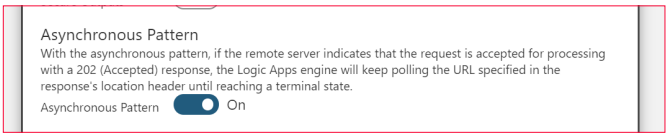


Imagen 17.- Settings Http Action.

El resultado de respuesta podemos devolver por ejemplo lo que de vuelve nuestro Response Workflow, en este nuevo Polling Workflow y así comprobamos que la propia Logic App simula nuestra petición a la cabecera Location, y por tanto nos podemos olvidar de implementar a mano el patrón. Si lanzamos la nueva Logic App podremos ver que se queda esperando el tiempo necesario haciendo polling sobre la segunda Logic App, en este caso algo más de 4 minutos.

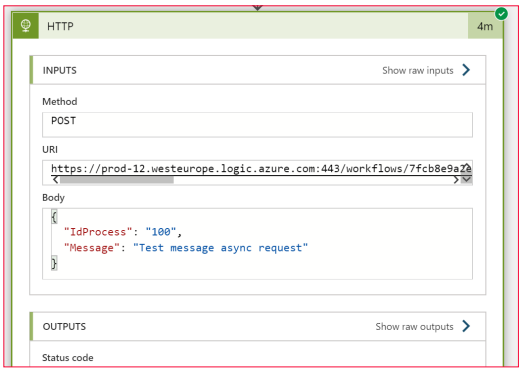


Imagen 18.- Resultado del polling.

Si por ejemplo tuviéramos que el proceso pesado no es una Logic App, si no que debe ser un API Rest o una Functions, podemos tener dos opciones:

- Simular el patrón Pulling, desde Logic App mediante temporizadores y do until (poco recomendable porque puede ser foco de control de errores, más coste y peor rendimiento que usando el patrón que nos aporta HTTP Action) Implementar nuestra Function o API con este patrón, tal y como refleja en este enlace (https://docs.microsoft.com/es-es/archive/blogs/logicapps/long-running-tasks-in-logic-apps). Al final, es controlar un ID de proceso que llevaremos mediante la cabecera Location ,que sí entiende nuestra Logic App y ofrecer los métodos de "check status" necesarios, para poder obtener el resultado final del proceso.

Conexiones Webhook,una forma más refinada de controlar las llamadas Async,

La verdad que debemos de admitir que el patrón Polling está muy evolucionado en las HTTP Actions, pero en mi opinión me sigo quedando con el patron Webhook para realizar estas llamadas Async, incluso procesos que requieran muchas pausas como puede ser construir una máquina de estados, que requiere de eventos externos para ir cambiando el estado del proceso.

"Debemos recordar que una Logic App tiene una duración máxima de 90 días por lo que es el límite que nos ofrece el servicio para implementar estos procesos"

Si bien para un sistema por ejemplo de Pedido de nuestra tienda Online, nos puede servir de sobra el patrón polling, imaginaros ahora un proceso que tenga 3, 4 cuatro pasos encadenados como puede ser el seguimiento de un pedido desde que se compra hasta que se envía en la vivienda. Para esto necesitamos ahora sí:

- Varios eventos por proceso (uno por cada cambio de estado).
- Controlar el callback de cada estado, ¿Puedo pasar al siguiente estado?

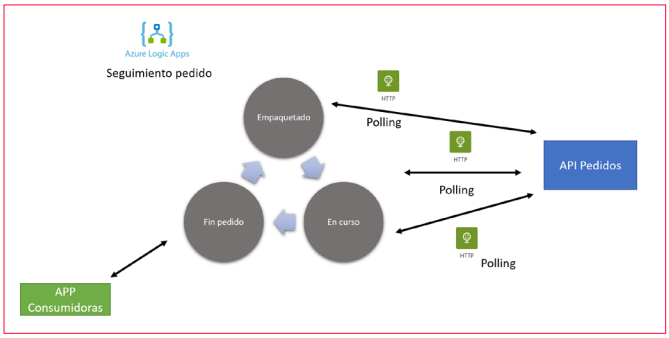


Imagen 19.-Flujo de seguimiento de pedido.

Para mi este escenario ya se va de las manos al patrón Polling y las acciones HTTP, porque al final si vamos a tener varias HTTP Actions, haciendo polling al sistema, estamos sobrecargando al sistema de seguimiento del pedido. Imaginaros que tenemos "100 pedidos lanzados", no dejamos de estar haciendo 100 peticiones concurrentes por cada

estado, o lo que es lo mismo si el proceso es largo en este caso ya de días, vamos a estar sobrecargando el API de seguimiento sin motivo aparente.

Si nos vamos al ejemplo anterior, y vemos sobre el terreno, para un delay de 4 minutos, se le evalúa a la Logic App Response desde la Logic App Request Polling , un total de 4 veces sin necesidad o lo que es lo mismo 4 skipped Actions, que sin duda tiene un coste mínimo, pero un coste tanto en rendimiento con en la factura al fin de mes.

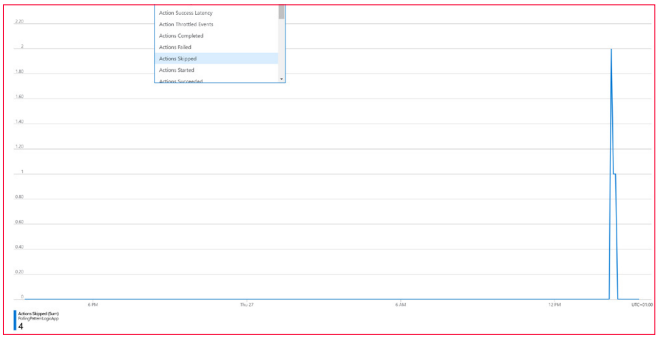


Imagen 20.- Métricas skipped action.

Si nos centramos en "vamos a intentar" reducir el número de peticiones no necesarias al sistema, podemos optar por implementar una suscripción Webhook.

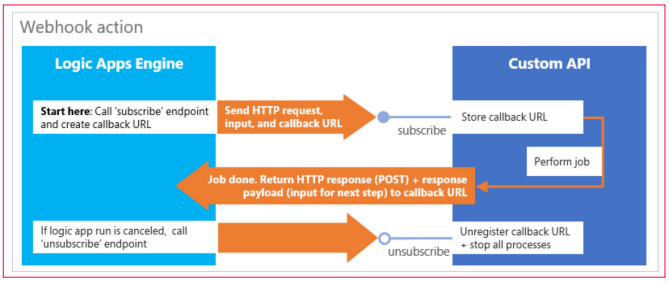


Imagen 21.- Patrón webhook.

A diferencia del patrón Polling, si suscribimos por Webhook una Action de Logic App, esta solo se va a invocar justo cuando el sistema haya terminado su proceso. Por ejemplo, para nuestro caso del pedido, solo se va a invocar la Acción Webhook, cuando el API de pedidos, quiera indicar a nuestra Logic App que debe cambiar de estado.

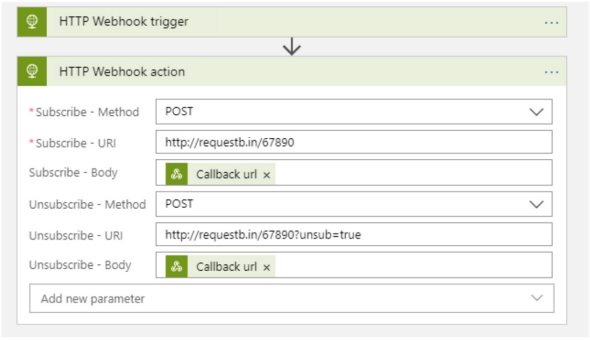


Imagen 22.- Acción de Webhook.

Nuestro API de pedidos debe admitir una suscripción vía webhook (podemos ver ejemplos en https://github.com/logicappsio/LogicAppTriggersExample/blob/master/LogicAppTriggers/Controllers/WebhookTriggerController.cs) de como montar nuestro Cliente Webhook en un Azure Functions o un App Service.

Cada vez que se produzca un evento al cual mi Logic App está suscrito, esta acción HTTP Webhook se disparará, y mientras eso no suceda el flujo seguirá en ejecución. Una vez se ejecute la acción de Webhook, esta se de-suscribe para este evento, y permite continuar al flujo, es importante configurar correctamente esta caja, para que pueda continuar el flujo de forma correcta.

Si volvemos a nuestro flujo anterior, ya no tenemos un polling para cada uno de los estados, sobrecargando al proceso, si no que el propio flujo con la acción webhook se suscribe en tiempo real al evento que le interese para ese estado, y una vez finalizado deja de escuchar ese estado.

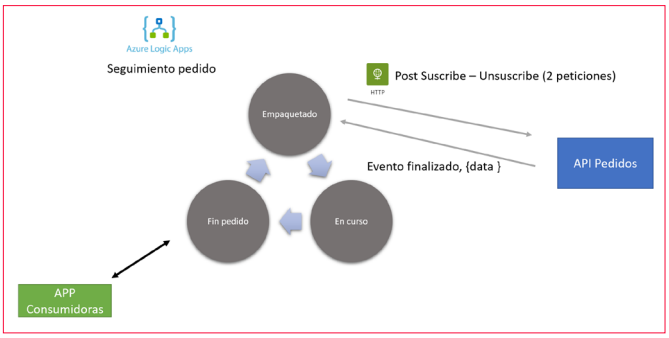


Imagen 23.- Seguimiento pedido con webhook.

Además, los eventos por estado los recibe la Logic App en tiempo real, y los proporciona el API de Pedidos, por lo que el API de pedidos solo recibe 2 peticiones por cada estado de un flujo la de suscribirse y la de desuscribirse al evento correcto.

Conclusiones, ¿Por qué nos rendimos tan pronto?

Este servicio es un "avión", es super potente, pero por mi experiencia con él, los equipos de trabajo y los arquitectos tienen a rendirse muy muy pronto y acuden a otras vías más "familiares". Estas derivan en soluciones a medida o un catálogo de funciones en Azure en Net Core, que sin duda son muy capaces pero igual no idóneas si las comparamos en algunos casos de orquestación o integración, donde Logic App puede tener mucho que hablar.

En este artículo solo hemos visto un poco de su poder, pero tiene mucho más, tiene un gran poder de integración entre sistemas, de autenticación, de orquestación..., que sin duda algún día iremos explotando, y en combinación con App Service o Azure Functions en función del tipo de aplicación que tengamos, podremos ir orquestando procesos muy muy solidos que además tienen un coste muy bajo comparado con programar workflows a medida, tanto en servicios de Azure como en horas de implementación.

Animo a que "valoremos" un poco más a este servicio que sin duda nos puede aportar mucho en muchos escenarios.

SERGIO HERNÁNDEZ MANCEBO
Azure MVP

Nosotros

Alberto Diaz

Alberto Diaz cuenta con más de 15 años de experiencia en la Industria IT, todos ellos trabajando con tecnologías Microsoft. Actualmente, es Chief Technology Innovation Officer en ENCAMINA, liderando el desarrollo de software con tecnología Microsoft, y miembro del equipo de Dirección.

Desde 2011 ha sido nombrado Microsoft MVP, reconocimiento que ha renovado por séptimo año consecutivo. Se define como un geek, amante de los smartphones y desarrollador. Fundador de TenerifeDev (**www.tenerifedev.com**), un grupo de usuarios de .NET en Tenerife, y coordinador de SUGES (Grupo de Usuarios de SharePoint de España, **www.suges.es**)

Email: adiazcan@hotmail.com
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.siderys.com
Email: fabiani@siderys.com.uy
Blogs: http://blog.siderys.com
Twitter: @fabianimaz

Gustavo Velez

Gustavo Velez es Ingeniero Mecánico y Electrónico; trabaja en la arquitectura, diseño e implementación de sistemas de IT basados en tecnologías de Microsoft, especialmente SharePoint, Office 365 y Azure.

Propietario del sitio especializado en información sobre SharePoint en español **http://www.gavd.net**, autor de ocho libros sobre SharePoint y sus tecnologías y numerosos articulos y conferencias sobre el tema.

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 14 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 Apps & Services y co-fundador del Grupo de Usuarios de SharePoint de España (SUGES, www.suges.es), delGrupo de Usuarios de Cloud Computing de España (CLOUDES) y de la Comunidad de Office 365. Hasta la fecha, ha publicado 11 libros sobre SharePoint & Office 365, así como varios artículos encastellano y en inglés sobre ambas plataformas.

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

Santiago Porras

Innovation Team Leader en ENCAMINA, lidera el desarrollo de productos mediante tecnologías Microsoft. Se declara un apasionado de la tecnología, destacando el desarrollo para dispositivos móviles y web, donde ya cuenta con 16 años de experiencia.

Microsoft MVP in Developer Technologies, colabora con las comunidades de desarrolladores desde su blog personal **http://blog.santiagoporras.com** y ocasionalmente en **CompartiMOSS.com**. Además, es uno de los coordinadores de TenerifeDev, grupo de usuarios de .NET en Tenerife (**http://www.tenerifedev.com**)

Sitio Web: http://www.santiagoporras.com
Email: santiagoporras@outlook.com
Blogs: http://blog.santiagoporras.com
Twitter: @saintwukong

¿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