Nº41 septiembre 2019

CompartiMOSS

REVISTA ESPECIALIZADA EN TECNOLOGÍAS MICROSOFT

Entrevista Juan Ignacio Oller

¡Alarma 1202!

Dataflows – Desde lo básico, y más allá

Un vistazo a Microsoft Graph Toolkit: Web Components con Graph

Contenido

Staff

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

DIRECCIÓN GENERAL

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

DISEÑO Y DIAGRAMACIÓN

- Santiago Porras Rodríguez

Contacte con nosotros

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

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

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

03

Editorial

Nuestro día a día como desarrolladores de software puede llegar a ser bastante complejo, no sólo nos ponemos a tirar líneas de código, si no que tenemos que añadir diversas piezas de un puzle que forma la aplicación. Piezas tan diversas como Microsoft Graph Toolkit, Azure Functions, React Hooks, etc. Nuestros autores lo saben, y es por esto por lo que nos escriben las pinceladas precisas y necesarias para conocer estos componentes y darles un buen uso en nuestras aplicaciones.

En este número, además de aprender nuevas piezas del puzle, publicamos un artículo sobre la computadora de guiado de las misiones Apolo, y como con tan poca potencia de cálculo fueron capaces de lanzar al espacio naves tripuladas. Haced el cálculo de la potencia que tenéis disponible en vuestros ordenares o en Azure y preguntaros, ¿podemos hacer software que cambie el mundo? Desde esta revista creemos que sí, y esperamos que nuestros artículos sirvan de inspiración para que desarrollemos software para un mundo mejor.

Gracias por formar parte de esta familia que escribe y/o lee nuestros artículos, que nos ayuda a difundir la tecnología relacionado con Microsoft para compartir el conocimiento con el objetivo de hacer mejor nuestro día a día.

El Equipo Editorial de CompartiMOSS

4

¡Alarma 1202!

Este artículo no trata de SharePoint, Office 365, ni nada de lo habitualmente expuesto en CompartiMOSS. Trata sobre la computadora de guiado de las misiones Apolo, dado que en Julio se cumplieron 50 años del primer alunizaje.

Dentro de la gran hazaña que supuso el programa Apolo en muchísimos ámbitos de la ciencia y tecnología, este artículo se centra en una pequeña parte: la computadora AGC (Apollo Guidance Computer).

Si eres de los que piensa que esto nunca ocurrió, que el programa espacial es un montón de mentiras y que estoy a sueldo de los reptilianos extraterrestres sección illuminati… pasa al siguiente artículo.

Ya que has seguido leyendo, vamos a ello: La función principal de la AGC era proporcionar el sistema de guiado y navegación al módulo de mando (CM) y al módulo de excursión lunar (LEM). El CM disponía de 2 computadoras y el LEM de una.

La AGC estaba conectada a varios sistemas de instrumentación y control como radares, el sistema de navegación inercial (otra gran innovación desarrollada en la época), receptores de telemetría o el control del motor de descenso, entre otros.

Adicionalmente, la tripulación disponía de un sistema manual de navegación basado en la observación directa de estrellas (un sextante de toda la vida), ya que la NASA consideró que un sistema novedoso como la AGC podría fallar durante la misión.

Un poco de hardwareporn sobre la AGC:

Frecuencia del oscilador: 2,048 MHz.
Memoria RAM: 2 KPalabras en memoria de núcleos magnéticos.
Memoria ROM: 36 KPalabras en memoria de ferrita.
Longitud de palabra: 16 bits.
Consumo: 55 W.
Peso: 31,8 Kg.
Dimensiones aprox: 61x32x17 cm

Desarrollo

El sistema de navegación fue de los primeros contratos que se adjudicaron en el programa Apolo, solamente dos meses después del inicio del programa. Tal era la importancia que la NASA daba a la navegación.

Se adjudicó al laboratorio de instrumentación del MIT en lugar de a una empresa privada, ya que había desarrollado en 1953 un sistema de navegación inercial capaz de guiar un avión en piloto automático.

Este sistema consistía en giroscopios y acelerómetros mecánicos muy precisos que eran capaces de mantener una plataforma de referencia fija y medir los cambios de posición del vehículo. Los datos obtenidos eran procesados por un sistema electromecánico para actuar sobre los mecanismos de vuelo.

> ***"La función principal de la AGC era proporcionar el sistema de guiado y navegación al módulo de mando (CM) y al módulo de excursión lunar (LEM). El CM disponía de 2 computadoras y el LEM de una."***

Pero, para un viaje de 1,5 millones de Km, se requeriría algo mucho más sofisticado: una computadora digital.

En los años 60 las computadoras eran grandes monstruos que ocupaban salas enteras y consumían cantidades ingentes de electricidad… nada adecuado para llevar a bordo de una nave espacial.

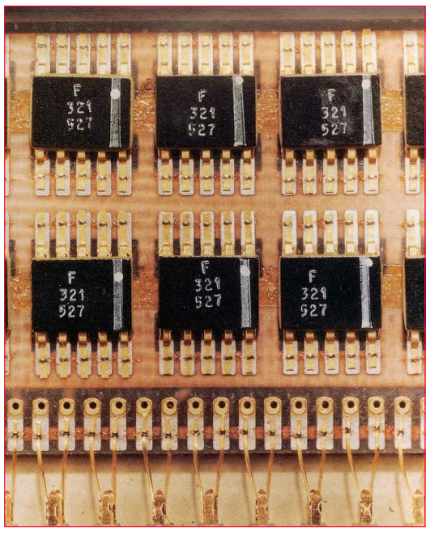


Imagen 1 - Circuitos Integrados de la AGC. Fuente: Wikipedia

Por ello el MIT recurrió a unos componentes novedosos: los circuitos integrados. Ligeros, de poco consumo y mucho más pequeños que las válvulas de vacío ofrecían las ventajas necesitadas… salvo por su confiabilidad. Al ser una tecnología en pañales se llevaron a cabo pruebas exhaustivas de cada lote para asegurar que soportarían las condiciones de vuelo.
La AGC fue la primera computadora construida con los novedosos circuitos integrados. Tal era la novedad que su construcción requirió el 60% de la producción total de circuitos integrados en EE.UU.

La AGC estaba formada por 2.800 de estos circuitos integrados, cada uno de ellos contenía 2 puertas NOR de 3 entradas. Los circuitos integrados eran interconectados mediante cableado para formar las unidades lógicas. Después de ser cableadas, se añadía un molde de epoxi para protegerlos de las vibraciones.

Otro problema que debieron afrontar fue la potencia de procesamiento requerida. No se disponía de la suficiente potencia de procesamiento para realizar todas las tareas necesarias por lo que se ideó otro novedoso sistema: Procesar las tareas de acuerdo con una prioridad asignando el tiempo de procesamiento de acuerdo con estas prioridades.

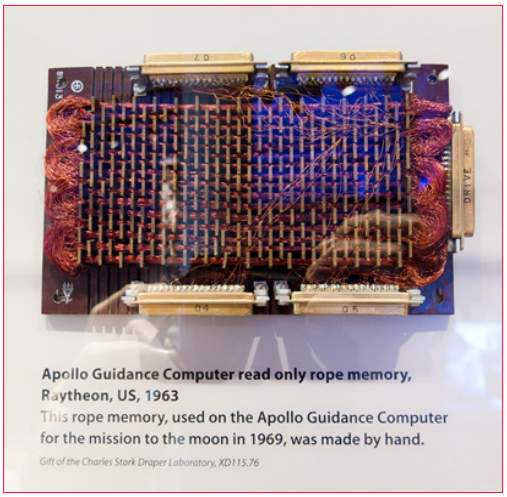


Imagen 2 - Módulo de memoria ROM de la AGC expuesto en el museo de historia de informática de S. Francisco

Una tarea de baja prioridad (por ejemplo, actualizar un display) podría no ejecutarse para permitir la ejecución de tareas más importantes (por ejemplo, mantener la posición de la nave).
La AGC permitía ejecutar hasta 8 tareas a la vez gestionadas por un rudimentario sistema operativo en tiempo real (Exec). Cada tarea debía devolver el control al Exec de forma periódica para que comprobara si había tareas pendientes y su prioridad.

La programación se realizaba en un lenguaje interpretado que proporcionaba pseudo instrucciones de aritmética trigonométrica, matricial y vectorial utilizadas para el cálculo de navegación, y era tan manual y lenta que podían pasar horas hasta tener el resultado impreso desde que se lanzaba el proceso de generación del programa.

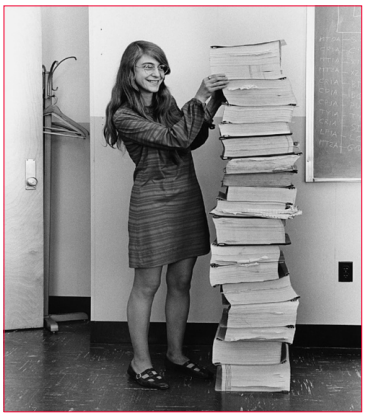


Imagen 3 - Margaret Hamilton, responsable de desarrollo de software de vuelo, con los programas de la AGC impresos. Fuente: Wikipedia

Los programas eran almacenados en memorias de ferritas. Estas memorias debían ser “cosidas” literalmente, haciendo pasar o no cables por los núcleos de ferritas para representar los ceros y los unos.
Esta tarea llevaba meses y era realizada por un grupo de mujeres tejedoras que fueron contratadas para este difícil trabajo.

Una vez cableado el programa y probado, el módulo se protegía con un recubrimiento de epoxi.

El interfaz de usuario

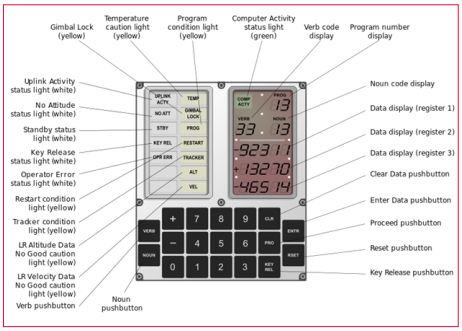


Imagen 4 - Consola DSKY de la AGC. Fuente: Wikipedia

Los astronautas utilizaban la AGC mediante un interfaz llamado DSKY (Display and Keyboard). Este interfaz disponía de un conjunto de luces indicadoras de estado, el teclado y un conjunto de displays de 7 segmentos.
Los comandos se introducían mediante una combinación de verbo y nombre. El verbo indicaba la acción a realizar y el nombre indicaba el dato afectado por la acción del verbo.

“La AGC fue la primera computadora construida con los novedosos circuitos integrados.”

Por ejemplo, presionando Verb 06 Noun 52 ENTR la AGC mostraba en los data display los datos de velocidad actual, velocidad de ascenso y altitud. La tripulación debía memorizar verbos y nombres para cada procedimiento de vuelo.

Las misiones Apolo

La AGC fue probada durante las misiones Apolo tripuladas 7, 8, 9 y 10 que fueron avanzando en las pruebas hacia el viaje final a la luna.

La misión Apolo 7 probó los sistemas en órbita terrestre.

La misión Apolo 8 realizó el primer vuelo alrededor de la luna perdiendo la conexión con el control de misión durante los 45 minutos de vuelo por la cara oculta de la luna.

La misión Apolo 9 probó todos los acoplamientos y desacoplamientos del CM y LEM en órbita terrestre.

La misión Apolo 10 realizó un ensayo de alunizaje viajando a la luna.

Durante estas misiones la AGC funcionó correctamente y entonces…

La alarma 1202

En la misión Apolo 11, la primera en alunizar el 20 de Julio de 1969, se produjo una alarma en la AGC durante el descenso a la superficie lunar del LEM. Esta fase de la misión era la más exigente para la AGC en cuanto a procesamiento.

El descenso consistía en tres fases, controladas por la AGC:

- Una primera fase de frenado desde la órbita lunar a 14 km de altura. Para ello se encendía el motor de descenso del LEM para disminuir la velocidad y la altura. En esta fase la posición del LEM respecto a la superficie era de una inclinación considerable y con el motor hacia delante para permitir el frenado.
- Una segunda fase de aproximación al lugar del alunizaje cuando se descendía a unos 2.100 metros. El LEM continuaba disminuyendo su velocidad y altura, pero rotaba para ir quedando en la posición en la que se alunizaba.
- La tercera fase, el LEM estaba a unos 150 metros en su descenso final a la superficie lunar, la velocidad horizontal respecto a la superficie era muy baja y el comandante tenía el control del motor para posar la nave.

En la misión Apolo 11, durante este descenso, Aldrin empezó a informar a Houston que se producía la alarma 1202 y 1201 en la AGC.

> ***"La AGC permitía ejecutar hasta 8 tareas a la vez gestionadas por un rudimentario sistema operativo en tiempo real (Exec)."***

En las simulaciones y el entrenamiento nunca se habían disparado estas alarmas, por lo que ni Armstrong ni Aldrin sabían que querían decir, aunque Aldrin se dio cuenta de que se producían al ejecutar el programa que calculaba la diferencia de altura entre la teórica calculada y la medida por el radar del LEM.

Desde Houston se dio el OK para continuar y finalmente el Águila alunizó ('The Eagle has landed', una de las frases famosas de esta misión).

¿Qué causó estos errores? Al consultar a los diseñadores descubrieron que eran alarmas por sobrecarga de procesamiento en la AGC.

Esta sobrecarga no se produjo por un mal diseño del hardware o el software si no por un problema con uno de los radares que provocó que, durante el descenso, la AGC recibiera datos erróneos de forma aleatoria sobre la posición de la antena de dicho radar causando un sobre procesamiento en el peor momento posible, de forma que la primera ley de Murphy se cumplió una vez más.

La era post Apolo

Después de las misiones Apolo se utilizó la AGC en programas experimentales de control de vuelo fly by wire en aviones a reacción y también fue usada para vehículos submarinos.

La AGC sirvió de predecesora para los sistemas de vuelo fly by wire, sus avances sentaron las bases de conceptos que hoy en día damos por sentados en un ordenador y supuso un salto enorme en las computadoras de la época. Todo ello dentro de un programa que permitió que 12 personas pisaran la luna tan solo 66 años después del primer vuelo a motor.

Enlaces de interés

- Emulador AGC OnLine: http://svtsim.com/moonjs/agc.html
- Enlace de Wikipedia: https://es.wikipedia.org/wiki/Apollo_Guidance_Computer
- Video del descenso del Apolo 11. La alarma 1202 de produce a partir del min 10:20 https://youtu.be/xc1SzgGhMKc
- Esquemas lógicos de la AGC: http://klabs.org/history/ech/agc_schematics
- Documental sobre la AGC: https://www.youtube.com/watch?v=tTaZjyMApKY
- Misión Apolo 11: https://www.nasa.gov/mission_pages/apollo/missions/apollo11.html

ALBERTO ESCOLA
Ingeniero de Sistemas y Cloud

07

Introducción a Azure Search

En este artículo se aprenden los conceptos básicos de Azure Search y el procedimiento para una configuración básica del servicio, desde el índice, indexador y fuente de datos, hasta la ejecución de consultas de prueba.

En el mundo de los desarrolladores .NET, casi en cualquier proyecto de desarrollo se tiene la obligación de implementar funcionalidades de búsqueda por algún requerimiento de los usuarios, desde los elementos sencillos como buscar documentos, registros de personas o de facturas, hasta otros tipos de contenido. Estas son funcionalidades básicas que el software desarrollado debe proveer a los usuarios.

Los desarrolladores .NET que están acostumbrados a utilizar Azure SQL o SQL Server como motor de base de datos, utilizan en su mayoría procedimientos almacenados, vistas o simples consultas para buscar registros en las tablas de la aplicación, sin embargo, si esta es más compleja y utiliza otras fuentes de datos como CosmosDB, Azure Storage para documentos de Office o algunas otras, entonces la búsqueda se vuelve un poco más compleja porque son de diferentes tipos.

Con Azure Search podremos buscar información en todas estas fuentes de datos con un API en común, independientemente de donde se halla indexado el contenido, si fue SQL Server, Cosmos DB, Blob Storage u otro, el API y el código implementado para realizar las búsquedas no tiene ninguna diferencia, esto presenta un beneficio para el desarrollador en termino de curvas de aprendizaje y en términos de productividad.

En este artículo se explican los conceptos necesarios para entender la arquitectura de Azure Search, al mismo tiempo se presenta un ejemplo de configuración de este, esto para mostrar los beneficios al utilizar este servicio. El texto está dividido en tres partes: en la primera se presenta los conceptos básicos para entender la arquitectura de Azure Search, en la segunda parte creamos y configuramos un ejemplo básico y se realiza la prueba del motor de búsqueda.

Parte 1. Azure Search: conceptos y procesos

Los proyectos en Azure Search tienen un proceso similar en todos los casos: se crea el servicio – a través del portal, de Azure CLI o Powershell-, se crea el índice, se cargan datos y se realizan búsquedas. El crear el servicio y configurarlo es sencillo después de entender los conceptos necesarios de acuerdo con la aplicación que se vaya a desarrollar.

El primer concepto es el índice, definido como una estructura eficiente para almacenar y organizar palabras clave de búsqueda, estas son extraídas de documentos o registros en una fuente de datos externa, este es el componente principal para poder ejecutar las búsquedas. por ejemplo, si se tiene un libro sobre El Quijote, al ingresar su contenido, se pueden identificar palabras clave por medio de las cuales los usuarios podrían encontrar este libro.

> ***"Casi en cualquier proyecto de desarrollo se tiene la obligación de implementar funcionalidades de búsqueda por algún requerimiento de los usuarios. "***

Por otro lado, el "query parser", es el componente que descompone las palabras que el usuario busca, de esta manera crea una estructura de consulta para ser enviada al motor de búsqueda, en otras palabras, traduce su cadena de búsqueda en instrucciones específicas para el motor de búsqueda. Se interpone entre usted y los documentos que está buscando, por lo que su papel en la recuperación de texto es vital. Por otra parte, el motor de búsqueda es el encargado de devolver los documentos o registros encontrados en base al índice, además de tomar en cuenta la jerarquización de los resultados encontrados.

Por último, es importante hablar sobre los analizadores, un analizador es el componente responsable de procesar el texto en cadenas de consulta y documentos indexados. Los diferentes analizadores transforman el texto de diferentes maneras según el escenario para obtener resultados deseados. Existen aquellos que analizan el lenguaje procesando texto por medio de reglas lingüísticas que mejoran la calidad de la búsqueda, otros analizadores realizan tareas como la conversión de caracteres a minúsculas, buscar sinónimos de las palabras consultadas, eliminación de palabras inútiles en una búsqueda (el, la, los, las), entre otros.

Por otro lado, las particiones son las que almacenan índices y dividen automáticamente los datos de búsqueda, dos particiones dividen su índice a la mitad, tres particiones lo

dividen en tercios, y así sucesivamente. La capacidad de las particiones es importante ya que va a determinar el precio del servicio.

Las réplicas son instancias del servicio de búsqueda, cada réplica alberga una copia equilibrada de carga de un índice, por ejemplo, un servicio con seis réplicas tiene seis copias de cada índice cargado en el servicio.

Por ultimo las Unidades de Búsqueda, (Search Units), son la multiplicación entre particiones y replicas,

R x P = SU. (Search Units) El mínimo de unidades de búsqueda es uno y el máximo es 36 como resultado de la multiplicación, por ejemplo, no es válido tener 12 réplicas por 12 particiones, sin embargo si es válido tener 6 réplicas con 6 particiones.

Parte 2. Creando y configurando Azure Search

Al entender los conceptos se procede a profundizar en el proceso de provisionamiento de Azure Search, una de las opciones es a través, del portal, comenzando con la siguiente pantalla.

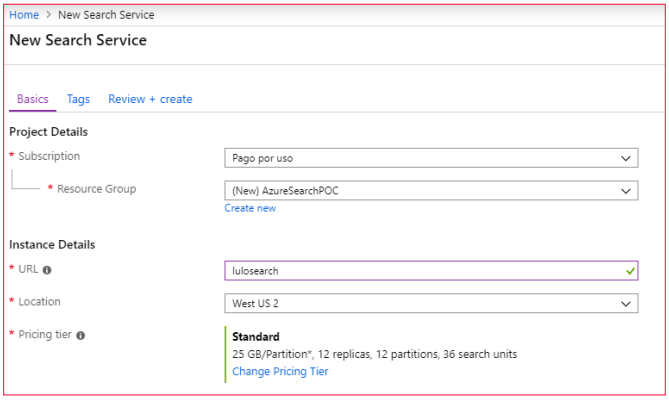


Imagen 1.- Configuración básica del servicio de Azure Search.

Primero seleccionamos la suscripción y luego el grupo de recursos, que nos ayuda a mantener una mejor visibilidad de todo lo que tenemos en ejecución en Azure, además sirve para administrar los permisos y para controlar los gastos del grupo como tal. En la pantalla se observa la URL que es el identificador único de nuestro servicio, mediante la cual se identifica desde el código al realizar las consultas.

"Los proyectos en Azure Search tienen un proceso similar en todos los casos: se crea el servicio – a través del portal, de Azure CLI o Powershell-, se crea el índice, se cargan datos y se realizan búsquedas."

También se puede observar la escala de precios la cual define la capacidad de almacenamiento y procesamiento de nuestro servicio.

Los precios pueden ser consultados en el siguiente enlace: https://azure.microsoft.com/en-us/pricing/details/search/

En el enlace anterior se puede observar que hay muchas opciones para seleccionar la que más convenga, todo depende de la capacidad de almacenamiento y procesamiento que se necesite (cantidad de índices, cantidad de almacenamiento, rendimiento esperado etc.).

Una vez hemos seleccionado nuestro nivel de precios, procederemos a la validación y creación del recurso.

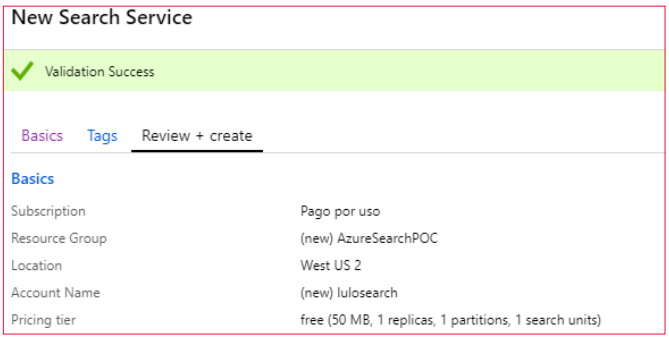


Imagen 2.- Validación del servicio de Azure Search.

Después de haber creado el servicio de búsqueda se procede a los siguientes pasos:

- Crear índices e importar datos desde las fuentes que tenemos (por ejemplo, tabla de SQL Server). El portal de Azure permite inferir el índice a partir de una fuente de datos, es decir al darle clic a importar datos y especificar la conexión, Azure Search infiere que campos retorna nuestra consulta, y nos permite configurar cada campo como deseemos.
  Al importar datos y seleccionar el tipo de conexión se puede realizar de Azure SQL, CosmosDB, Blob Storage u otros.
  En este paso es importante definir la consulta, en mi caso en CosmosDB tengo una colección compartida, es decir, almacene varios tipos de documentos dentro de la misma colección, utilizando un campo que se llama CosmosEntityName para poder identificar fácilmente el tipo de objeto, como pueden ver en la imagen busco solo entidades de tipo Rol, y luego Azure Search nos sugiere implementar una cláusula adicional en el WHERE, esta cláusula es para implementar re indexación de registros o documentos que han sido cambiados desde la última indexación.

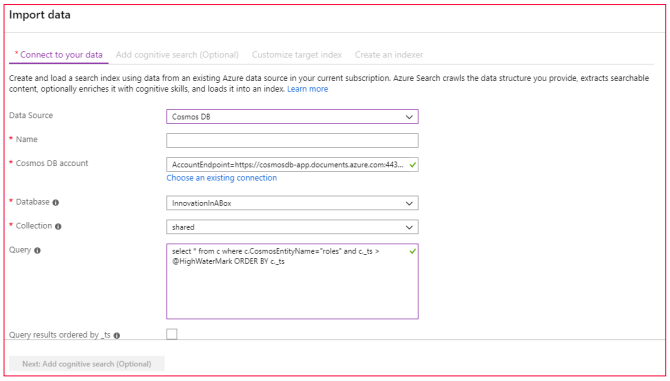


Imagen 3.- Importando datos en Azure Search.

- En el siguiente paso, Azure Search infiere los campos según los resultados de la consulta anterior y nos permite configurar cada campo por separado, para esto debemos definir los siguientes conceptos:

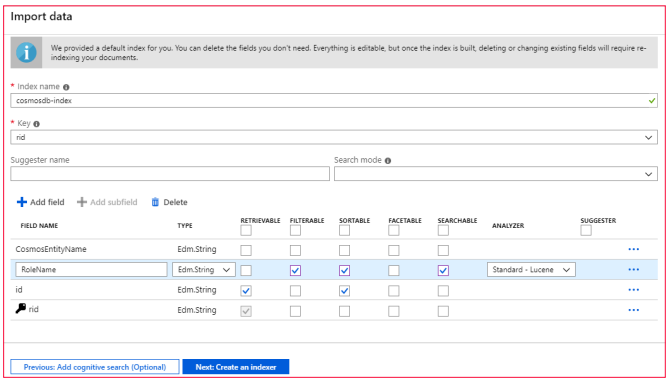


Imagen 4.- Campos inferidos por Azure Search.

  - Retrievable: Indica si el campo debe ser retornado o no al realizar la búsqueda.
  - Filterable: Indica si el campo puede utilizarse en expresiones de búsqueda, por ejemplo, Rolename=”Administrador”
  - Sortable: Indica si el campo se puede utilizar en una expresión para ordenar los resultados.
  - Searchable: Indica si el campo puede ser “buscado”, en este caso utilizamos el RoleName como campo para buscar.

“Las réplicas son instancias del servicio de búsqueda, cada réplica alberga una copia equilibrada de carga de un índice.“

- Y por último creamos el Indexador, en el cual podemos definir la recurrencia de indexación, Azure Search soporta hasta intervalos de 5 minutos, se puede ejecutar por demanda, o se puede crear un proceso para que se haga en tiempo real.

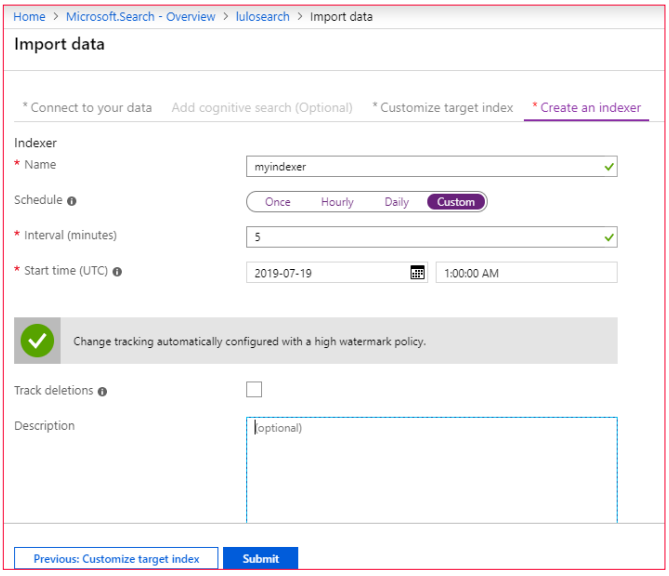


Imagen 5.- Configuración del indexador.

Una vez terminamos este paso, podremos revisar en el portal como se ha creado una fuente de datos:

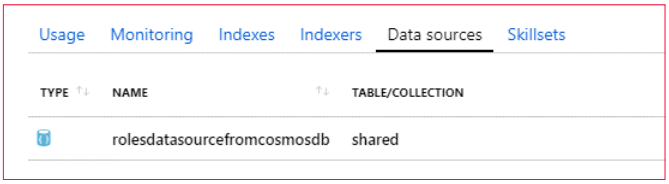


Imagen 6.- Fuente de datos creada.

El índice :

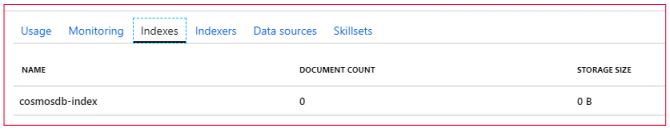


Imagen 7.- Índice.

El indexador:

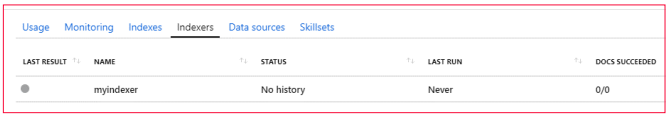


Imagen 8.- Indexador.

En este momento, el indexador no se ha ejecutado, como se puede ver en la pantalla tenemos cero documentos. Para solucionar esto, se selecciona el indexador y se ejecuta la consulta, este proceso agrega los documentos al índice, el tiempo de respuesta va a depender del tamaño de la consulta y los campos utilizados, es decir, de la cantidad de datos.

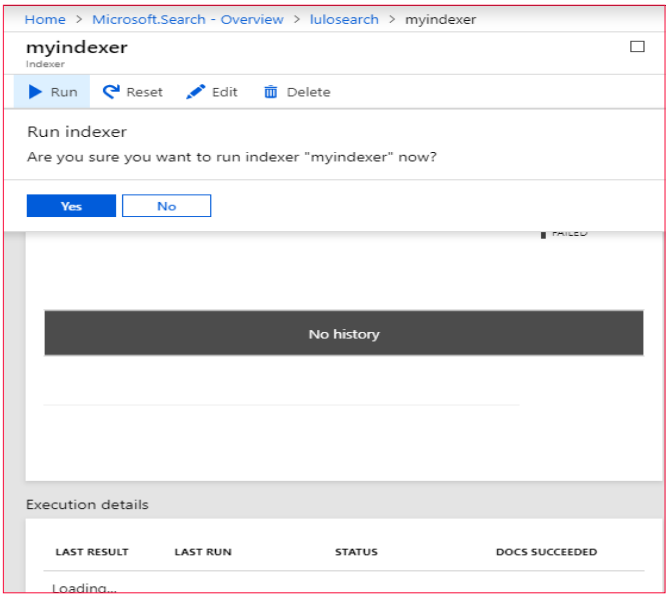


Imagen 9 - Ejecución del indexador.

Después de ejecutar la consulta en la pestaña de indexadores, podríamos ver cuando se ejecutó y el número de documentos que devuelve:

Imagen 10.- Resultado de la indexación.

Después de algunos segundos, podremos revisar la pestaña de Índices, y veremos cómo los documentos ya han sido indexados según la fuente de datos configurada.

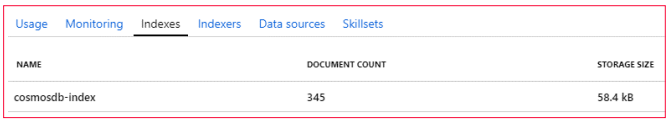


Imagen 11.- Pestaña de Índices.

Parte 3. Prueba del motor de búsqueda

La prueba del motor de búsqueda es importante mencionarla en este texto porque es necesario verificar que nuestro proceso de indexación fue realizado correctamente. Para esto, en la parte superior del portal le damos clic a Search Explorer y podremos probar la búsqueda en cualquier índice.

Es importante mencionar que se debe seleccionar el índice y luego digitar una consulta y presionar en el botón de búsqueda. En la parte inferior en JSON aparecerán los resultados según la configuración realizada anteriormente. Esto es una búsqueda sencilla que consulta en todos los campos indexados, sin embargo, es posible utilizar operadores de búsqueda más avanzados para buscar en campos específicos y también se pueden combinar con operadores AND, OR, entre otros.

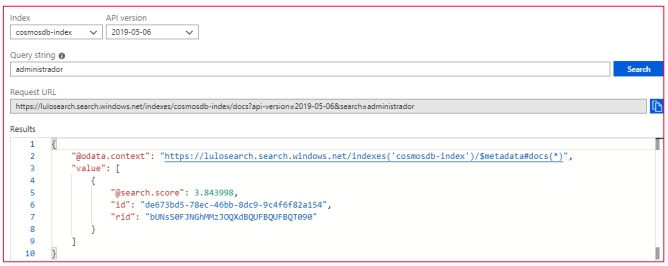


Imagen 12.- Prueba de la búsqueda con Azure Search

Conclusión

En este articulo pudimos ver la facilidad con la cual el portal de Azure nos permite crear el servicio de búsqueda, configurar índices, indizadores y fuentes de datos, a su vez que entendimos todos los conceptos necesarios para poder realizar una configuración sencilla de todos estos componentes. En este caso, este proceso no dura más de 10 minutos para terminar. En otro número de esta revista, revisaremos ejemplos más prácticos de código, utilizando búsquedas básicas y con operaciones complejas.

LUIS VALENCIA
Office Development MVP

11

Un repaso al Modern Admin Center de SharePoint Online

Tras las últimas funcionalidades añadidas al Modern Admin Center de SharePoint Online (SPO) por parte de Microsoft, podemos considerar que desde el punto de vista de administración del servicio proporciona todo lo necesario a los administradores de SPO para poder administrar y configurar la plataforma. En este artículo haremos un repaso al estado del arte actual (Julio de 2019) del Modern Admin Center de SharePoint Online.

"Podemos considerar que desde el punto de vista de administración del servicio el Modern Admin Center proporciona todo lo necesario a los administradores de SPO para poder administrar y configurar la plataforma."

Acceso al Modern Admin Center de SPO

El acceso al Modern Admin Center de SPO es posible:

- Desde el enlace "SharePoint" disponible en la sección de Admin Centers del menú de navegación del Admin Center de Office 365 que proporciona, para tenants con menos de 50 usuarios, acceso directo al Modern Admin Center de SPO.
- Desde el botón "Abrir ahora" que se muestra en el banner informativo en el Admin Center clásico de SPO que sigue siendo la opción por defecto para tenants con más de 50 usuarios.

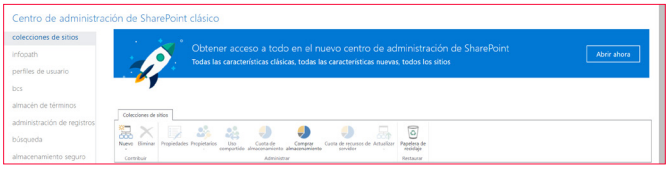


Imagen 1.- Banner de acceso al Modern Admin Center para tenants con más de 50 usuarios.

- Escribiendo la siguiente URL en el navegador: https://<-DominiO365>-admin.sharepoint.com/_layouts/15/online/AdminHome.aspx#/home

Cualquiera de las opciones nos lleva a la página principal del Modern Admin Center de SPO en el que se visualizan:

- Informes de uso de actividad en SPO relativos actividad con archivos en sitios de SPO y sitios activos frente al total de sitios en el tenant.
- Mensajes del centro de mensajes de Office 365 específicos para SPO.
- El estado de saludo del servicio de SPO.

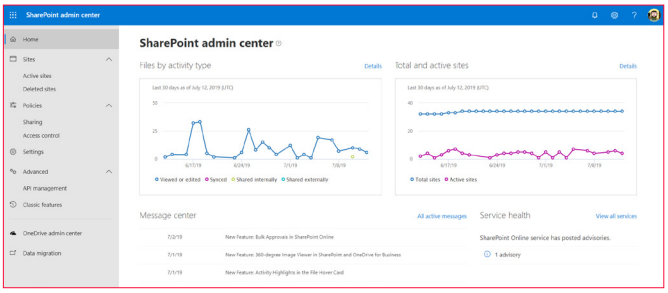


Imagen 2.- Página principal del Modern Admin Center de SPO.

A continuación, se detallarán cada una de las secciones principales del Modern Admin Center de SPO.

Sección Sitios (Sites)

Proporciona acceso tanto al listado de Colecciones de Sitios del tenant de SPO (Sitios activos) como a las Colecciones que se han borrado y se encuentran en la papelera de reciclaje bien para su borrado definitivo (manual o bien automático una vez transcurren 30 días desde que se borró un Sitio) o bien para que se puedan restaurar. En el caso de la página de Sitios activos, las principales funcionalidades que proporciona son las siguientes:

- Mostrar el listado completo de Sitios en el tenant, incluyendo Sitios de comunicación y Sitios de equipo moderno. El listado se visualiza haciendo uso de la misma experiencia de usuario que tenemos para listas y bibliotecas modernas de SPO por lo que podemos aplicar filtros, ordenaciones, realizar búsquedas de Sitios, modificar vistas de Sitios existentes o bien crear nuevas vistas. La siguiente imagen muestra dicho listado en un tenant de prueba y también las vistas provistas por defecto.

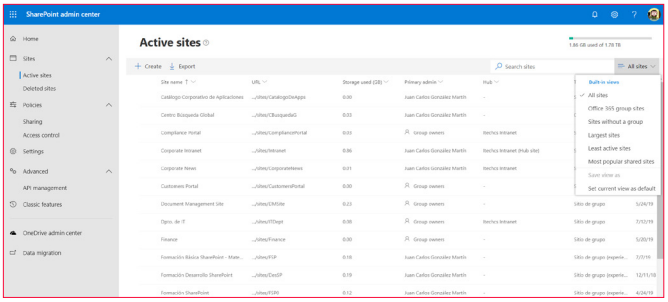


Imagen 3.- Listado de Sitios activos en un tenant y vistas de información por defecto.

- Crear nuevos Sitios ya sean modernos (Sitios de Comunicación o Sitios de equipo modernos) o clásicos.
- Si se selecciona un Sitio en el listado, se habilitan una serie de opciones en la barra de acciones y a través del icono de "i" se puede visualizar los detalles del Sitio seleccionado. Por ejemplo, para un Sitio de comunicación se muestran las siguientes acciones:
  - Owners, permite cambiar el administrador principal del Sitio o bien los administradores secundarios que estén configurados.
  - Hub, si el Sitio es un Hub site permite modificar las configuraciones del Hub o bien anular el registro como Hub del site. Si el Sitio no es un Hub, se muestra la opción de cambiar el Hub al que está asociado al Sitio o bien de registrar el Sitio como Hub site si no está asociado a un Hub.
  - Sharing facilita configurar la opción de "Compartir" para el sitio seleccionado indicando uno de los siguientes valores: Cualquiera, Invitados nuevos y existentes, Solo invitados existentes, Solo los miembros de su organización.
  - Storage, facilita administrar la cuota de almacenamiento del Sitio en el caso en el que las cuotas de SPO se hayan configurado para que se gestionen de forma manual.

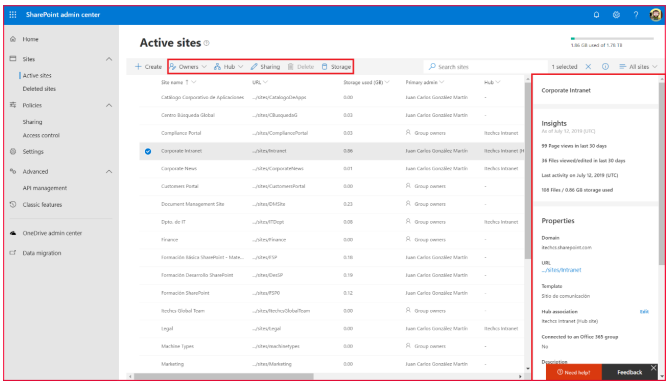


Imagen 4.- Acciones disponibles para el Sitio seleccionado y panel de detalle del Sitio.

"La sección Sites proporciona acceso tanto al listado de Colecciones de Sitios del tenant de SPO (Sitios activos) como a las Colecciones que se han borrado y se encuentran en la papelera de reciclaje"

Directivas (Policies)

Esta sección proporciona acceso a las directivas o políticas de SPO relativas a "Compartir" y "Control de Acceso". La página de "Compartir" permite administrar el uso compartido externo para SPO y OneDrive For Business (ODFB):

- Por defecto, tanto en SPO como en ODFB está habilitada la posibilidad de compartir contenido (Documentos y Carpetas) con cualquier usuario sin que tenga que iniciar sesión para poder acceder a este. Partiendo de esta configuración base, se puede restringir para ambos servicios el uso compartido de acuerdo con las siguientes posibilidades: Invitados nuevos y existentes, Solo invitados existentes, Solo los miembros de su organización.
- Se pueden aplicar configuraciones avanzadas al uso compartido externo como por ejemplo limitar que solo se pueda compartir con usuarios externos que pertenezcan a ciertos dominios admitidos, forzar a que los usuarios invitados tengan que iniciar sesión con la misma cuenta a la que se le envío la invitación o bien permitir o no que los invitados puedan compartir contenidos de los que no son propietarios.
- Se puede configurar el vínculo por defecto para compartir de acuerdo con las siguientes posibilidades: Personas específicas, Solo los miembros de su organización, Cualquiera con el vínculo. Para este último caso, además se puede establecer por defecto el permiso por defecto para archivos y carpetas, así como el tiempo de expiración (Número de días) para el contenido que se ha compartido.

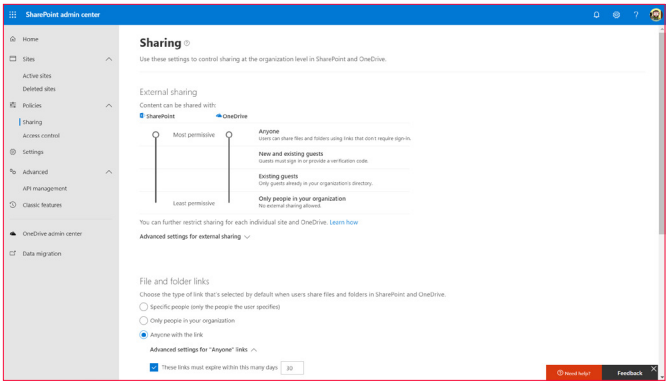


Imagen 5.- Página de "Compartir" en el Modern Admin Center de SPO.

La sección de "Control de acceso" proporciona acceso a las siguientes configuraciones:

- Dispositivos no administrados: permite establecer la configuración de acceso condicional a aplicar para dispositivos no gestionados por la organización que acceden a contenidos almacenados en SPO y ODFB. Para poder aplicar una configuración de acceso condicional, se requiere disponer de una suscripción de EMS que permita establecer las configuraciones de acceso definidas a nivel corporativo.
- Cierre de sesión inactiva (Iddle Session Sign-out): permite configurar el cierre automático de sesiones de usuario una vez que ha transcurrido un período de tiempo sin actividad.
- Ubicaciones de red: facilita configurar las ubicaciones de red (direcciones IP o rango de direcciones IP) desde las que se permite el acceso a los servicios de SPO y ODFB.
- Aplicaciones que no usan autenticación moderna: permite configurar si se permite el acceso a contenidos de SPO y ODFB a aplicaciones que no usen autenticación moderna. Un ejemplo de dichas aplicaciones puede ser cualquiera de la suite ofimática Office 2010.

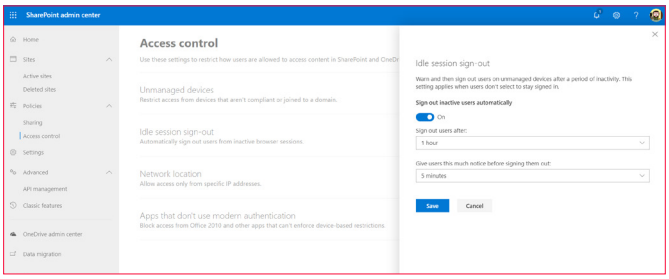


Imagen 6.- Sección "Control de acceso".

Configuración (Settings)

Desde esta sección se pueden realizar las siguientes configuraciones:

- Notificaciones, opción activa por defecto de manera que los usuarios reciben notificaciones relativas a contenidos de SPO en aplicaciones móviles de Office.
- Límite de almacenamiento del sitio, permite establecer el modo por defecto para administrar el espacio de almacenamiento de cualquier Colección de Sitios del tenant: manual vs. automático.
- Experiencia de administración predeterminada, establece el Admin Center de SPO por defecto: clásico vs. moderno. Para tenants con menos de 50 usuarios, la opción por defecto es la del Modern Admin Center.
- Creación de sitios, permite establecer los siguientes parámetros relativos a creación de Sitios:
  - La ruta por defecto bajo la que se crearán nuevos Sitios.
  - La zona horaria por defecto para los nuevos Sitios a crear.
  - El espacio de almacenamiento máximo para los nuevos Sitios a crear.

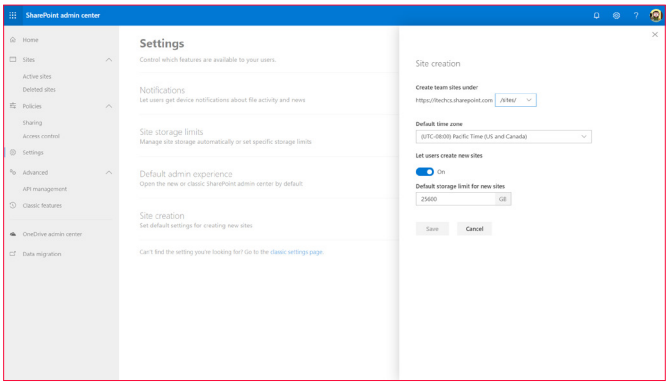


Imagen 7.- Sección de "Settings".

Avanzado

Esta sección proporciona acceso a características avanzadas de administración de Office 365. A fecha de julio de 2019, únicamente existe una opción de configuración en esta sección: Administración de API. Esta opción muestra las APIs que se han habilitado para ser llamadas desde desarrollos para SPO.

> ***"Características clásicas proporciona un acceso directo a las opciones de administración clásicas que no han sido modernizadas todavía por Microsoft."***

Características clásicas

Proporciona un acceso directo a las opciones de administración clásicas que no han sido modernizadas todavía por Microsoft.

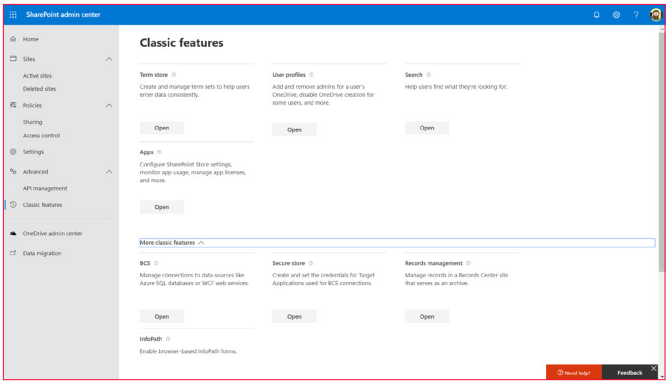


Imagen 8.- Acceso a las Características clásicas en el Modern Admin Center.

Conclusiones

Tras años de trabajo, podemos decir que el Modern Admin Center de SPO incorpora todas las funcionalidades que requiere un administrador de la plataforma para poder configurar y manejar de forma completa tanto los Sitios que se hayan creado como los servicios que conforman la plataforma.

JUAN CARLOS GONZÁLEZ
Office Apps & Services MVP | Office 365 SME en RICOH
@jcgm1978

14

Entrevista Juan Ignacio Oller

Soy Juan Ignacio Oller Aznar, nací y crecí en Zaragoza, España y a lo largo de los años he trabajado en distintas localidades de Aragón y Barcelona.

He trabajado mayormente como administrador de sistemas y comunicaciones, aunque en los últimos años he comenzado a trabajar como consultor de optimización de datacenters.

He sido galardonado como MVP en Cloud and Datacenter Management desde el año 2011.

Actualmente estoy trabajando como consultor de sistemas e instructor de distintas tecnologías.

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

La verdad es que en mi caso era algo natural, ya que mi familia se dedica a los montajes eléctricos industriales desde principios del siglo XX y siempre se ha hablado de temas relacionados con la electricidad, electrónica y lo que cuando yo era crío se llamaban “Nuevas Tecnologías”, algo que hoy en día suena tan anacrónico.

Cuando tenía 10 años, mis padres me regalaron un Amstrad CPC, aquella maravilla en 8 bits que en aquel momento era una auténtica joya y que aún conservo. En aquella época por supuesto, lo que más me interesaba era jugar, sobre todo aquellas aventuras conversacionales como “La guerra de las vajillas”, pero empecé a hacer amagos de programar en Basic, que en realidad consistían en seguir a rajatabla la guía que venía al final del libro, aunque entonces, empecé a hacer pequeños experimentos para ver que sucedía si cambiaba las cosas de sitio, sin mucho éxito, por cierto. Pero bueno, en el momento en que me regalaron el Amstrad yo tenía 9 o 10 años con lo que considero esos fracasos más una victoria que una derrota.

Después todo siguió su curso y cuando tuve que elegir si saltar de EGB a BUP y COU o irme por la rama técnica vía Formación Profesional (FP) decidí tomar el segundo camino y seguir los pasos de la familia. En aquel momento, casi todos los compañeros nos decían aquello de “la FP es para fracasados que no saben hacer nada” y con el tiempo alguno de esos me ha dicho “ojalá hubiera hecho lo mismo que tú”. En aquella FP me especialicé en electrónica, cosa que me ha venido muy bien en más de una ocasión.

Durante la FP, estuve haciendo prácticas en una empresa de informática, me empezó a picar el gusanillo, después, uno de mis profesores de esa misma FP, que también era profesor de universidad me empezó a arrastrar al lado oscuro, y así, un chaval que estaba loco por irse a trabajar, terminó comenzando nueva singladura como estudiante de ingeniería informática.

Desde entonces no he parado de estudiar, tanto en la universidad como certificaciones de distintos tipos, casi siempre relacionados con sistemas operativos, redes y seguridad.

¿Cuáles son tus principales actividades tecnológicas hoy en día?

En la actualidad estoy en un periodo de cambio, ya que durante más de 15 años he estado trabajando en departamentos de sistemas y comunicaciones o consultoras como administrador de sistemas “on premise”, pero hace un año hice un parón, ya que estaba algo cansado de esos trabajos y acepté la oportunidad de colaborar como profesor en un centro universitario.

Desde entonces tengo roles duales, por un lado, la parte docente, que no la considero tecnológica, y por otra parte labores de consultoría relacionadas con sistemas y comunicaciones, sobre todo en lo que respecta a Microsoft

System Center y temas relacionados con ISOS, como por ejemplo la 27001. Otros sistemas que estoy comenzando a tocar, después de casi 10 años, son los sistemas GNU/Linux, teniendo que empezar a hacer pequeñas labores de administración y consultoría sobre entornos RHEL. Por supuesto, también me toca jugar y dar soporte con temas relacionados con Azure.

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

A día de hoy, colaboro con un centro universitario impartiendo formación en una carrera de informática e imparto formación como instructor certificado de Cisco Systems y Microsoft para varios centros, también colaboro con varias comunidades, como por ejemplo DTC2Mobility de la que soy miembro fundador y estoy colaborando con otras comunidades en la implantación de charlas semanales en centros de estudios.

Otra actividad, a la que por desgracia no dedico todo el tiempo que debería es el voluntariado en protectora de animales, en concreto con ADALA, a la que me "arrastra" a veces mi novia, que invierte mucho de su tiempo libre en ayudar a esos animales que buscan una segunda oportunidad. Gracias al trabajo de mi novia hemos sido casa de acogida, responsables de guardería, etc. Es un trabajo que a veces se hace duro, pero cuando ves cómo se recuperan y lo felices que son compensa 100%.

¿Cuáles son tus hobbies?

Mis principales hobbies están relacionados con la construcción de maquetas y la recreación de batallas históricas a pequeña escala, sobre todo de la edad antigua (siento debilidad por el imperio romano) y de la segunda guerra mundial. Por supuesto tengo otros hobbies como el cine, los libros de historia y por supuesto, trastear con mi entorno virtualizado para probar distintas tecnologías que no puedo tocar en el trabajo.

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

Recuerdo que, en 2009, cuando Microsoft empezó a hablar de Azure, tomando un café nos juntamos varios informáticos y todos pensábamos que eso no triunfaría y sería una moda pasajera, pero está claro que o bien es una moda muy larga, o nos equivocamos de extremo a extremo. Está claro que, en la actualidad, nos encontramos en un periodo de cambio (por no decir que, si no hemos cambiado, ya vamos tarde), y tal como empezaba mi charla sobre Seguridad en Azure hace un par de semanas, no se trata de pensar en si se va a subir a la nube o no, porque básicamente todos vamos a acabar allí, si es que no lo estamos ya.

Con los servicios y nuestros datos en la nube y con la capacidad de cómputo que tenemos disponible en las nubes públicas a nuestro servicio, nuestra capacidad de crecimiento es exponencial, pero también lo son las amenazas, que cada vez son más complejas y de adaptación más rápida a las contramedidas que se interponen en su camino, por tanto, creo que aquí tenemos uno de los vectores de mayor interés en el panorama tecnológico actual, el de los sistemas dedicados a prevenir y paliar los ataques a las nubes de cualquier tipo y de los perfiles técnicos encargados de gestionarlas.

Por supuesto, todos estos sistemas tienen una base de apoyo sobre la IA y otras tecnologías que, si bien en la actualidad ya están tomando fuerza, verán un gran crecimiento en años venideros.

JUAN IGNACIO OLLER AZNAR
MVP Cloud and Datacenter Management
@jioller

16

Uso de Funciones de Azure con Flow y PowerApps - Part 1: OpenAPI y Funciones

Flow es la implementación que Microsoft ha creado para integrar el motor de flujos de trabajo de Azure, Logic Apps, en Office 365. A su vez, PowerApps 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 utilizando la posibilidad de "conectores" para Flow y PowerApps.

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

1.– Como crear Funciones de Azure para que puedan ser utilizadas por Flow y PowerApps (CompartiMOSS No. 41).

2.– Usando Funciones de Azure con Office Flow (CompartiMOSS No. 42).

3.– Usando Funciones de Azure con PowerApps (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 PowerApps: aunque ofrece conectividad con muchos otros tipos de sistemas, no es posible definir capacidades de cálculo dentro de la aplicación misma.

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

Para solucionar el problema, ambos sistemas permiten la utilización de "OpenAPI" para crear "conectores". OpenAPI es 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 Flow y PowerApps 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.

Las funciones de Azure se pueden programar en una variedad de idiomas de programación (C#, 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 Flow o PowerApps.

Microsoft ha publicado parches para poder crear Funciones de Azure para que puedan ser utilizadas directamente desde Flow y PowerApps, 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 primer artículo de la serie de tres, se indica como crear una Función de Azure que expone su API por medio

de un OpenAPI del Azure API Management, que, a su vez, puede ser utilizado desde Flow y/o PowerApps. 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 Flow acoplado a la Lista pasa los datos de la reserva a una 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 independiente de PowerApps. El algoritmo para determinar si una sala está ocupada o no, no está 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.

Creación de la infraestructura en Azure

1.– Abra el portal de Azure (https://portal.azure.com) utilizando una cuenta con suficientes derechos para crear servicios. Una cuenta temporal puede ser creada desde el mismo sitio.

2.– Cree un Grupo de Recursos: abra la sección de Grupos de Recursos (Resource Groups) y utilice el botón de "+Add" para crear uno nuevo.

3.– Después de que el grupo ha sido creado, ábralo y agregue un servicio de storage: use el botón de "+Add" en el grupo de recursos, introduzca en la casilla de búsqueda "storage account", haga clic sobre el bloque de resultados correspondiente y haga clic sobre "Create". Introduzca los datos pedidos sobre la suscripción y el Grupo de Recursos creado anteriormente. Defina un nombre para el servicio y la localización de su centro de datos. Seleccione "Standard" para "Performance", "StorageV2" en "Account kind", "Read-acces geo-redundant" en "Replication" y "Hot" en "Access tier". Use el botón de "Review + create" y luego "Create" para crear el servicio.

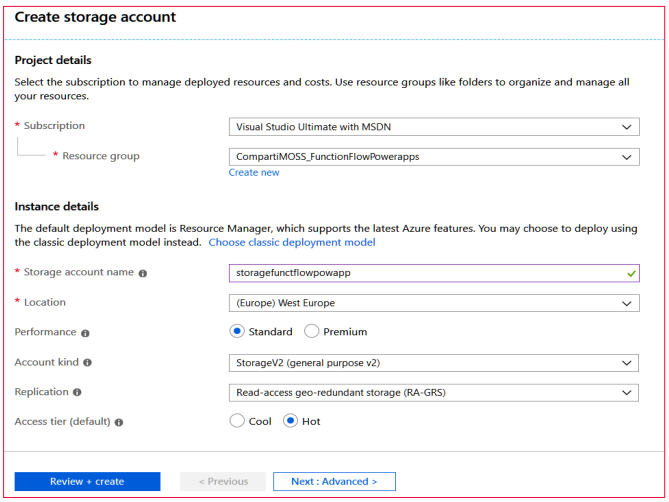


Imagen 1.- Creación de una cuenta de almacenamiento

4.– Cree un App Service para alojar la Función de Azure: Regrese al Resource Group creado en el punto 2, use el botón de "+Add" y busque esta vez por "function app". Haga clic sobre el bloque correspondiente en los resultados y también sobre el botón de "Create". Defina el nombre para el servicio en la siguiente ventana, seleccione la suscripción y el Grupo de Recursos creado anteriormente. Seleccione "Windows" como "OS", "Consumption Plan" en "Hosting Plan", la localización de su centro de datos, ".NET Core" en "Runtime Stack" y el storage creado anteriormente. Si desea, remueva el servicio de "Application Insights" al final.

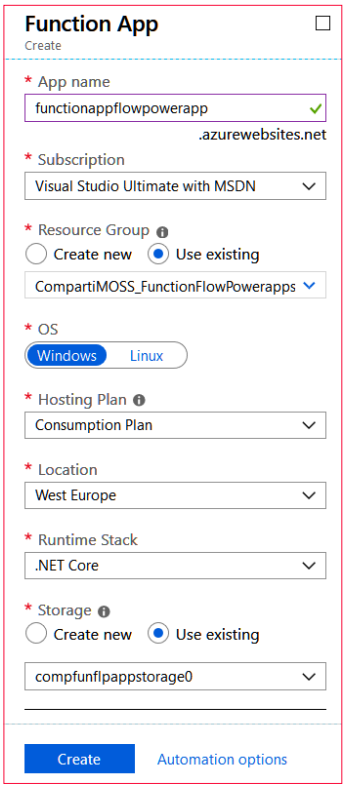


Imagen 2.- Creación de la Function App en Azure

5.– Cree un servicio de API Management que es el que se va a encargar de convertir la Función en un recurso de OpenAPI: Regrese al Grupo de Recursos, use el botón de "+Add" y busque por "api management". Haga clic sobre el bloque de "API Management" y el botón de "Create". Defina un nombre para el servicio, seleccione la suscripción y el Grupo de Recursos, la localización de su centro de datos y defina un nombre para la organización y su email (estos dos datos no son validados, pero son obligatorios). Seleccione "Consumption (99.9 SLA, %)" en el "Pricing tier". Si desea, seleccione la creación de un servicio de Application Insights.

"Las Funciones de Azure proporcionan toda la infraestructura técnica para poder crear funcionalidad "serverless"."

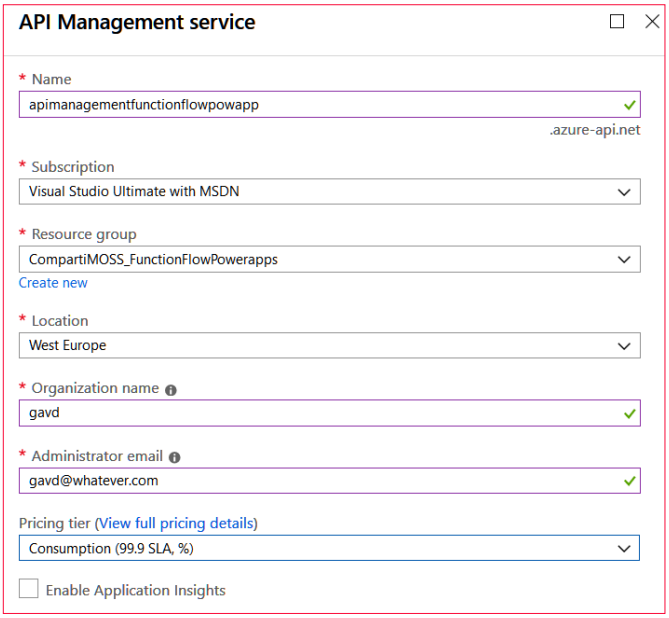


Imagen 3.- Creación del servicio de API Management en Azure

6.– Cambie la definición de las funciones a versión 1. Funciones de Azure pueden ser configuradas como versión 1 o 2. La versión 1 funciona con el .NET Framework de Microsoft y la versión 2 con el .NET Core Framework. Como todos los APIs de SharePoint no son (todavía) compatibles con el .NET Core Framework, todas las Funciones de Azure que tengan que trabajar (o que potencialmente vayan a trabajar) con SharePoint tienen que ser de versión 1. Regrese al Grupo de Recursos y haga clic sobre el "App service" creado en el punto 4. Haga clic sobre "Platform features" y luego sobre "Function app settings" (sección "General Settings"). En la sección de "Runtime version" seleccione "-1".

7.– Cree la función. Desde la ventana del App service, haga clic sobre el botón "+" al lado derecho de "Functions" para crear una nueva función. Haga clic sobre "C#" en el bloque "HTTP Trigger". Seleccione "C#" en "Language", defina un nombre para la función y seleccione "Function" en "Authorization level". Cree la nueva función.

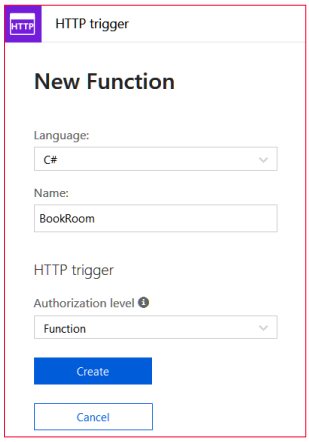


Imagen 4.- Creación de una Función de Azure

8.– Codificación de la función. La función es creada con algo de código por defecto que permite capturar los parámetros de entrada, sea que se hayan enviado por medio del QueryString o en el Body de la llamada. En el ejemplo se va a utilizar solamente el QueryString porque Flow y PowerApps tiene problemas haciendo llamadas REST que utilizan el Body para enviar parámetros.

Reemplace todo el código original por el siguiente framento:

```
using System;
using System.Net;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWriter log)
{
  log.Info("Request for a Conference Room is arrived");

  // Parse query parameters
  string roomname = req.GetQueryNameValuePairs()
      .FirstOrDefault(q => string.Compare(q.Key, "roomname", true) == 0)
    .Value;
  log.Info("roomname = " + roomname);

  string persons = req.GetQueryNameValuePairs()
    .FirstOrDefault(q => string.Compare(q.Key, "persons", true) == 0)
    .Value;
  log.Info("persons = " + persons);

  HttpResponseMessage myResponse = null;
   if(string.IsNullOrEmpty(roomname) == true || string.IsNullOrEmpty(persons) == true)
  {
        myResponse = req.CreateResponse(HttpStatusCode.BadRequest, "roomname and number of persons are obligatory");
  }
  else
  {
    // Faking a room algorithm ...
    string[] roomState = { "Free", "Occupied" };
    Random myRnd = new Random();
    int stateIndex = myRnd.Next(roomState.Length);

     myResponse = req.CreateResponse(HttpStatusCode.OK, roomState[stateIndex]);
  }

  log.Info("Returning - " + myResponse);
  return myResponse;
}
```

Testee la función para comprobar que no hay errores de compilación.

9.– Configuración del API Management para exponer la función como un OpenAPI. Regrese al Grupo de Aplicaciones y haga clic sobre el servicio del API Management creado en el punto 5. En la nueva ventana, haga clic sobre "APIs" (menú vertical izquierdo) y luego sobre el botón de "Function App" en la sección de "Add a new API". En la nueva ventana, use el botón de "Browse" y luego el de "Function App", el que abre una ventana con una lista con todas las Function Apps en la suscripción. Elija la Function App correcta, lo que produce a su vez una lista con las funciones dentro de ella. Seleccione la función creada en el punto 8, y luego el botón "Select". Los nombres para utilizar aparecen automáticamente, extraí-

dos del nombre del Function App, pero pueden ser configurados a discreción.

Nota: Una función puede ser exportada directamente a un servicio de API Management (desde la ventana de "Platform features" del App Service, vinculo "API Management" en la sección "API"), pero se pierde el control sobre nombres y configuración del servicio, por lo que es mejor hacerlo siguiente este procedimiento. Igualmente, desde esta sección hay un botón para exportar la definición directamente a Flow y PowerApps, pero, como se indicó inicialmente, solamente funciona si tanto Azure como Office 365 comparten el mismo Directorio Activo (AD).

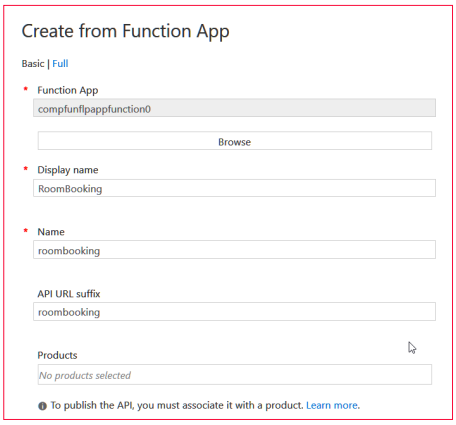


Imagen 5.- Configuración del API Management

Utilice el botón de "Create". La ventana de "Operations" aparece con toda la configuración del API para la función:

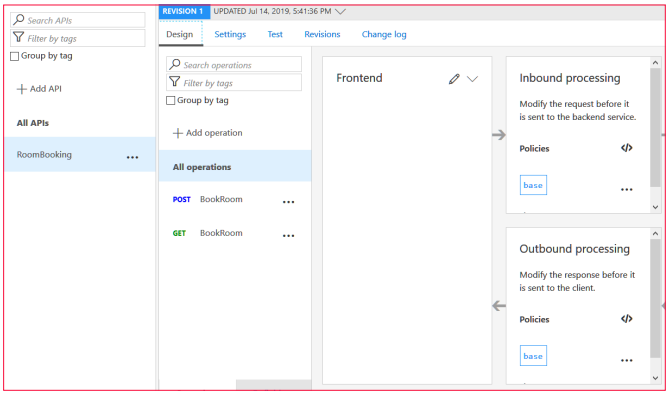


Imagen 6.- Ventana de configuración del API Management

Haga clic sobre el método "POST" (bajo "All operations"). En el panel de "Frontend" haga clic sobre el botón con el lápiz para configurar los parámetros de entrada del QueryString. Seleccione la pestaña de "Query" y agregue (botón "+Add parameter") los nombres de los dos parámetros del QueryString que la función espera encontrar ("roomname" y "persons"). En la misma ventana se puede configurar el tipo de parámetro y un valor por defecto, pero solamente el nombre es obligatorio. Después de guardar los cambios, la ventana de configuración aparece de nuevo mostrando los dos nombres de los parámetros.

Para comprobar que el API funciona correctamente, seleccione la pestaña de "Test" con el método "POST" seleccionado, suministre valores para los dos parámetros, y use el botón de "Send". En la respuesta debe aparecer "Free" o "Occupied". La función presenta su funcionalidad desde este momento al mundo externo en forma de un método REST que es compatible con la definición de OpenAPI que se va a generar. En el siguiente artículo de la serie veremos cómo exportar la definición del método y como utilizarlo en Microsoft Flow.

Conclusión

Para darle más flexibilidad y capacidad de interacción con otros sistemas, Microsoft Flow y PowerApps 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 estándar de OpenAPI (primer artículo), como conectar Flow con la función (segundo articulo) y como PowerApps puede utilizar la misma función (tercer articulo).

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

20

El nivel de acceso Archive en Azure

En las cuentas de Storage en Azure, disponemos de varios elementos de configuración que influyen directamente en la disponibilidad de la información que vamos a almacenar, el precio de tenerlo guardado, el de acceder a los datos y la latencia que tendremos hasta poder leer el primer byte de información. Entre esos elementos está el propio tipo de cuenta y el nivel de rendimiento, así como el tipo de replicación. Para simplificar todo y no desviarnos del objetivo de este artículo, nos centraremos en las cuentas de tipo propósito general V2 (StorageV2) con nivel de rendimiento estándar y replicación LRS (Locally-redundant storage). De este modo nos podremos centrar en el elemento que nos interesa explicar hoy: el nivel de acceso.

"El precio del almacenamiento de información en la nube es bajo en comparación a los sistemas on-premise, pero es necesario conocer sus características para no llevarnos sorpresas."

El nivel de acceso es una propiedad a nivel de cuenta, que también se puede establecer a nivel de archivo individual.

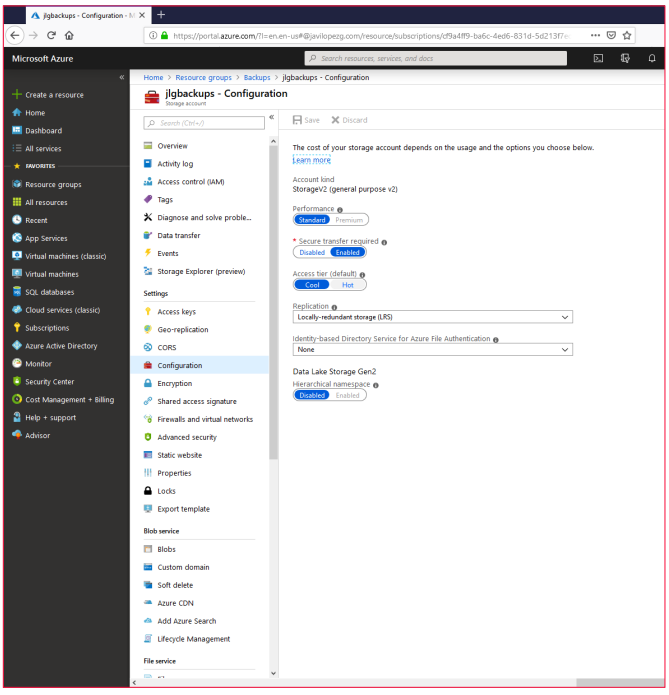


Imagen 1.- Configuración de cuenta de Storage.

Los tipos de nivel de acceso básicos son dos: Hot (frecuente) y Cool (esporádico). El primero te ofrece una mayor disponibilidad, un precio más alto por el almacenamiento y un precio más bajo por el acceso a los datos. El segundo ofrece un 9 menos en cuanto a disponibilidad, su precio de almacenamiento es más barato (un 25%) pero las operaciones con la información son más caras (más del doble en función del tipo de operación). Además, los ficheros almacenados con un nivel de acceso Cool deben pasar al menos 30 días en dicho estado, y de no ser así habrá repercusiones en tu factura.

Como decíamos, estos tipos de nivel básicos se pueden poner a nivel de cuenta o se puede modificar individualmente en cada archivo almacenado.

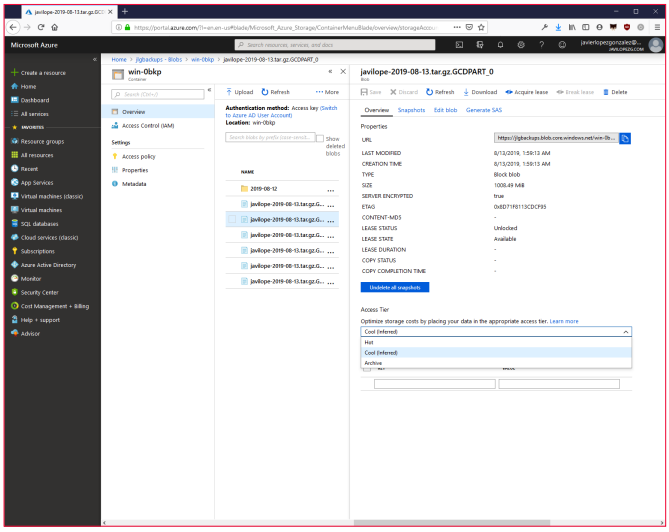


Imagen 2.- Configuración de nivel de acceso de un blob individual.

En ese caso, vemos que aparece un tercer nivel: Archive. Este nivel sólo se puede establecer en los ficheros individuales y no a nivel de cuenta, por lo que no es un nivel que se pueda heredar.

Los archivos en niveles Hot y Cool se almacenan en discos magnéticos online, pero los ficheros del nivel Archive se almacenan offline teniendo una gran repercusión tanto en el precio del almacenamiento (un 5% respecto a Hot), como de su disponibilidad (de 0%) y latencia (de horas vs los milisegundos de los otros niveles).

Las operaciones de escritura tienen el mismo coste que el nivel Cool, pero las de lectura son entre 500 y 2500 veces más caras en función de si te vale tener acceso a la información en hasta 15 horas o la necesitas cuanto antes (al

menos en una hora).

El hecho es que al estar offline hace que no podamos acceder a estos archivos directamente. Para poder leer a su información tendremos que hacer una operación de "rehidratación" devolviéndolos a alguno de los otros niveles y esperando a que las unidades en las que estén almacenados se vuelvan a poner online para transferir la información.

> ***"Una de las mayores pegas que tiene este nivel de acceso, es que no se puede heredar y por tanto hay que ir asignándolo a cada archivo individualmente."***

Sin embargo, a pesar de estar offline, sí que podremos acceder a sus metadatos y por lo tanto listarlos y ver sus propiedades. Usando Powershell podremos usar los siguientes métodos de los blobs cuando estén en nivel Archive: GetBlobProperties, GetBlobMetadata, ListBlobs, SetBlobTier, and DeleteBlob.

Estas características los hacen ideales, por ejemplo, para escenarios donde se precisa guardar copias de seguridad que probablemente no vayan a ser accedidas nunca, por ejemplo, de un equipo o un servidor que no tenga muchos cambios ni muchos riesgos. Podríamos montar la cuenta de blobs como si fuese un disco más (en Linux o en Windows) e ir copiando ahí los archivos con los datos de tal modo que usando nuestro sistema de copia tradicional estaríamos explotando un medio de almacenamiento muy económico.

Una de las mayores pegas que tiene este nivel de acceso, es que no se puede heredar y por tanto hay que ir asignándolo a cada archivo individualmente. Esto se puede remediar creando una cuenta de Azure Automation y creando un Runbook que se ajuste a nuestras necesidades. Por ejemplo, en la galería podéis encontrar un Runbook de Powershell creado por Marc Kean que permite archivar todos los blobs de una cuenta que sean más viejos de N días.

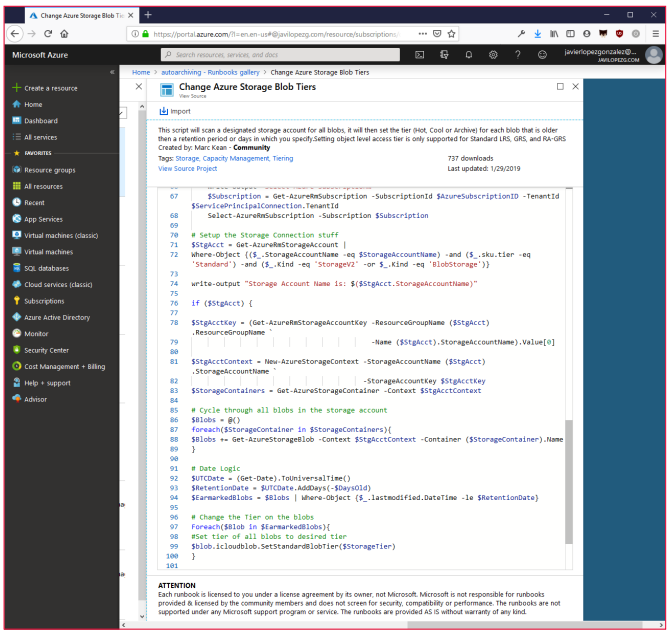


Imagen 3.- Runbook que cambia el nivel de acceso de los archivos más viejos de N días.

Del mismo modo, se podría crear un Runbook que cambiara el nivel de cada blob que se cree en una cuenta, o de todos los archivos de un container menos del que sea más actual, etc.

Al igual que pasaba con los archivos del nivel Cool, que precisaban estar en ese estado 30 días, en el caso del nivel Archive han de estar 180 días si no quieres tener repercusiones en tu factura.

Ahora, conociendo cómo funcionan, ya estamos preparados para usar este nivel de acceso y aprovecharnos del precio tan económico que ofrece sin llevarnos la gran sorpresa cuando queramos acceder a la información.

JAVI LÓPEZ
Freelance
mail@javilopezg.com
@javilopezg
https://javilopezg.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.

23

SPFx React Hooks ¿Qué son y como los podemos utilizar en nuestros desarrollos?

La reciente versión del Yeoman de SPFx la versión 1.9.1 ha traído una multitud de novedades que van a mejorar mucho la vida del desarrollador. Para mí las dos novedades principales son:

- El poder utilizar la versión 16.8.5 de ReactJS, cuya principal novedad son los React Hooks de los cuales hablaremos a lo largo del artículo.
- La migración de las herramientas de compilación a la versión 4 de WebPack. Este cambio es transparente para el desarrollador, salvo que se tenga extendido este proceso. Sin embargo, el subir la versión da una mejora notable del tiempo de compilación, algo que para proyectos grandes y pesados podía suponer más de 10 minutos de compilación. Con este cambio el tiempo de compilación se llega a reducir en un 50% el tiempo de compilación.

Esta versión era una de las más esperadas debido a que en la versión 1.9 se introdujeron todas estas novedades sin embargo hubo una serie de issues y bugs en dicha versión que obligo al equipo de producto a retirar la versión 1.9 y volver a la versión 1.8.2.

"Podemos definir un Hook como una función en la que podemos acceder a elementos propios de React como son su estado y su ciclo de vida."

Introducción

La primera pregunta que nos hacemos sobre los Hooks es: ¿qué son? Podemos definir un Hook como una función en la que podemos acceder a elementos propios de React como son su estado y su ciclo de vida. Tras esta primera definición, muchos desarrolladores nos hacemos la siguiente pregunta: ¿pero eso no lo podemos hacer con Clases? La respuesta es que sí se puede hacer haciendo uso de las clases. Pero entonces: ¿Por qué viene la motivación de reemplazar las clases por funciones? El motivo no es otro de que por un lado mejorar legibilidad del código al poner los hooks en funciones aisladas de forma que el código queda más legible y mantenible. El otro motivo es la refactorización del código, es difícil reutilizar la lógica de estado entre componentes, React no nos ofrece una forma de reutilizar diversas acciones como puedan ser conectarse a un store.

Sería posible hacerlo en base a utilizar diversos patrones, pero digamos que el resultado es una gran cantidad de líneas de código que encapsulan dicho comportamiento y que no son muy fáciles de leer. Con Hooks, puedes extraer lógica de estado de un componente de tal forma que este pueda ser probado y re-usado independientemente. Los Hooks te permiten reutilizar lógica de estado sin cambiar la jerarquía de tu componente.

Antes de continuar con los Hooks hay que dejar bien claro algunos aspectos:

- Los Hooks son opcionales el utilizarlo, aunque se recomienda su uso.
- React seguirá soportando las clases.
- Los Hooks no son un nuevo modelo de desarrollo sobre React sino que los afianza más, ya que dispones de una API más directa en la que se accede a elementos ya conocidos como props, state, ciclo de vida, etc
- No hace falta el migrar el código que tengamos ya implementado.

Tipos de Hooks

Los Hooks los podríamos agrupar en tres tipos:

- Hooks de estado => Cuando nuestro componente funcional se tiene la necesidad de hacer uso del state se podrá utilizar este tipo de Hook.
- Hooks de efecto => Agrega la capacidad de realizar efectos secundarios desde un componente funcional. Tiene el mismo propósito que componentDidMount, componentDidUpdate y componentWillUnmount en las clases React, pero unificadas en una sola API.
- Hooks Personalizado => En este tipo se puede ver la potencia que tienen los hooks, ya que este tipo de hooks se utilizan cuando se quiere reutilizar alguna lógica de estado entre componentes. Tradicionalmente, había dos soluciones populares para este problema: componente de orden superior y render props. Los Hooks personalizados te permiten hacer esto, pero sin agregar más componentes a tu árbol de jerarquía.

Para entender el funcionamiento de cada uno de ellos vamos a ver varios ejemplos de código que ayudan a entender mejor su funcionamiento.

Hooks de estado

Un Hook de estado podemos decir que su equivalente es

una clase funcional en la que tienen que hacer uso en algún momento del state. Pongamos por ejemplo el típico botón que incrementa el contador haciendo uso de lo que estábamos acostumbrados pondríamos un código como el siguiente:

```
demo.jsx > Example
1  class Example extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        count: 0
6      };
7    }
8
9    render() {
10     return (
11       <div>
12         <p>You clicked {this.state.count} times</p>
13         <button onClick={() => this.setState({ count: this.state.count + 1 })}>
14           Click me
15         </button>
16       </div>
17     );
18   }
19 }
```

Ahora bien, como haríamos el mismo código utilizando Hooks:

```
demo.jsx > Example
1  import React, { useState } from 'react';
2  function Example() {
3    const [count, setCount] = useState(0);
4    return (
5      <div>
6        <p>You clicked {count} times</p>
7        <button onClick={() => setCount(count + 1)}>
8          Click me
9        </button>
10     </div>
11   );
12 }
```

Dentro de este código tenemos varios elementos que comentar. En primer lugar, centrémonos en la línea 3: ¿Qué hace la llamada al useState? Declara una "variable de estado". Nuestra variable se llama count, pero podemos llamarla como queramos, por ejemplo, banana. Esta es una forma de "preservar" algunos valores entre las llamadas de la función. useState es una nueva forma de usar exactamente las mismas funciones que this.state nos da en una clase. Normalmente, las variables "desaparecen" cuando se sale de la función, pero las variables de estado son conservadas por React. ¿Qué parámetro espera la llamada del useState? El parámetro que espera es el valor de la iniciación de dicha variable de estado, puede ser valor entero, string o el tipo que necesitemos.

Ahora bien, dentro de la línea 3 todavía nos queda un valor del que no hemos hablado es el segundo valor del array, setCount. Este valor se refiere al nombre de la función en la que se va a cambiar el valor del estado cuando lo invoquemos. En este mismo ejemplo tal y como vemos en la línea 7 cada vez que se llama a la función setCount esta incrementa el valor de la variable de estado count +1.

¿Cómo podríamos definir múltiples variables de estado? La respuesta puede parecer obvia, aunque pueda parecer que tenga más miga de la que parece. La recomendación oficial es que por cada variable de estado que vayamos a utilizar creemos una línea algo similar a esto:

```
const [age, setAge] = useState(42);
const [fruit, setFruit] = useState('banana');
const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```

Hay otras opciones como puede ser añadiendo todas las variables en un único objeto donde están todos los valores de cada variable, esta opción es más complicada y no es recomendada ni por tema de rendimiento ni por complejidad del código. Algo similar a este código:

```
const [todos, setTodos] = useState([{ text: 'Learn Hooks',age:42, fruit:'banana' }]);
```

Hooks de efecto.

El Hook de efecto te permite llevar a cabo efectos secundarios en componentes funcionales. Hay dos tipos de efectos secundarios en los componentes de React: aquellos que no necesitan una operación de saneamiento y los que si la necesitan. Veamos la diferencia entre cada uno de ellos

Sin saneamiento => En ciertas ocasiones, queremos ejecutar código adicional después de que React haya actualizado el DOM. Peticiones de red, mutaciones manuales del DOM, y registros, son ejemplos comunes de efectos que no requieren una acción de saneamiento. Decimos esto porque podemos ejecutarlos y olvidarnos de ellos inmediatamente.

Ejemplo: Dado el ejemplo anterior queremos actualizar el título de nuestra página web con el valor del contador, para ello en un desarrollo tradicional añadiríamos el siguiente código:

```
demo.jsx > Example
1  class Example extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        count: 0
6      };
7    }
8    componentDidMount() {
9      document.title = `You clicked ${this.state.count} times`;
10   }
11   componentDidUpdate() {
12     document.title = `You clicked ${this.state.count} times`;
13   }
14   render() {
15     return (
16       <div>
17         <p>You clicked {this.state.count} times</p>
18         <button onClick={() => this.setState({ count: this.state.count + 1 })}>
19           Click me
20         </button>
21       </div>
22     );
23   }
24 }
```

Veamos ahora como podemos hacer lo mismo con el Hook useEffect:

```
demo.jsx > Example
1  import React, { useState, useEffect } from 'react';
2
3  function Example() {
4    const [count, setCount] = useState(0);
5
6    useEffect(() => {
7      document.title = `You clicked ${count} times`;
8    });
9
10   return (
11     <div>
12       <p>You clicked {count} times</p>
13       <button onClick={() => setCount(count + 1)}>
14         Click me
15       </button>
16     </div>
17   );
18 }
```

De los dos códigos, vamos a centrarnos primero en el tradicional. Como podéis observar tenemos dos métodos que estamos duplicando el mismo código. Esto se debe a que en muchas ocasiones queremos llevar a cabo el mismo efecto secundario sin importar si el componente acaba de montarse o si se ha actualizado. Conceptualmente, queremos que ocurra después de cada renderizado, pero las clases de React no tienen un método que haga eso. Podríamos extraer un método, pero aun así tendríamos que llamarlo en los dos sitios.

En el segundo código (con Hooks) centremos en useEffect. Al hacer uso de este Hook le estamos indicando a React que una vez termine de actualizar el DOM que realice la llamada a esta función. ¿Ahora bien, porque llamamos a useEffect dentro del propio componente? El motivo es que al estar en el mismo ámbito de la función podemos acceder directamente a la variable de estado y no hace falta utilizar ninguna API especifica.

Este hook viene a simular lo que anteriormente hacíamos con los métodos componentDidMount y componentDid Update, pero con una diferencia y que al hacer uso del hook no se bloquea la ejecución de la pantalla en el navegador. Eso si, con los hooks se ejecuta la función useEffect después de cada renderizado y de cada actualización del estado, aunque dicho comportamiento se puede modificar.

Con saneamiento=> El saneamiento lo podemos definir como la acción necesaria una vez el componente se va a destruir para evitar algún fallo en la aplicación. Pongamos un ejemplo, estamos implementando un chat y en el momento de que el usuario abandone el mismo queremos saberlo para cambiarle su estado y de esta forma evitar un comportamiento incorrecto de la aplicación. Veamos un ejemplo utilizando clases:

```
demo.jsx > FriendStatus > constructor
class FriendStatus extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isOnline: null };
    this.handleStatusChange = this.handleStatusChange.bind(this);
  }
  componentDidMount() {
    ChatAPI.subscribeToFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
  componentWillUnmount() {
    ChatAPI.unsubscribeFromFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }
  handleStatusChange(status) {
    this.setState({
      isOnline: status.isOnline
    });
  }
  render() {
    if (this.state.isOnline === null) {
      return 'Loading...';
    }
    return this.state.isOnline ? 'Online' : 'Offline';
  }
```

Como podéis observar controlamos tanto el evento cuando el componente se ha creado como cuando el componente se ha destruido. Ahora vamos a ver cómo nos quedaría haciendo uso de Hooks:

```
demo.jsx > FriendStatus
import React, { useState, useEffect } from 'react';

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    // Especifica como sanear este efecto:
    return function cleanup() {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

Como podéis ver en la función useEffect ahora nos devuelve una función tras ejecutar el código que se hace al montar. Este es un mecanismo opcional de los efectos. Todos los efectos pueden devolver una función que los sanea más tarde. Esto nos permite mantener la lógica de añadir/crear y eliminar suscripciones cerca la una de la otra, lo cual nos facilita mucho la legibilidad de este. ¿Cuándo se ejecuta la devolución del useEffect? Naturalmente cuando el componente se va “desmontar/destruir”, pero si el useEffect se ejecuta cada vez que el componente se renderiza es posible que tengamos problemas de rendimiento, ¿no? Correcto si no controlamos que el useEffect se ejecute cuando se haya realizado alguna modificación nuestro componente se volverá a pintar cada vez. Esto era uno de los errores más comunes que penalizan nuestros desarrollos en React como pudimos ver en el anterior artículo publicado en el número 38.

¿Cómo podemos hacer esto con los Hooks? Dentro del useEffects se dispone de un parámetro opcional en el que le podemos indicar que solo ejecute este método en caso de que una determinada variable o condición se ejecute. Por ejemplo:

```
demo.jsx > ...
useEffect(() => {
    document.title = `You clicked ${count} times`;
  }, [count]);
```

Hooks Personalizado

Una vez entendido el funcionamiento de los dos tipos de Hooks que disponemos, podremos crear nuestros propios Hooks con la finalidad de empezar a reutilizar la lógica entre los diversos componentes. Partiendo de la base del Hook de Efecto creado anteriormente vamos a extraer el código a un Hook Personalizado para poder reutilizarlo:

```
demo.jsx > useFriendStatus
import React, { useState, useEffect } from 'react';
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });
  return isOnline;
}
```

Los Hooks personalizados son más una convención que

una funcionalidad. Si el nombre de una función comienza con “use” y llama a otros Hooks, decimos que es un Hook personalizado.

> ***“Podremos crear nuestros propios Hooks con la finalidad de empezar a reutilizar la lógica entre los diversos componentes”***

Ahora vamos a crearnos dos compontes que van a utilizar nuestro Hook personalizado:

demo.jsx > FriendStatus

```jsx
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

demo.jsx > FriendListItem

```jsx
function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friend.id);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

En este ejemplo vemos como hemos podido aprovechar la lógica entre ambos componentes.

Reglas de los Hooks

Hooks son funciones de JavaScript, pero imponen dos reglas adicionales:

- Solo se pueden llamar Hooks en el nivel superior. No llames Hooks dentro de loops, condiciones o funciones anidadas.
- Solo llamar Hooks desde componentes funcionales de React. No llames Hooks desde las funciones regulares de JavaScript, salvo en los Hooks personalizados.

Hooks y SPFx

La verdad es que dentro del desarrollo en SPFx es el lugar idóneo para empezar a utilizar los Hooks. Como ya hemos comentado en anteriores ocasiones en los WebParts generalmente son aplicaciones pequeñas y más ligeras que vienen a extender alguna funcionalidad necesaria y puntual. Con lo que podemos decir que son aplicaciones más pequeñas y por regla general más sencillas (aunque no siempre sea así). Ahora bien, vamos a ver cómo podemos aprovecharnos de los beneficios que nos dan los Hooks. Para ello, en primer lugar, vamos a crearnos un hook Personalizado llamado useAvengerCollection. El funcionamiento de este hook es que nos va a devolver los elementos de una lista de SharePoint filtrados por un determinado valor en caso de que sea necesario. Para ello podemos utilizar el siguiente código:

```
import * as Re      setFilter: React.Dispatch<React.SetStateAction<string>>;
import {  Envi      setAvengerCollection: React.Dispatch<React.SetStateAction<any[]>>;
import { sp } }
export const useAvengerCollection = (type:EnvironmentType) => {
    const [filter, setFilter] = React.useState("");
    const [avengerCollection, setAvengerCollection] = React.useState([]);
    const loadAvengers = () => {
      if (type==EnvironmentType.Local)
      {
      fetch(`https://jsonplaceholder.typicode.com/users?name_like=${filter}`)
       .then(response => response.json())
       .then(json => setAvengerCollection(json));
      }
      else
      {
          if (filter=="")
          {
          sp.web.lists.getByTitle("AvengersList").items.select('Title', 'Id').get().then((items) => {
             let result:any[]=[];
             items.forEach(element => {result.push({"id":element["Id"], "name":element["Title"]
                    });
             });

             return result;
          })
          .then(json => setAvengerCollection(json));
      }
      else{
          sp.web.lists.getByTitle("AvengersList").items.select('Title', 'Id').filter("substringof('" + filter + "',Title)")
          .get().then((items) => {
             let result:any[]=[];
             items.forEach(element => {
                          result.push({
                    "id":element["Id"],
                   "name":element["Title"]
                    });
             });

             return result;
          })
          .then(json => setAvengerCollection(json));
      }
      }
    };
     return {avengerCollection, loadAvengers, filter,setFilter, setAvengerCollection};
    };
```

Del código en cuestión tenemos que ver cómo estamos gestionando el estado: por un lado, nos creamos la variable filter y setFilter para que se pueda utilizar desde otro Hook/artefacto que lo vaya a consumir. ¿De esta forma que beneficios obtenemos? El gran beneficio que obtenemos es que delegamos el control del ciclo de vida a nuestro componente y de esta forma evitamos que nuestro componente se vuelva a renderizar cada vez que haya una modificación del estado. Otro beneficio que tiene esto es que tenemos controlado cada elemento que se va a modificar y no se interfiere una modificación con otro.

La otra opción que tiene dicho código es que realizamos una llamada a la API REST de SharePoint (cuando estamos dentro de nuestro Sitio de SharePoint) y cuando obtenemos los datos de la API esta se almacena en el estado del componente.

¿Ahora ya veis que ganamos con todo esto? Tenemos un artefacto reutilizable y que gestiona el ciclo de React que se encarga de consumir datos de una lista de SharePoint y lo podemos reaprovechar y reutilizar en cualquier sitio de nuestra aplicación.

Una vez tenemos creado nuestro Hook, vamos a implementar un Hook de Efecto que lo consuma de forma que tengamos un componente que dado un campo sea capaz de realizar filtros sobre una lista de SharePoint para ello utilizaremos el useEffect. El código sería el siguiente:

src > webparts > hooks > components > LoadListTsx > ...

```tsx
import * as React from 'react';
import {useAvengerCollection} from './AvengersCollection';
import { Environment} from '@microsoft/sp-core-library';
export const LoadListComponent = () => {

  const {avengerCollection, loadAvengers, filter, setFilter} = useAvengerCollection(Environment.type);
  React.useEffect(() => {
    loadAvengers();
  }, [filter]);
  return (
    <div>
      <input value={filter} onChange={e => setFilter(e.target.value)} />
      <ul>
        {avengerCollection.map((user, index) => (
          <li key={index}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
};
```

De este Hook, que tenemos que comentar, pues en primer

lugar consumimos las variables que tiene el Hook Personalizado que hemos creado anteriormente. Utilizaremos el Hook de Efecto que cuando la variable de estado filter (obtenida del hook personalizado) se modifique se lanza la función que consume los datos de SharePoint.

¿Como podemos consumir este Hook LoadList? Pues llamándolo directamente desde nuestra clase de React de una forma muy simple:

```
src > webparts > hooks > components > Hooks.tsx > ...
1  import * as React from 'react';
2  import { IHooksProps } from './IHooksProps';
3  import {LoadListComponent} from './LoadList';
4  export default class Hooks extends React.Component<IHooksProps, {}> {
5    public render(): React.ReactElement<IHooksProps> {
6      return (
7        <LoadListComponent />
8      );
9    }
10 }
```

Conclusión

Los Hooks han levantado una gran cantidad de opiniones dentro de la comunidad de desarrolladores, aunque a primera vista pueda parecer que es una forma diferente de realizar las cosas. Creo que son una evolución natural de React para adaptarse a la potencia que ofrece JavaScript, siguiendo con los principios de React. Esto no quiere decir que no podamos usar nuestro desarrollo por clases de forma tradicional, sino que es una forma en el que el rendimiento de nuestra aplicación mejora y por otro parte nuestro código es más legible y mantenible.

Si nos ceñimos a los desarrollos en Spfx creo que los Hooks encajan como anillo al dedo dentro del ciclo de desarrollo. Dejan un código mucho más simple, más fácil de entender y sobre todo más mantenible.

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

28

PnP Provisioning - Tras la magia del TokenParser y localizando nuestras templates

En este artículo quiero enseñaros algo más de las tripas del Provisioning framework del PnP. A estas alturas, supongo que ya todos conocemos la librería PnP-Sites-Core, pero si no es el caso, que sepáis que esta librería de .NET nos ofrece utilidades y extensiones para trabajar con CSOM. Entre ellas, destaca por encima de todo el Framework de Provisioning, que nos permite provisionar artefactos en un site de SharePoint (ContentTypes, Listas, Ficheros, Páginas, etc), desde ficheros XML.

"La librería PnP-Sites-Core, pero si no es el caso nos ofrece utilidades y extensiones para trabajar con CSOM."

Tenéis más información, así como el código fuente de la librería, en la siguiente URL:

https://github.com/SharePoint/PnP-Sites-Core

De igual modo, el Schema XML del que se basa el framework de provisioning, lo podemos encontrar en la siguiente URL:

https://github.com/SharePoint/PnP-Provisioning-Schema

Hoy nos centraremos en el TokenParser, veremos cómo funciona, que tokens existen ya predefinidos, y veremos la importancia de éste a la hora de Localizar nuestras plantillas de provisioning (de hecho, comentaremos un pequeño issue del TokenParser a la hora de resolver Localized strings, y veremos un workaround rápido, y como resolverlo más correctamente)

Tu vecino y amigo, el TokenParser

Como os adelantaba, el TokenParser es una clase pública dentro del proyecto, que nos va a permitir poder usar tokens en nuestras templates XML de provisioning. Dichos tokens serán resueltos en tiempo de ejecución. Por ejemplo, existe un Token que permite resolver la URL relativa de una lista, a partir del título de esa lista. De esta forma, podemos registrar una CustomAction apuntando a una lista concreta (imagina una spfx extensión con un botón que sólo quieres para una lista concreta):

```
<pnp:CustomAction Name="MY_CA"
Title="Detail Panel Command Bar"
RegistrationId="{listid:My List Title}"
RegistrationType="List"
Location="ClientSideExtension.ListViewCommandSet.Com-
mandBar"
ClientSideComponentId="...
ClientSideComponentProperties="....." />
```

En tiempo de ejecución, el TokenParser sacará el ID de la lista con título "My List Title".

En la siguiente tabla, puedes encontrar algunos de los Tokens más utilizados:

TOKEN	DESCRIPTION	EXAMPLE	RETURNS
{contenttypeid:[contenttypename]}	Returns the ID of the specified content type	{contenttypeid:My Content Type}	0x0102004F51E-FDEA49C49668E-F9C6744C8CF87D
{fieldtitle:[internalname]}	Returns the title/displayname of a field given its internalname	{fieldtitle:LeaveEarly}	Leaving Early
{fileuniqueid:[siteRelativePath]}	Returns the unique id of a file which is being provisioned by the current template.	{fileuniqueid:/sitepages/home.aspx}	f2cd6d5b-1391-480e-a3dc-7f7f96137382
{guid}	Returns a newly generated GUID	{guid}	f2cd6d5b-1391-480e-a3dc-7f7f96137382
{keywordstermstoreid}	Returns a id of the default keywords term store	{keywordstermstoreid}	f2cd6d5b-1391-480e-a3dc-7f7f96137382
{listid:[name]}	Returns a id of the list given its name	{listid:My List}	f2cd6d5b-1391-480e-a3dc-7f7f96137382
{listurl:[name]}	Returns a site relative url of the list given its name	{listid:My List}	Lists/MyList
{localization:[key]}	Returns a value from a in the template provided resource file given the locale of the site that the template is applied to	{localization:MyListTitle}	My List Title
{masterpagecatalog}	Returns a server relative url of the master page catalog	{masterpagecatalog}	/sites/mysite/_catalogs/masterpage

{now}	Returns the current date in universal date time format: yyyy-MM-dd-THH:mm:ss.fffK	{now}	2018-04-18 T15:44:45.898+02:00
{parameter:[parametername]}	Returns the value of a parameter defined in the template	{parameter:MyParameter}	the value of the parameter
{site}	Returns the server relative url of the current site	{site}	/sites/mysitecollection/mysite
{sitecollection}	Returns the server relative url of the site collection	{sitecollection}	/sites/mysitecollection
{sitecollectiontermgroupid}	Returns the id of the site collection term group	{sitecollectiontermgroupid}	767bc144-e605-4d8c-885a-3a980feb39c6
{sitecollectiontermgroupname}	Returns the name of the site collection term group	{sitecollectiontermgroupname}	Site Collection - mytenant.sharepoint.com-sites-mysite
{sitecollectiontermsetid:[termsetname]}	Returns the id of the given termset name located in the sitecollection termgroup	{sitecollectiontermsetid:MyTermset}	9188a794-cfcf-48b6-9ac5-df2048e8aa5d
{sitecollectiontermstoreid}	Returns the id of the given default site collection term store	{sitecollectiontermstoreid}	9188a794-cfcf-48b6-9ac5-df2048e8aa5d
{termsetid:[groupname]:[termsetname]}	Returns the id of a term set given its name and its parent group	{termsetid:MyGroup:MyTermset}	9188a794-cfcf-48b6-9ac5-df2048e8aa5d
{termstoreid:[storename]}	Returns the id of a term store given its name	{termstoreid:MyTermStore}	9188a794-cfcf-48b6-9ac5-df2048e8aa5d
{themecatalog}	Returns the server relative url of the theme catalog	{themecatalog}	/sites/sitecollection/_catalogs/theme
{viewid:[listname],[viewname]}	Returns a id of the view given its name for a given list	{viewid:My List,My View}	f2cd6d5b-1391-480e-a3dc-7f7f96137382
{webpartid:[webpartname]}	Returns the id of a webpart that is being provisioned to a page through a template	{webpartid:mywebpart}	66e2b037-f749-402d-90b2-afd643850c26

Y en esta URL tenéis la tabla completa con todos los tokens soportados:

https://github.com/SharePoint/PnP-Sites-Core/blob/master/Core/ProvisioningEngineTokens.md

"Utilizando los Extensibility Handlers, podemos ampliar los Tokens y crear nuestros propios tokens, que serán resueltos en cualquier punto de la template."

Cada Token, posee su propia clase interna dentro del PnP, que hereda de una clase abstracta y pública llamada TokenDefinition. Cada Token definition, en su constructor recibe un string (o varios) con el token a sustituir:

```
[TokenDefinitionDescription(
   Token = "{masterpagecatalog}",
    Description = "Returns a server relative url of the master page catalog",
   Example = "{masterpagecatalog}",
   Returns = "/sites/mysite/_catalogs/masterpage")]
  internal class MasterPageCatalogToken : TokenDefinition
  {
    public MasterPageCatalogToken(Web web)
      : base(web, "{masterpagecatalog}")
    {
    }
```

Dentro de la clase, se sobrescribe el método GetReplaceValue, que es donde se aplica la lógica de reemplazo del token. Todas las definiciones de Token, se añaden a la clase TokenParser en su constructor:

```
    public TokenParser(Web web, ProvisioningTemplate template, ProvisioningTemplateApplyingInformation applyingInformation)
    {
      var tokenIds = ParseTemplate(template);

        web.EnsureProperties(w => w.ServerRelativeUrl, w => w.Url, w => w.Language);

      _web = web;

      _tokens = new List<TokenDefinition>();

      if (tokenIds.Contains("sitecollection"))
        _tokens.Add(new SiteCollectionToken(web));
```

Posteriormente, los ObjectHandlers utilizan el TokenParser para reemplazar los tokens que necesitan. Por ejemplo, aquí tenemos el ObjectHandler que se encarga de provisionar los ContenTypes, y que para darle el Name al ContentType, utiliza el TokenParser (por si dicho Name, en la XML template, se ha especificado utilizando algún Token):

```
existingContentType.Name = parser.ParseString(templateContentType.Name);
```

Utilizando los Extensibility Handlers, podemos ampliar los Tokens y crear nuestros propios tokens, que serán resueltos en cualquier punto de la template. Podéis encontrar más información en un artículo que escribí para el número 29 de la revista:

http://www.compartimoss.com/revistas/numero-29/custom-extensibility-handlers-para-el-framework-de-provisioning-del-pnp

Localizando nuestras templates

Gracias al TokenParser, podemos localizar nuestras plantillas PnP en diferentes lenguajes. Existe un Token "localization" (también puedes usarlo como: loc, localize, resource, res) que nos permite utilizar un fichero típico de resources

(.resx). Pare ello, en nuestra plantilla, hacemos referencia a los diferentes ficheros de resources que queremos usar:

```
<pnp:Localizations>
    <pnp:Localization LCID="1033" Name="English" Resource-
File="MyTemplate-en.resx"/>
    <pnp:Localization LCID="1043" Name="Dutch" ResourceFi-
le="MyTemplate-nl.resx"/>
</pnp:Localizations>
```

Y a la hora de aplicar el resource concreto, lo podemos hacer de la siguiente manera:

<Field ID="{23203E97-3BFE-40CB-AFB4-07AA2B86BF45}" Type="Text" Name="ProjectID" DisplayName="{resource:Myfield}" Group="Foundation.Columns" MaxLength="255" AllowDeletion="TRUE" />

Hasta aquí todo genial. Sin embargo, en plantillas grandes, es bastante común que queramos utilizar más de un set de ficheros de resources, me explico. Imagina que tus literales para los ContentTypes, los tienes en un set de ficheros de resources, ejemplo:

- ContentTypes.en-us.resx
- ContentTypes.es-es.resx
- ContentTypes.nl-nl.resx

Y los resources para el resto de tu plantilla, están en:

- Core.en-us.resx
- Core.es-es.resx
- Core.nl-nl.resx

Pues bien, si atendemos al esquema del PnP, podemos ver que precisamente tenemos un Name en el nodo Localization, y podemos hacer algo como:

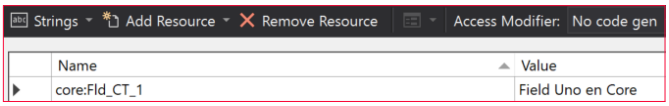
```
<pnp:Localizations>
    <pnp:Localization LCID="1033" Name="core" ResourceFi-
le="core.en-us.resx" />
    <pnp:Localization LCID="3082" Name="core" ResourceFi-
le="core.es-es.resx" />
    <pnp:Localization LCID="1033" Name="intranet" Resource-
File="intranet.en-us.resx" />
    <pnp:Localization LCID="3082" Name="intranet" Resource-
File="intranet.es-es.resx" />
</pnp:Localizations>
```

Y luego, utilizar el Token de la siguiente manera:

```
<Field ID="{23203E97-3BFE-40CB-AFB4-07AA2B86BF45}"
Type="Text" Name="ProjectID" DisplayName="{resource:-
core:Fld_CT_1}" Group="Foundation.Columns" MaxLeng-
th="255" AllowDeletion="TRUE" />
```

Sin embargo, el Código de provisioning, no va a funcionar correctamente, y no va a resolver el token. Esto es debido a que el TokenParser, cuando carga los tokens desde los ficheros de resources, asume que siempre va a haber un único "Name" en los nodos Localization.

Si necesitáis un workaround rápido, lo que podéis hacer es editar todas las Keys de cada fichero resx, y añadir el Name del fichero a cada key, por ejemplo:



Si en vuestro proyecto estáis usando el código del PnP, y queréis corregirlo ahí, entonces nos tenemos que ir a la clase TokenParser, y en el método:

```
private void AddResourceTokens(Web web, LocalizationCo-
llection localizations, FileConnectorBase connector)
```

Donde hace lo siguiente (alrededor de la línea 300):

```
resourceEntries.Add(new Tuple<string, uint, string>($"{entry.
Key}", (uint)localizationEntry.LCID, entry.Value.ToString().Re-
place("\"", "&quot;")));
```

Lo cambiamos para que cree el Token prefijado con el Name del fichero resx:

```
resourceEntries.Add(new Tuple<string, uint, string>($"{loca-
lizationEntry.Name}:{entry.Key}", (uint)localizationEntry.LCID,
entry.Value.ToString().Replace("\"", "&quot;")));
```

Esto hará que se resuelvan correctamente las entradas de tipo "{resource:core:Fld_CT_1}"

Tened en cuenta, sin embargo, que si sólo tenéis un fichero de resources, con este cambio siempre tenéis que usar el token con el Name del resx, es decir, esto funcionará "{resource:core:Fld_CT_1}" pero esto no lo hará "{resource:-Fld_CT_1}"

Podéis ver el siguiente Issue reportado en GitHub, que es exactamente lo que os estoy explicando aquí. Espero que, para la fecha de publicación del artículo, haya enviado una Pull Request con la solución que os comento, así que estad atentos al thread:

https://github.com/SharePoint/PnP-Provisioning-Schema/issues/301

Hasta el próximo artículo!

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

31

Entrevista Plain Concepts

Plain Concepts se fundó en 2006 por 4 MVPs de Microsoft, con el objetivo de desarrollar y facilitar a todo tipo de compañías la adopción de nuevas tecnologías destinadas a mejorar su productividad y procesos.

Premiados en 2016 como Microsoft Partner del Año, actualmente contamos con una plantilla superior a los 300 empleados, marcando un hito dentro del sector tecnológico al contar con 10 profesionales reconocidos como MVP de Microsoft y más de una decena de certificaciones a nivel empresarial.

Con presencia en USA, UAE, UK, España, Alemania y Holanda, hemos desarrollado más de 2000 proyectos para compañías pertenecientes a todo tipo de sectores.

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

Plan Concepts surge a raíz de un viaje iniciático en la costa oeste de EEUU de 4 amigos en 2006, todos ellos MVPs de Microsoft, es decir, todos los socios de la empresa son perfiles técnicos. Y aunque hoy ya somos más de 300 empleados, el 90% de la plantilla son perfiles técnicos.

¿Cuáles son las principales actividades tecnológicas hoy en día?

Tenemos expertos en todas las áreas, desde Cloud, Inteligencia Artificial, Big Data, IoT, Web, Apps, Blockchain, Virtual and Mixed Reality a Office 365.

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

No solo nos apasiona la tecnología, todos los años durante el mes de junio tiene lugar la Plain Camp, lo que significa que todas las oficinas nos reunimos un fin de semana para realizar diferentes actividades juntos, playa, montaña, aventura ¡nunca se sabe! También participamos en carreras benéficas, tenemos clases de inglés, de oratoria y en general, nos preocupamos y ocupamos del bienestar de nuestros empleados. De hecho, creemos que esta filosofía es la que ha ayudado a lograr por segundo año consecutivo ser reconocidos como un Best Place to Work.

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

Si hay algo que de verdad nos caracteriza es que nos encanta organizar y participar en todo tipo de eventos técnicos, ya sea como patrocinadores, ponentes o colaboradores, porque nos apasiona la tecnología. Para ser más exactos, en lo que llevamos de año tenemos previsto participar en 90 eventos de 26 ciudades diferentes.

También cedemos espacios en nuestras oficinas para hacer Meetups, tenemos nuestro canal de Youtube PlainTV donde publicamos periódicamente entrevistas a nuestros empleados y desde hace dos años somos los organizadores del principal evento de tecnología .Net en España, la DotNet.

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

Creemos que en los próximos años van a predominar las soluciones relacionadas con la Inteligencia artificial. Es la tecnología por la que están apostando las grandes empresas y aunque todavía está poco madura, es la que creemos que sin duda tendrá mayor recorrido. Por eso estamos organizando un evento anual enfocado exclusivamente en la Inteligencia Artificial que tendrá lugar en Barcelona el 28 de noviembre: Singularity Tech Day.

32

Securizando tus apis con Azure Api Management y Oauth

El catálogo de Apis securizado con Azure Api Management, Azure AD y Oauth en tu APIM. Azure API Management como Middelware de tus servicios

Seguramente todos hemos visto o probado a estas alturas Azure Api Management, y hemos podido observar toda la funcionalidad que nos aporta a nivel de Publicación de APIs, o lo que Microsoft denomina la funcionalidad de FACHADA. En este artículo nos queremos centrar en esa otra visión de Middleware que nos aporta APIM, y sobre todo en como autenticar nuestros servicios y validar los TOKEN que nos hacen llegar los clientes de nuestros servicios.

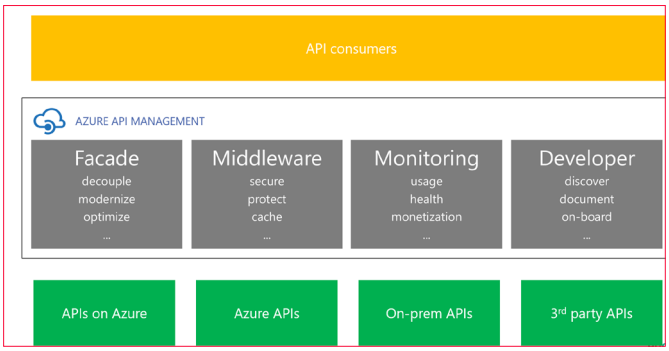


Imagen 1.- APIM features.

Lo más básico, autenticación por Clave de Suscripción

Antes de nada, tenemos que conocer cómo se estructura un catálogo de APIs en APIM. Lo primero que debemos hacer antes de publicar un API es crear un Producto, por defecto una instalación de APIM desde cero nos proporcionan dos productos Starter y Unlimited. A este producto le vamos a poder asignar Políticas, APIs, y Suscripciones.

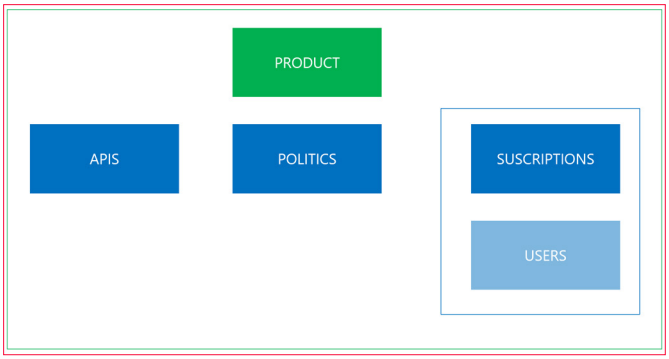


Imagen 2.- Jerarquía en APIM,

Es muy importante el concepto Suscripción, ya que nos permita relacionar a los productos con los usuarios, y por tanto darles acceso a nuestros clientes a una serie de APIS en concreto. Para crear una suscripción por ejemplo en el producto Starter, accedemos al producto y posteriormente seleccionamos el apartado Suscription, tal cual vemos en la imagen.

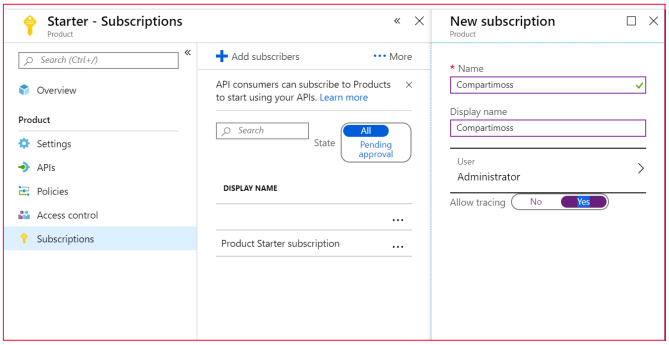


Imagen 3.- Añadir una suscripción.

Deberemos darle un nombre a la suscripción, y añadir un usuario propietario de esta suscripción. Evidentemente no debemos crear una suscripción por usuario, aunque es viable, podemos crear una suscripción por grupos de trabajo y entorno y así facilitar el gobierno de estas. Lo importante de crear esta suscripción, es que se van a generar un par de claves privadas y nominales a esta suscripción – usuario, que van a ser necesarias para poder llamar a las APIs de este producto.

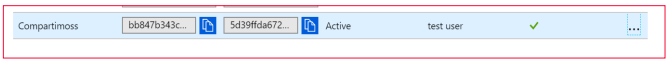


Imagen 4.- Clave de suscripción.

El último paso para securizar nuestro api por clave de suscripción, es acceder al apartado de APIs, y sobre un API en concreto en el apartado Settings, observamos que está marcada la opción “Subscription Required” en el apartado Suscription.

“Lo primero que debemos hacer antes de publicar un API es crear un Producto, por defecto una instalación de APIM desde cero nos proporcionan dos productos Starter y Unlimited”

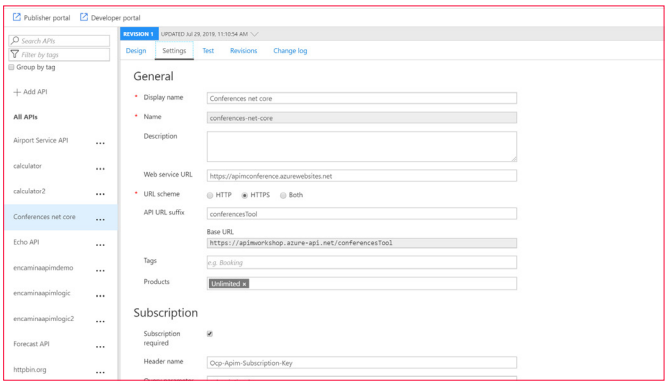


Imagen 5.- Configurando el API.

Si revisamos la imagen vemos que hemos marcado el api para que pida un suscription key válido y para ello en las cabeceras de la petición deberemos incorporar la clave de suscripción de la forma "Ocp-Apim-Subscription-Key : key", o fallará tal cual vemos en la siguiente imagen.

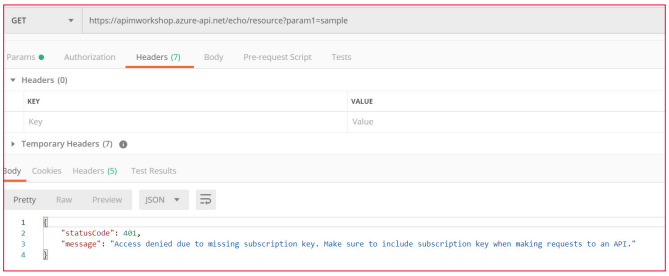


Imagen 6.- Error 401, suscription key.

En la imagen anterior vemos una petición desde Postman a un api de nuestro catálogo, en este caso a un api por defecto que se genera en el servicio que se llama Echo API. Como vemos, no le hemos pasado ninguna clave de suscripción. Por el contrario, si añadimos una clave válida obtenemos el valor de forma correcta.

La forma más fácil de aprender a consultar nuestras APIS con las claves correctas es dentro de un API publicada, seleccionamos uno de sus métodos, y hacemos uso de la pestaña de test.

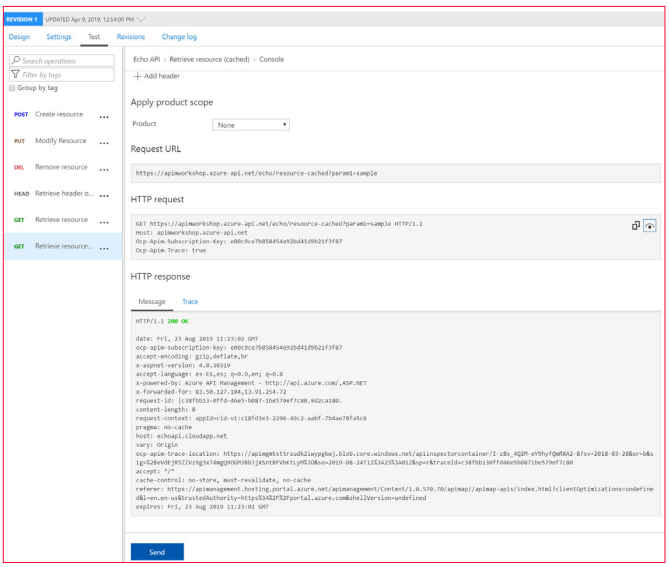


Imagen 7.- Testeando la API.

Aquí podremos elegir el producto con el que consultar el API, nos inserta el propio APIM la clave de suscripción correcta, y nos deja un ejemplo de petición y de respuesta que podemos simular posteriormente desde Postman o desde nuestra aplicación cliente.

Algo más complejo, Azure AD y Oauth en APIM

Como hemos visto con clave de suscripción podemos securizar de una forma mínima nuestras APIs, pero si queremos, por ejemplo, securizar nuestros APIS con OAuth y Azure AD también podemos hacerlo, e implantarlo como solución de seguridad base de nuestros servicios. Para comenzar debemos localizar en APIM el menú el apartado "Seguridad" y veremos que hay varias opciones, entre la que encontramos OAuth 2.0

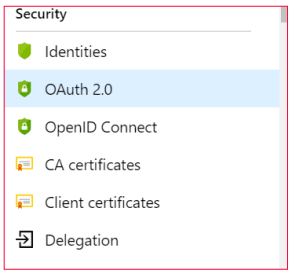


Imagen 8.- Security en APIM.

Ahora vamos a configurar nuestro OAuth 2.0 rellenando el formulario que se nos presenta, teniendo en cuenta de donde ir sacando los parámetros de configuración:

- Display name: Campo libre, ponemos un nombre descriptivo para nosotros.
- Client registration URL: no vamos a permitir el registro de aplicaciones, así que dejaremos htpp://localhost.
- Marcamos la opción "Authorization code".

Hasta aquí todo fácil, pero los siguientes parámetros necesitan que demos de alta una Aplicación en nuestro directorio Activo, como sería mucho alargar este artículo os dejo un enlace de Microsoft para poder introducirnos en cómo hacerlo https://docs.microsoft.com/es-es/azure/active-directory/manage-apps/

Del apartado Endpoints de nuestra Aplicación creada en Azure AD, vamos a obtener los datos para seguir configurando nuestro OAuth en APIM:

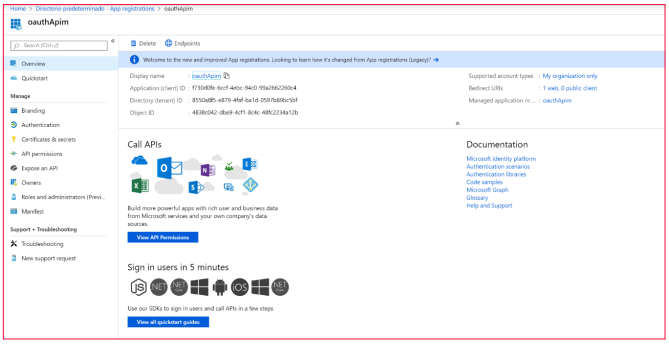


Imagen 9.- Overview de la APIM creada.

- Autorization endpoint URL:

- Token endpoint Url

- Client authentication mode: Marcamos la opción “In the body”.
- Client Id: Lo obtenemos de nuestra aplicación creada en AD, en Overview.
- Client secret: Lo obtenemos creando una clave privada, en el apartado Client & Secret de nuestra aplicación en AD.

Nos quedará una configuración parecida a la de la imagen siguiente, y si no hemos tenido problemas pulsamos en crear y se creará nuestro OAuth 2.0 en APIM.

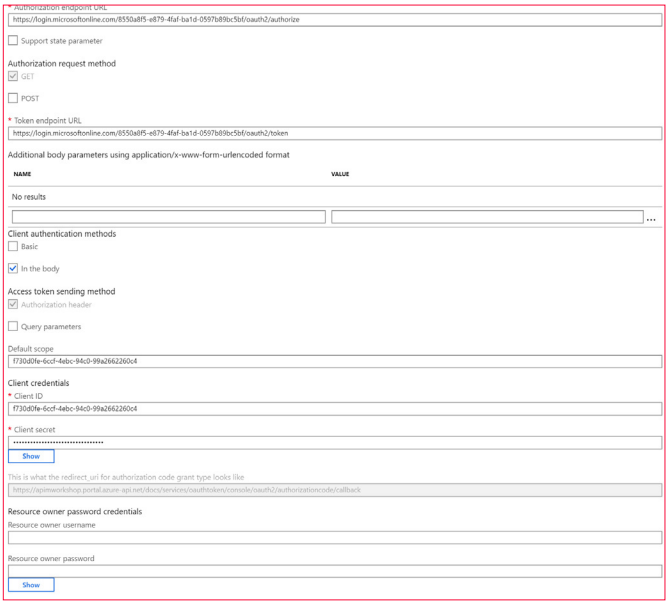


Imagen 10.- Oauth 2.0 en APIM.

Probando la generación del token desde el Portal del desarrollador

Para poder ver si se genera el Token de forma correcta, debemos acceder lo primero en un API, y, en la configuración, seleccionar la conexión OAuth 2.0 que hemos creado en el punto anterior.

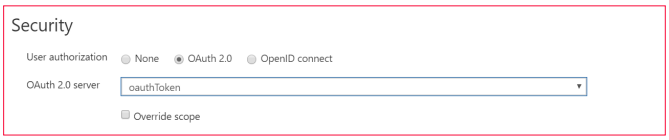


Imagen 11.- Seguridad en nuestro API.

Ahora si podemos acceder al Portal del desarrollador, buscar este API en concreto y probarla.

Nos encontraremos que además de la clave de suscripción que configuramos al inicio ahora además nos permite añadirle un token de OAuth.

“Es muy importante el concepto Suscripción, ya que nos permita relacionar a los productos con los usuarios, y por tanto darles acceso a nuestros clientes a una serie de APIS en concreto”

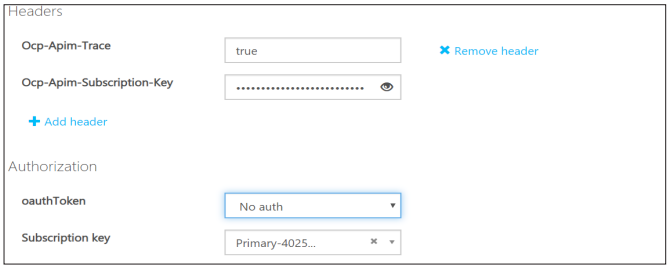


Imagen 12.- Testing Oauth developer portal.

Seleccionamos en OauthToken nuestro token y se nos abrirá una pantalla emergente que nos pedirá usuario y contraseña para logarnos contra nuestra aplicación de AD. Si las credenciales son válidas y no falla nada en la configuración veremos que se genera un token valido y que APIM le hará llegar en la cabecera Authorization a nuestras APIS.

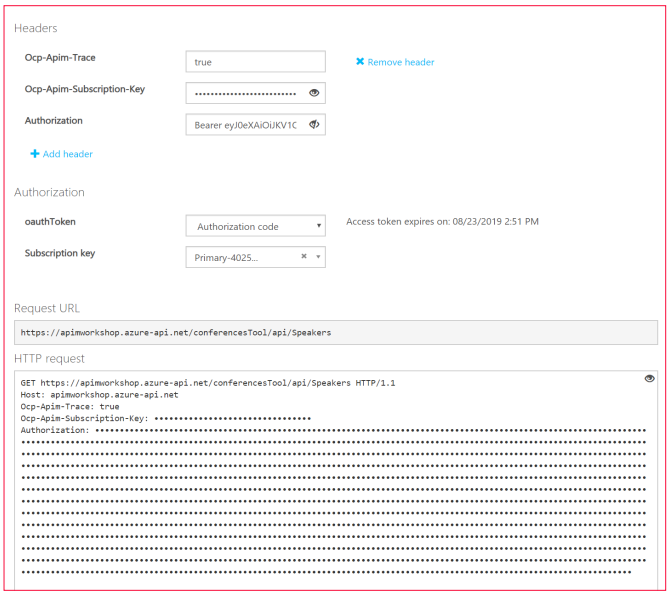


Imagen 13.- Nuestro primer token en APIM.

En este punto APIM no valida el token, simplemente lo genera y se los hace llegar a nuestros servicios que, si están configurados en nuestro mismo tenant, podrá entender el token y autorizar las llamadas por OAuth. Si damos a enviar la petición y nos centramos en el apartado Trace en los datos de entrada veremos todo lo que le hacemos llegar a nuestro backend desde el portal del desarrollador, y vemos que le hacemos llegar un token generado por nuestra conexión de OAuth 2.0 en APIM

Imagen 14.- Trace inbound.

En el siguiente punto veremos cómo hacer que APIM si valide el token, y hace de proxy de seguridad entre nuestros backend y nuestras aplicaciones cliente.

¿Qué nos falta para que APIM valide nuestro token? – Políticas de validación

Como hemos adelantado, por defecto APIM no valida el token de nuestras APIS, simplemente se queda en generarlo de forma correcta y aportar a los equipos de trabajo las herramientas para poder autenticar las API, que no es poco porque antes proporcionar token de autenticación a distintos equipos de trabajo cuanto menos era costoso y en algunos casos imposible, por el mantenimiento que esto conllevaba.

"La forma más fácil de aprender a consultar nuestras APIS con las claves correctas es dentro de un API publicada, seleccionamos uno de sus métodos, y hacemos uso de la pestaña de test."

Pero no nos queremos quedar aquí y vamos a añadir una política de validación de JWT que abra nuestro token y nos diga si viene de una aplicación de confianza. Para ello sobre el API que hemos configurado el OAuth, en la pestaña de design vamos a añadir una política de la siguiente forma:

1.– Añadimos una nueva política en la parte de Inbound para que quede como la imagen:

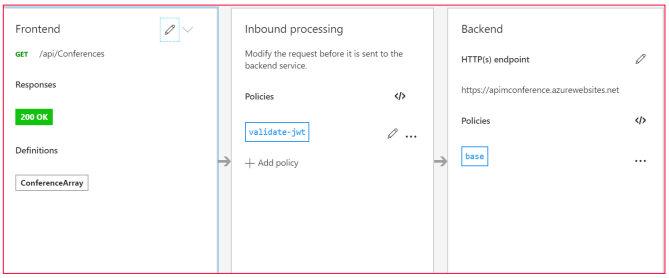


Imagen 15.- Inbound jwt.

2.– Añadimos el siguiente código a nuestra política:

```
<inbound>
        <validate-jwt header-name="Authorization" failed-validation-httpcode="401"  failed-validation-error-message="Unauthorized. Access token invalid" require-expiration-time="false" require-signed-tokens="false">
      <openid-config url="https://login.microsoftonline.com//<DirectorytenantID>/v2.0/.well-known/openid-configuration" />
        <issuers>
          <issuer>https://sts.windows.net/<DirectorytenantID>
        </issuers>
</validate-jwt>
```

3.– Validamos la política y si es correcto lo guardamos. Si analizamos el código que hemos añadido, necesitamos simplemente añadir el ID de nuestro tenant, y esta política validará contra nuestro tenant si el token es válido, si no lo es dará un 401.

Si volvemos al portal de desarrollador, y probamos el método al que hemos añadido la política veremos que, si no indicamos el token de OAuth, nos devuelve el error que hemos configurado en la política.

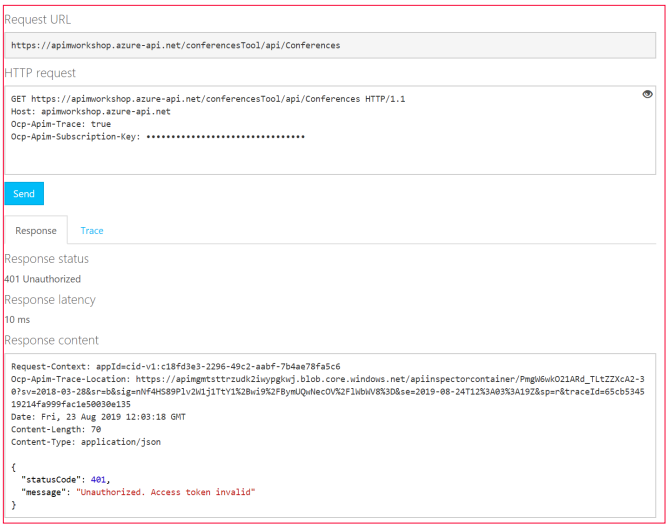


Imagen 16.- Error generado si no se indica el token.

Con esto ya tenemos nuestro API securizado con clave de suscripción, y además con un token de Azure AD por OAuth 2.0

Una seña más de que Azure Api Management ayuda al desarrollo y al desarrollador

Siempre que programamos y más si somos la parte backend, no nos paramos mucho a pensar lo complejo que resulta generar un token válido, con los scope necesarios para que el usuario cliente pueda obtener contexto y sus datos desde nuestras APIS. Con Azure Api Management, tenemos una herramienta más para que los clientes de nuestras APIS desde un portal 100% accesible para cualquiera, pueda obtener su token, simular las llamadas y llevárselas a su código y poder hacer las pruebas necesarias.

Sin duda para mí esto es un avance y un ahorro en coste, tiempo y, sobre todo, malentendidos entre los distintos equipos de trabajo de un proyecto.

SERGIO HERNANDEZ MANCEBO
Principal Team Leader en Encamina | Azure MVP

36

Un vistazo a Microsoft Graph Toolkit: Web Components con Graph

En muchos proyectos nos piden funcionalidades como mostrar las tareas de Planner del usuario, o la agenda del usuario, y para ello utilizamos Microsoft Graph. Para facilitar el acceso a estas informaciones, Microsoft ha creado una serie de componentes que podemos utilizar en nuestras aplicaciones web y que de una forma muy sencilla permite implementar estas funcionalidades.

"Lo bueno que tiene Microsoft Graph Toolkit es que lo podemos utilizar en aplicaciones web modernas, WebParts de SharePoint Framework (SPFx), tabs de Teams, etc"

Lo bueno que tiene Microsoft Graph Toolkit es que lo podemos utilizar en aplicaciones web modernas, WebParts de SharePoint Framework (SPFx), tabs de Teams, etc. Y utilizando cualquier framework, ReactJS, Angular, etc.

¿Cómo empiezo a trabajar con Microsoft Graph Toolkit?

Tenemos 2 formas para poder utilizar Microsoft Graph Toolkit en nuestros desarrollos:

- Cargándolo directamente en nuestra aplicación referenciando un fichero JavaScript:

```
<script src="https://unpkg.com/@microsoft/mgt/dist/bundle/mgt-loader.js"></script>
```

- Mediante NPM, por lo que lo primero sería agregar el paquete NPM con:

```
npm install @microsoft/mgt
```

y luego ya podríamos referenciarlo:

```
<script src="node_modules/@microsoft/mgt/dist/es6/components.js"></script>
```

Componentes

- Inicio de sesión: Este componente facilita la autenticación del usuario. Tiene 2 estados:
  - Si el usuario no ha iniciado sesión, el control es un botón sencillo para iniciar el proceso de inicio de sesión.
  - Si el usuario ha iniciado sesión, el control muestra el nombre de usuario de la sesión actual, la imagen de perfil y el correo electrónico. Cuando se hace clic en él, se abre un control flotante con un comando para cerrar la sesión.

  Para utilizarlo lo único que debemos hacer es incrustarlo en nuestro html:

```
<mgt-login></mgt-login>
```

Si no nos hemos autenticado, nos mostrará el botón para autenticarnos y una vez autenticado, nos mostrará nuestro nombre y nuestra imagen donde podemos pulsar donde nos mostrará nuestro email y un enlace para cerrar la sesión.

Imagen 1.- Componente inicio de sesión sin iniciar sesión

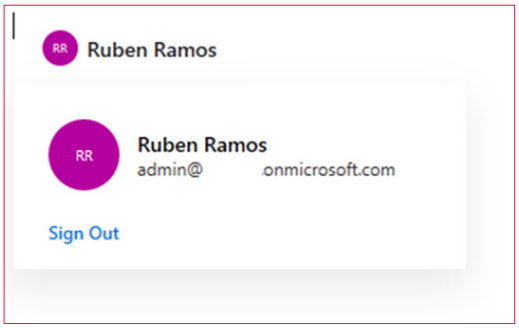


Imagen 2.- Componente inicio de sesión con sesión iniciada

Este componente tiene los siguientes eventos que podemos capturar:

EVENTO	DESCRIPCIÓN
loginInitiated	El usuario hizo clic en el botón iniciar sesión para iniciar el proceso de inicio de sesión.

loginCompleted	el proceso de inicio de sesión se realizó correctamente y el usuario ha iniciado sesión.
loginFailed	El usuario canceló el proceso de inicio de sesión o no pudo iniciar sesión.
logoutInitiated	El usuario empezó a cerrar sesión.
logoutCompleted	El usuario ha cerrado la sesión.

- Contacto: Este componente se utiliza para mostrar datos de una persona. Para poder elegir el contacto que queremos mostrar tenemos 3 opciones distintas

PROPIEDAD	ATRIBUTO	DESCRIPCIÓN
userId	user-id	Obtener el usuario de Microsoft Graph mediante su identificador
personQuery	person-query	Buscar una persona determinada en Microsoft Graph. Elegirá la primera persona disponible y obtendrá los detalles de la persona
personDetails	person-details	Establecer manualmente los detalles de la persona

Por ejemplo, si queremos mostrar los datos del usuario que se ha validado utilizaremos:

```
<mgt-person person-query=”me” show-name show-email></
mgt-person>
```

Y nos mostrará los datos del usuario autenticado:

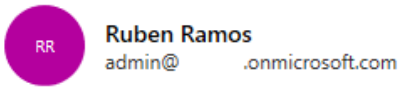


Imagen 3.- Componente Contacto

También tiene una serie de propiedades con las que podemos indicar si queremos mostrar el nombre y/o el mail del contacto.

PROPIEDAD	ATRIBUTO	DESCRIPCIÓN
showName	show-name	Establecer marca para mostrar el nombre para mostrar de la persona
showEmail	show-email	Establecer marca para mostrar correo electrónico de la persona

En la siguiente tabla vemos las llamadas que realiza a la API de Graph y los permisos que necesita para funcionar:

RECURSO	PERMISO/ÁMBITO
/me	User.Read
$value/me/Photo/	User.Read
/me/People/? $search =	People.Read
/me/contacts/*	Contacts.Read
$value/users/{ID}/Photo/	User.ReadBasic.All

- Contactos: Este componente se utiliza para mostrar datos de un grupo de personas. Este componente por defecto muestra los contactos frecuentes del usuario, es decir, realiza la llamada /me/people a la API de Graph, por ejemplo, si añadimos:

```
<mgt-people></mgt-people>
```

Nos mostrará nuestros contactos recientes:

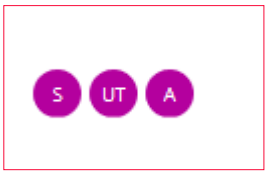


Imagen 4.- Componente Contactos

Este control lo podemos personalizar con las siguientes propiedades:

PROPIEDAD	ATRIBUTO	DESCRIPCIÓN
showMax	show-max	Indica el número máximo de usuarios que se van a mostrar. El valor predeterminado es 3.
people	people	Una matriz de personas para obtener o establecer la lista de personas que representa el componente. Utilice esta propiedad para tener acceso a las personas cargadas por el componente. Establezca este valor para cargar sus propios usuarios.

En la siguiente tabla vemos las llamadas que realiza a la API de Graph y los permisos que necesita para funcionar:

RECURSO	PERMISO/ÁMBITO
/me/people	People.Read

- Agenda: Este componente representa los eventos en un calendario de usuario o grupo. De forma predeterminada, el calendario muestra los eventos del usuario que ha iniciado sesión para el día actual, pero se puede modificar para mostrar los datos de otro endpoint que devuelva eventos, por ejemplo, /groups/{id}/calendar/calendarView

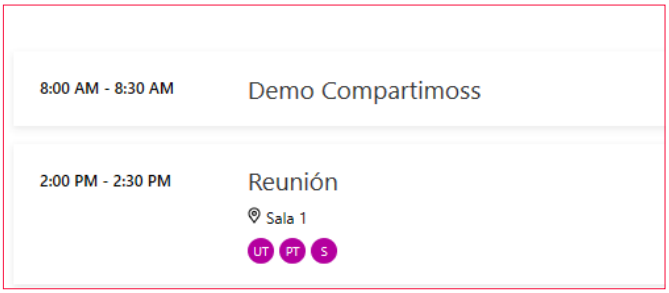


Imagen 5.- Componente Agenda

“Para poder realizar las llamadas a Microsoft Graph necesitamos un token de acceso. Microsoft Graph Toolkit proporciona una serie de proveedores que facilitan la adquisición del token necesario”

Este control lo podemos personalizar con las siguientes propiedades:

PROPIEDAD	ATRIBUTO	DESCRIPCIÓN
groupByDay	group-by-day	Un valor booleano para agrupar eventos por eventos por día de forma predeterminada no está agrupado.
date	date	Una cadena que representa la fecha de inicio de los eventos que se van a obtener de Microsoft Graph. El valor debe estar en un formato que se pueda analizar mediante la fecha el valor del constructor no tiene ningún event-query efecto si se establece el atributo.
days	days	Un número de días para obtener de Microsoft Graph: el valor predeterminado es 3-Value no tiene event-query efecto si se establece el atributo.
eventQuery	event-query	Cadena que representa una consulta alternativa que se va a usar al recuperar eventos de Microsoft Graph. Si lo desea, puede Agregar el ámbito delegado al final de la cadena mediante su delimitación con | ("/groups/GROUP-ID-GUID/calendar/calendarView | group.read.all").
events	events	Una matriz de eventos para obtener o establecer la lista de eventos que representa el componente: Utilice esta propiedad para tener acceso a los eventos cargados por el componente.

En la siguiente tabla vemos las llamadas que realiza a la API de Graph y los permisos que necesita para funcionar:

RESOURCE	PERMISO/ÁMBITO
/me/calendarview	`Calendars.Read`

- Tareas: Con este control podemos visualizar, añadir, editar y eliminar las tareas que tengamos en Planner o Microsoft ToDo, dependiendo del datasource que le indiquemos. Para utilizarlo tendremos que añadir

```
<mgt-tasks></mgt-tasks>
```

Y nos mostrará las tareas que tengamos asignadas, por defecto, nos mostrará las que tengamos asignadas en Planner:

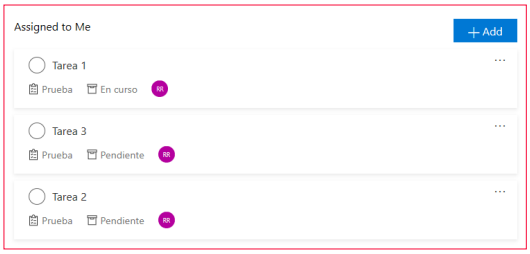


Imagen 6.- Componente Tareas.

Este control lo podemos personalizar con las siguientes propiedades:

PROPIEDAD	ATRIBUTO	DESCRIPCIÓN
dataSource	data-source="todo/planner"	Establece el origen de datos para las tareas, ya sea Microsoft to-do o Microsoft Planner. El valor predeterminado es planner.
readOnly	read-only	Establece que la interfaz de tareas sea de solo lectura (sin agregar ni quitar tareas). El valor predeterminado es false.
initialId	initial-id="planner_id/folder_id"	Establece la carpeta o el planificador mostrado inicialmente en el identificador proporcionado.
initialBucketId	initial-bucket-id="-bucket_id"	Establece el depósito que se muestra inicialmente (sólo origen de datos de Planner) en el identificador proporcionado.
targetId	target-id="planner_id/folder_id"	Bloquea la interfaz de tareas con el planificador o identificador de carpeta proporcionado.
targetBucketId	target-bucket-id="-bucket_id"	Bloquea la interfaz de tareas con el identificador de depósito proporcionado (sólo origen de datos de Planner).

En la siguiente tabla vemos las llamadas que realiza a la API de Graph y los permisos que necesita para funcionar

RECURSO	PERMISO/ÁMBITO
/me/planner/plans	Group.Read.All
/Planner/Plans/$ {ID}	Group.Read.All, Group.ReadWrite.All
/planner/tasks	Group.ReadWrite.All
/me/outlook/taskGroups	Tasks.Read
/me/outlook/taskFolders	Tasks.Read, Tasks.ReadWrite
/me/outlook/tasks	Tasks.ReadWrite

Proveedores

Para poder realizar las llamadas a Microsoft Graph necesitamos un token de acceso. Microsoft Graph Toolkit proporciona una serie de proveedores que facilitan la adquisición del token necesario para realizar las estas llamadas a Microsoft Graph.

Para que los componentes usen un proveedor, se debe establecer la propiedad Providers.globalProvider con el proveedor que necesitamos. Debemos tener en cuenta que todos los componentes que incluyamos utilizarán ese proveedor.

Actualmente existen los siguientes proveedores:

- MsalProvider: Este proveedor utiliza msal.js para obtener el token de acceso, por ejemplo, podemos utilizar este proveedor si estamos desarrollando una aplicación SPA. Cuando utilizamos este proveedor lo primero que tenemos que hacer es registrar nuestra aplicación dentro de Azure AD. En este enlace explican cómo hacerlo https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app

  Una vez registrada nuestra aplicación debemos inicializar el proveedor, para esto tenemos 2 opciones:

  - Inicializando desde HTML usando el componente mgt-msal-provider y estableciendo el clientID que obtenemos después de registrar nuestra aplicación en Azure AD. Por ejemplo:

```
<mgt-msal-provider client-id=" <YOUR_CLIENT_ID> "></
mgt-msal-provider>
```

- Iniciando desde JavaScript/Typescript de la siguiente manera:

```
import {Providers, MsalProvider} from '@microsoft/mgt'
import {UserAgentApplication} from "msal";

Providers.globalProvider = new MsalProvider(config: Msal-
Config);
```

Donde MsalConfig es:

```
interface MsalConfig {
 clientId: string;
 scopes?: string[];
 authority?: string;
 loginType?: LoginType;
 options?: Configuration;
}
```

- SharePointProvider: Utilizaremos este proveedor si lo que estamos desarrollando es un WebPart de SharePoint. Para utilizarlo debemos inicializar el proveedor dentro del método onInit.

```
import {Providers, SharePointProvider} from '@microsoft/
mgt';

protected async onInit() {
    Providers.globalProvider = new SharePointProvider(this.
context);
```

Una vez inicializado podemos añadir nuestro componente en el método render. Debemos tener en cuenta que Microsoft Graph Toolkit requiere Typescript 3.x, por lo que necesitamos estar utilizando como mínimo, la versión 1.8 de SPFx donde se añadió soporte a esta versión de Typescript (https://github.com/SharePoint/sp-dev-docs/wiki/SharePoint-Framework-v1.8-release-notes#support-for-typescript-27-29-and-3x)

- TeamsProvider: Utilizaremos este proveedor si estamos desarrollando una pestaña para Teams. Para utilizarlo lo primero que tenemos que hacer es asegurarnos que tenemos referenciado el SDK de Teams y luego inicializamos el proveedor.

```
<script src="https://unpkg.com/@microsoft/teams-js/dist/Mi-
crosoftTeams.min.js" crossorigin="anonymous"></script>
<script src="https://unpkg.com/@microsoft/mgt/dist/bundle/
mgt-loader.js"></script>

<mgt-teams-provider
 client-id="<YOUR_CLIENT_ID>"
    auth-popup-url="https://<YOUR-DOMAIN>.com/AUTH-PA-
TH">
</mgt-teams-provider>
```

También podemos utilizarlo mediante paquetes NPM, para ello necesitamos instalar los paquetes de Microsoft Graph Toolkit y el SDK de Teams.

```
npm install @microsoft/mgt @microsoft/teams-js
```

Una vez instalado podremos inicializarlo de la siguiente manera:

```
import '@microsoft/teams-js';
import {Providers, TeamsProvider} from '@microsoft/mgt';
Providers.globalProvider = new TeamsProvider(config);
```

Donde config es:

```
export interface TeamsConfig {
 clientId: string;
 authPopupUrl: string;
 scopes?: string[];
 msalOptions?: Configuration;
}
```

- Próximamente estará disponible un proveedor para complementos de Office.
- También podemos crear nuestro proveedor personalizado si necesitamos algo más específico. Podemos crearlo de 2 formas:
  - Creando un SimpleProvider que obtenga el token de acceso.
  - Ampliando la clase abstracta IProvider.

Cambiando el estilo

Cada componente posee sus propias propiedades CSS personalizadas que podemos modificar, por ejemplo, para el componente mgt-people podemos modificar las siguientes propiedades.

```
mgt-people {
 --list-margin: 8px 4px 8px; /* Margin for component */
 --avatar-margin: 0 4px 0 0; /* Margin for each person */
}
```

Si necesitamos más personalización en los componentes tendremos que hacer plantillas.

"Cada componente posee sus propias propiedades CSS personalizadas que podemos modificar, por ejemplo, para el componente mgt-people podemos modificar las siguientes propiedades"

Plantillas

Para todos los componentes podemos utilizar plantillas personalizadas para modificar la forma de mostrar la información. Para hacerlo lo que tendremos que hacer es añadir el elemento <template> dentro del componente que queremos modificar. Por ejemplo, para modificar la plantilla en el componente mgt-agenda lo haríamos de la

siguiente manera:

```
<mgt-agenda>
  <template data-type="event">
   <div>{{event.subject}}</div>
   <div data-for='attendee in event.attendees'>
      <mgt-person person-query="{{attendee.emailAddress.
name}}">
      <template>
       <div data-if="person.image">
        <img src="{{person.image}}" />
       </div>
       <div data-else>
        {{person.displayName}}
       </div>
      </template>
     </mgt-person>
   </div>
  </template>
</mgt-agenda>
```

Cuando creamos plantillas tenemos que tener en cuenta varias cosas:

- Debemos usar llaves dobles ({{expression}}) para expandir una expresión.
- Podemos usar los atributos data-if y data-else y para la representación condicional.
- Podemos usar el atributo data-for para repetir un elemento.
- Podemos usar el atributo data-type para especificar la parte del componente a la que se va a plantilla. Si no se especifica el tipo, la plantilla se aplicará a todo el componente.

Conclusión

Microsoft Graph Toolkit nos proporciona una serie de elementos web reutilizables que podemos utilizar en nuestros desarrollos en distintos escenarios bastante comunes cuando trabajamos con Microsoft Graph. Además, podemos utilizarlo en cualquier framework que estemos utilizando.

Debemos tener en cuenta que es una versión preview y que no es aconsejable que se utilice en entorno de producción. Recientemente se ha liberado la versión 0.2.0, incluyendo un control PeoplePicker y solucionando algunos errores.

Si quieres más información puedes ver la documentación oficial en https://docs.microsoft.com/es-es/graph/toolkit/overview y en el proyecto de GitHub https://github.com/microsoft-graph/microsoft-graph-toolkit

RUBÉN RAMOS MATEO
Technical Architect en Ricoh
ruben_rm@outlook.com
@rubenr79
https://rubenrm.com/

41

SQL Azure Serverless

El ecosistema serverless de Azure tiene un nuevo servicio: SQL Azure Serverless. Esta opción está en preview y nos permite ahorrar costes ya qué cuando no está en uso no tenemos costes de cálculo.

Características

Dos de las características principales que tiene este tipo de Base de Datos son:

- Escalado automático. A medida que lo necesite escalará automáticamente hasta el número máximo de vCores especificados y se facturara por segundos.
- Si no hay actividad, la base de datos se detiene automáticamente y queda inactiva, por lo que ya no se factura el procesado. En este caso solo se factura por el almacenamiento. En el momento que se necesiten datos la base de datos se reiniciará automáticamente.

Configuración

Cuando creamos una Base de datos SQL Azure Serverless podemos configurarla de la siguiente manera:

- Configuración de los vCores mínimos y los máximos. Esta configuración nos permite definir el intervalo de proceso disponible.
- Antes de que la Base de datos se detenga, podemos configurar el tiempo que queremos. que la Base de datos esté inactiva antes de que se detenga.
- Por último, también podemos desactivar la pausa automática.

En el siguiente gráfico podemos ver como funciona el procesamiento de este tipo de Base de Datos con un mínimo de 1 vCore y un máximo de 4 vCores.

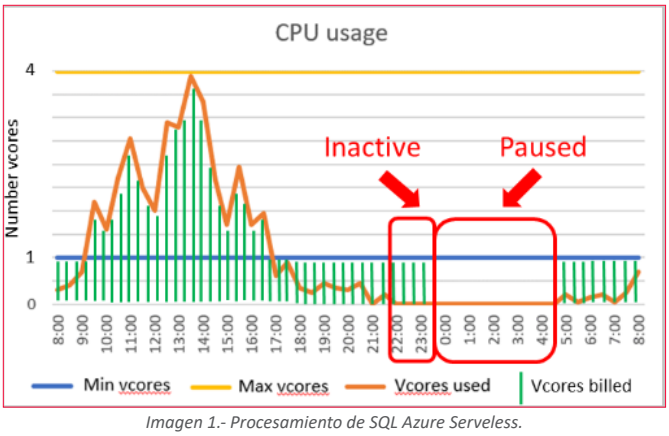


Imagen 1.- Procesamiento de SQL Azure Serveless.

Las líneas verdes muestran cuándo y a qué velocidad se facturan los ciclos de cálculo, dependiendo de la demanda real de cálculo (línea naranja) en la base de datos.

"El ecosistema serverless de Azure tiene un nuevo servicio: SQL Azure Serverless. Esta opción está en preview y nos permite ahorrar costes"

Costes

- El coste es la suma del coste de proceso y el coste de almacenamiento.
- Cuando el uso de proceso es entre los límites mínimos y máximos configurados, el coste de proceso se basa en los vCores seleccionados y la memoria utilizada.
- Cuando el uso de proceso está por debajo de los límites mínimos configurados, el coste de proceso se basa en los vCores mínimos y la cantidad mínima de memoria configurada.
- Cuando se pausa la base de datos, el coste de proceso es cero y solo se incurre en costes de almacenamiento.
- El coste de almacenamiento se determina de la misma manera que el nivel de proceso aprovisionado.

Ahora vamos a crear una SQL Azure Serverless. Entramos en el portal y creamos una SQL Azure como siempre.

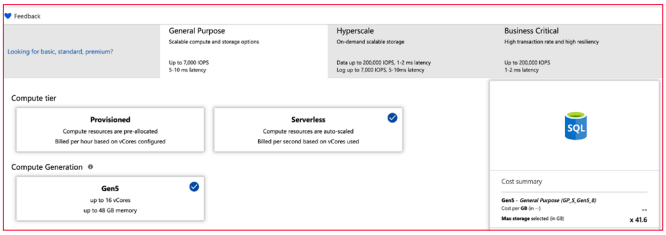


Imagen 2.-Creación de una BD SQL Azure Serveless.

Como la versión está en preview solo tenemos disponible la versión serverless en General Purpose y con Gen5. Después seleccionamos los vCores mínimos y máximos.

Imagen 3.- Selección del tipo de BD.

Por último, seleccionamos el intervalo de inactivación y el tamaño de almacenamiento

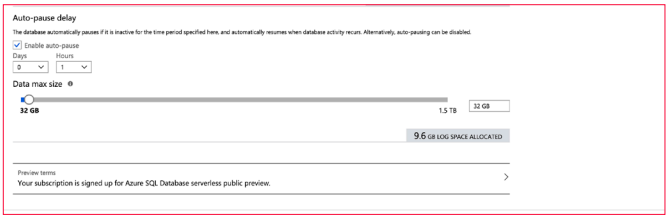


Imagen 4.- Intervalo de inactivación y tamaño de almacenamiento.

Una vez lo hemos configurado todo tenemos la siguiente configuración.

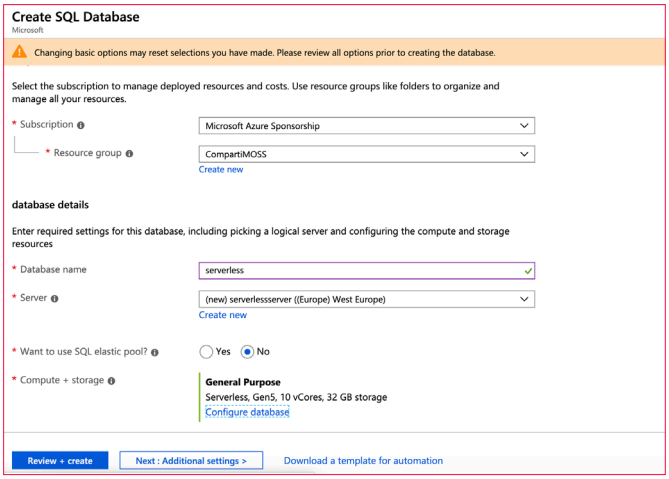


Imagen 5.- Configuraciones realziadas.

Le damos a Review + create y ya la tendremos creada.

"Se debe tener en cuenta que al ser un servicio serverless cuando este está inactivo y parado al reiniciarse tiene un tiempo de arranque, llamado "calentamiento"

Cuándo usarlas

Se debe tener en cuenta que al ser un servicio serverless cuando este está inactivo y parado al reiniciarse tiene un tiempo de arranque, llamado "calentamiento". Estos son los escenarios donde aplicaría este tipo de Base de datos:

- Base de datos donde su uso sea impredecible e intermitente, donde además haya períodos de inactividad.
- Base de datos donde haya frecuencia de cambios de escalado y queramos delegar el auto escalado al servicio.
- Cuando saber el uso y proceso de la base de datos no se sabe a priori.

ROBERT BERMEJO
Chapter Backend Lead in SCRM – Lidl Digital Hub
Microsoft Azure MVP
bermejoblasco@live.com
@robertbemejo
www.robertbermejo.com

43

Primeros pasos con Azure Resource Graph

Azure Resource Graph está diseñado para extender la administración de los recursos de Azure, ayudándonos a gobernar nuestro entorno un poquito más. Actualmente soporta querys contra recursos básicos de Azure, como nombres de recursos, ID, Type, Subscription… Pero como todo en Azure, seguro que evolucionará y muy rápido

Azure Resource Graph está basado en lenguaje Kusto, soportando un gran número de funciones y operadores.

Para preparar nuestro entorno para hacer querys contra Azure Resource Graph, como mínimo necesitaremos permisos de lectura sobre los recursos que queremos consultar, y utilizar Azure CLI, la extensión SDK con REST o bien PowerShell. En este post exploraremos este último método.

"Azure Resource Graph está diseñado para extender la administración de los recursos de Azure, ayudándonos gobernar nuestro entorno un poquito más"

Primero, para instalar los comandos de Azure Resource Graph, deberemos de abrir una consola de PowerShell y ejecutar el siguiente comando:

```
Install-Module -Name Az.ResourceGraph
```

A partir de aquí, sobre la misma consola de PowerShell, podemos ejecutar la siguiente query con lo que nos devolverá sobre qué número de objetos tenemos permisos sobre nuestro tenant:

```
Search-AzGraph -Query "summarize count()"
count_
------
   217
```

Ahora lo que haremos es consultar las máquinas virtuales que tenemos bajo la suscripción, y utilizaremos un operador para consultar tanto las máquinas en formato ARM como las que están en formato clásico (si algunos aún tenemos de esas)

```
Search-AzGraph -Query "where type == 'Microsoft.Compute/virtualMachines' or type == 'Microsoft.ClassicCompute/virtualMachines' | summarize count()"

count_
------
18
```

Incluso podemos ver el tipo de máquinas que tenemos:

```
Search-AzGraph -Query "where type =~ 'Microsoft.Compute/virtualMachines' | summarize count() by tostring(properties.storageProfile.osDisk.osType)"

properties_storageProfile_osDisk_osType count_
----------------------------------------- ------
Windows                                   14
Linux                                     4
```

Otros ejemplos que podemos utilizar (entre muchos) son:

```
# Listar Recursos por tipo de la suscripción
Search-AzGraph -Query "summarize count() by type, subscriptionId | order by type, subscriptionId asc"

# Listar Máquinas virtuales que cumplen con un patrón en concreto
Search-AzGraph -Query "where type =~ 'microsoft.compute/virtualmachines' and name matches regex @'^Contoso(.*)[0-9]+$' | project name | order by name asc"

 # Listar todas las IP públicas
Search-AzGraph -Query "where type contains 'publicIPAddresses' and isnotempty(properties.ipAddress) | project properties.ipAddress"

# Listar WebApps:
Search-AzGraph -Query "where type=='microsoft.web/sites' | project name, subscriptionId, type | order by type, subscriptionId"

# Listar Storage accounts:
Search-AzureRmGraph -Query "where type=='microsoft.storage/storageaccounts' | project name, resourceGroup,subscriptionId"
```

ALBERTO ANDRÉS RODRÍGUEZ
Cloud Architect en Seidor
@albandrod

Alberto Diaz

Alberto Díaz es SharePoint Team Lead en ENCAMINA, liderando el desarrollo de software con tecnología Microsoft.

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

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

Fabián Imaz

Fabián Imaz, MVP de SharePoint Server trabaja en el mundo del desarrollo de software desde hace más de 10 años, teniendo la suerte de trabajar en distintas arquitecturas y tecnologías Microsoft. Pertenece a la firma Siderys, http://www.siderys.com empresa de desarrollo de Software especializada en SharePoint 2007/2010/2013 y en desarrollo de soluciones inteligentes.

Desde los comienzos Fabián ha trabajado en distintitas comunidades donde organiza y promueve eventos locales para la difusión de tecnología dentro de los miembros de las mismas. Es director de la carrera SharePoint 2010 y SharePoint 2013 en Microsoft Virtual Academy, http://www.mslatam.com/latam/technet/mva2/ Home.aspx y cuenta con un sitio en CodePlex con varios desarrollos http://siderys.codeplex.com.

Sitio Web: http://www.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://geeks.ms/santypr** 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.es
Email: santiagoporras@outlook.com
Blogs: http://geeks.ms/santypr
Twitter: @saintwukong

Coordinadores de sección

GASTÓN CRUZ
Coordinador de PowerBi
gastoncruz@gmail.com

¿Desea colaborar con CompartiMOSS?

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

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

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

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

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

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


Comparti
MOSS