Usando WebAPI para generar Azure SAS para tus apps

Por Walter Novoa, **Microsoft Developer Evangelist**

* 1. 

Twitter: @warnov

Sitio personal: [http://warnov.com](http://warnov.com/)

He venido escribiendo una serie de posts acerca de los diversos mecanismos que existen para poder subir archivos desde nuestras apps de Windows 8 o de Windows Phone a Windows Azure.

En [este artículo](http://blogs.msdn.com/b/warnov/archive/2012/11/10/azure-como-backend-de-storage-para-windows-apps-y-el-poder-de-los-mobile-services.aspx) explico por qué es importante un mecanismo que nos permita autorizar solo por un tiempo determinado manejar un archive determinado en el Storage de Azure, de manera que no expongamos nuestra información de cuenta de storage a que un hacker la pueda descubrir hacienda ingeniería inversa de nuestras apps. Este mecanismo es lo que se conoce como Shared Access Signature de Windows Azure y no es más que una URL que tiene validez temporal, con la cual podemos consultar un archive, eliminarlo, modificarlo o crearlo.

En [este otro artículo](http://blogs.msdn.com/b/warnov/archive/2012/11/10/adicionando-archivos-al-storage-de-azure-apps-con-mobile-services.aspx), muestro cómo podemos usar los Mobile Services de Windows Azure para poder obtener dichas SAS. Es un mecanismo que nos permite ahorrarnos tener web servers y que escala muy bien, porque maneja muy poca transferencia. Además nos economiza la administración. Todo esto, versus tener un servicio web que reciba las fotografías y las envíe al storage de Windows Azure, ya que toda la responsabilidad de ese servicio recae sobre nosotros.

* 1. 

\*Azure + Web + Apps

Estas ventajas son muy válidas. Pero sucede que para poder ejecutar los Mobile Services, se requiere por el momento, tener una base de datos de SQL Azure, y esto puede generar costos adicionales. Además, Mobile Services se encuentra en etapa de preview, por lo que salir a producción usándolos, puede ser riesgoso.

Por estos motivos, he creado este post, en el que explico una manera alternativa para que nuestras apps puedan acceder a las codiciadas SAS, para poder subir archivos a Azure.

Dado que las SAS son fácilmente generables a través del API de Storage de Azure: Microsoft.WindowsAzure.StorageClient,  es muy fácil pensar en un servicio Web que use esta API para retornar la SAS, dado un nombre de blob determinado. Y es esto precísamente lo que vamos a hacer. Pero será un servicio web muy liviano y además que permita un fácil acceso desde diversas plataformas. Todo esto se logra gracias a WebAPI. Un framework basado en ASP.NET para poder exponer construir de manera sencilla Servicios HTTP disponibles para una gran variedad de clientes que manejan REST.

Como el framework es muy liviano y el proceso a ejecutar también lo es (solo se recibe una palabra y se retorna una URL), entonces es totalmente plausible que éste servicio se despliegue en un par instancias pequeñas de Windows Azure y soporte muy buena carga. De hecho, cuando los Windows Azure WebSites ya salgan de preview y comiencen a estar en producción, tendremos una alternativa aún más económica ya que está basada en servidores compartidos que luego pueden escalar y convertirse en servidores dedicados si las exigencias de nuestras apps así lo demandan. De hecho, algo que hoy en día existe que es la posibilidad de tener hasta 10 WebSites gratuitos, se mantendrá cuando el servicio pase a producción. En ambos casos (tanto con Cloud Services como con WebSites), nos ahorramos el despliegue de una Base de Datos y tenemos un poco más de control que con los Mobile Services.

Aprovecho en este punto entonces para hacer notar que he mostrado varias opciones y ninguna de ellas es la absolutamente recomendable para todos los casos. Las condiciones de cada problema son particulares y por ende lo son sus soluciones. Mi misión es mostrarles las posibilidades. La de ustedes, elegir inteligentemente.

Entonces continuando con nuestra solución basada en WebAPI tenemos nuestro controlador:

SERVER:

* 1. C#
  2. public class SASController : ApiController
  3. {
  4. //Esto es solo un ejemplo y pongo aquí directamente
  5. //los datos de la cuenta de acceso.
  6. //Estos deberían estar en el config encriptados.
  7. static CloudStorageAccount \_account;
  8. static bool \_accountSet=false;
  9. //En bien se construye la clase, se trata de ajustar
  10. //la cuenta de almacenamiento de Azure
  11. static SASController()
  12. {
  13. string accountName = "tunombredecuenta";
  14. string pak = "fP4fXF9Vg...";
  15. \_accountSet = CloudStorageAccount.TryParse(
  16. String.Format("DefaultEndpointsProtocol=https;"+
  17. "AccountName={0};AccountKey={1}",
  18. accountName,
  19. pak), out \_account);
  20. }
  21. // GET api/sas/nombredeblob
  22. public string GetSAS(string id)
  23. {
  24. //Si la cuenta ha sido ajustada correctamente
  25. if (\_accountSet)
  26. {
  27. //Creamos un container por defecto.
  28. //Se puede trabajar tambien como parámetro
  29. var container = \_account
  30. .CreateCloudBlobClient()
  31. .GetContainerReference("testcontainer");
  32. container.CreateIfNotExist();
  33. //Obtenemos una referencia al blob que se subirá
  34. var blob = container.GetBlobReference(id);
  35. //Creamos la sas que da permiso únicamente
  36. //para ese blob, y solo por dos minutos
  37. //qué seguro no?
  38. var sas = blob.GetSharedAccessSignature
  39. (new SharedAccessPolicy()
  40. {
  41. Permissions = SharedAccessPermissions.Read
  42. | SharedAccessPermissions.Write,
  43. SharedAccessExpiryTime = DateTime.UtcNow +
  44. TimeSpan.FromMinutes(2)
  45. });
  46. //Armamos la url en la cual podremos
  47. //poner el blob desde el cliente
  48. return blob.Uri.AbsoluteUri + sas;
  49. }
  50. else return "The API has not been configurated yet, "+
  51. "pray to the heavens for them to configure it";
  52. }
  53. }

Vemos como solo bastó importar la librería del API de Storage, inicializar la cuenta de almacenamiento a través de un constructor estático y luego en la acción Get del controlador SAS usar la cuenta inicializada para producir la SAS basados en el nombre del blob que nos llega como parámetro “id”. Le dejé ese nombre para hacer prevalecer el paradigma de “Convention over Configuration”que al igual que en MVC, está muy presente en WebAPI.

Vemos cómo es posible configurar el tipo y tiempo de acceso que otorgará la SAS. Todo eso queda incluido en la URL que se retorna al cliente. Algo del tipo: <https://cuenta.blob.core.windows.net/img/archivo.jpg?sv=201324kjn,mnetc>…

Con esta URL luego el cliente lo único que tiene que hacer es instanciar un HTTPClient y a través de este, poner el contenido del archivo con los headers específicos en dicha URL. Veámos el código de una app de Win8 para este fin:

CLIENT

* 1. C#
  2. //Escogemos el archivo
  3. var filePicker = new FileOpenPicker();
  4. filePicker.FileTypeFilter.Add(".jpg");
  5. var file = await filePicker.PickSingleFileAsync();
  6. if(file!=null)
  7. {
  8. //Con un cliente http nos comunicamos
  9. //al web api para bajar la SAS
  10. //luego con ese mismo cliente subimos el archivo
  11. using (var client = new HttpClient())
  12. {
  13. //Bajando la SAS
  14. var sas = await client.GetStringAsync
  15. ("http://localhost:47805/api/sas/" + file.Name);
  16. sas = sas.Substring(1, sas.Length - 2);
  17. //Cargamos la imagen con el HttpClient al blob
  18. //service usando la SAS obtenida desde WebAPI
  19. //Obtenemos el stream de un storage file definido anteriormente
  20. using (var fileStream = await file.OpenStreamForReadAsync())
  21. {
  22. var content = new StreamContent(fileStream);
  23. content.Headers.Add("Content-Type", file.ContentType);
  24. content.Headers.Add("x-ms-blob-type", "BlockBlob");
  25. //Con el PutAsync, enviamos el archivo a Azure
  26. //a través de la URL autorizadora que está en SAS
  27. using (var uploadResponse =
  28. await client.PutAsync(new Uri(sas), content))
  29. {
  30. //Agregar cualquier post proceso adicional
  31. }
  32. }
  33. }
  34. }

Con el HTTPClient hemos descargado la SAS luego de tener el archivo escogido con su nombre identificado y luego aprovechamos ese mismo HTTPClient para poner el contenido del archivo al cual le hemos añadido unos headers para que pueda ser manejado correctamente por los Servicios REST del storage de Azure. Es importante notar aquí que la transferencia del archivo se realiza directamente desde el cliente a los Servicios de storage de Azure. El servicio web solo da la SAS. Esto es económico, veloz y escalable.

Quise usar el anterior ejemplo para mostrar un escenario de uso de WebAPI. Pero a decir verdad, hay otra manera aún más liviana de lograrlo y es a través del uso de un manejador genérico de ASP.NET o [ASHX](http://blogs.msdn.com/b/warnov/archive/2009/08/15/introducci-n-a-generic-handlers-en-asp-net.aspx). Un ASHX solo requiere de la Plataforma más elemental de ASP.NET (no requiere un motor de ruteo por ejemplo) para funcionar y lo hace también a través de la versatilidad de HTTP. Además el código es idéntico, except que el nombre del blob ya no va embebido en la ruta del request sino como un parámetro del querystring y el método como tal ya no va como una acción dentro del controller, sino como el ProcessRequest del handler.

**Summary:**

En conclusión, hemos visto cómo producir SAS desde Servicios web basados en HTTP, como lo proveen la WebAPI y los Generic Handlers. Luego vimos cómo usar esas SAS para poder subir archivos desde nuestras apps a Azure.