Azure Blob Storage - Uploading images using REST API

This article explains how to work with *Windows Azure Blob Storage* using its REST API, allowing access to images or other files that can be accessed from anywhere in the world via HTTP or HTTPS.

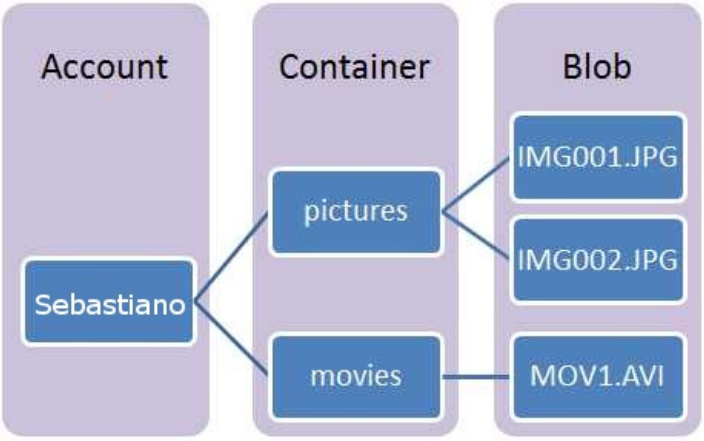
Introduction

Windows Azure Blob Storage is a web service for storing large amounts of unstructured data. You can use blob storage to expose data publicly to the world, or privately for internal application storage.

Common uses of Blob storage include:

- Serving images or documents directly to a browser
- Storing files for distributed access
- Streaming video and audio
- Performing secure backup and disaster recovery
- Storing data for analysis by an on-premises or Windows Azure-hosted service

Blobs are stored in containers, which are themselves associated with a "storage account". A single blob can be hundreds of gigabytes in size, and a single storage account can contain up to 100TB of blob:



- **Storage Account**: All access to Windows Azure Storage is done through a storage account. This is the highest level of the namespace for accessing blobs. An account can contain an unlimited number of containers, as long as their total size is under 100TB.
- **Container**: A container provides a grouping of a set of blobs. All blobs must be in a container. A container can store an unlimited number of blobs.
- **Blob**: A file of any type and size. There are two types of blobs that can be stored in Windows Azure Storage: block and page blobs. Most files are block blobs - and this is the type of blob used in the article. A single block blob can be up to 200GB in size. Page blobs can be up to 1TB in size, and are more efficient when ranges of bytes in a file are modified frequently. For more information about blobs, see Understanding Block Blobs and Page Blobs.

Blobs are addressable using the following URL format: *http://<storage account>.blob.core.windows.net/<container>/<blob>* . The following URL could be used to address one of the blobs in the diagram above: http://sebastiano.blob.core.windows.net/movies/MOV1.AVI or http://sebastiano.blob.core.windows.net/nokia/logo.png

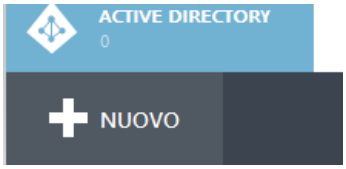
Why REST API?

The Azure .NET SDK provides classes to access Blob Storage and there is good documentation both on MSDN and the wider Internet. The REST API discussed in this article is much less well documented. However if you're working on a number of platforms (e.g. Nokia Asha) this approach gives you a common "cross platform" sign-in experience and is less dependent on the external SDK.

Create a Storage Account

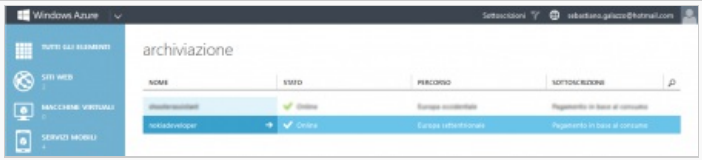
- Log into the Windows Azure Management Portal.
- At the bottom of the navigation pane, click NEW.

(C) Copyright Nokia 2014. All rights reserved.



Click **Data Services** -> **Storage**, and then click **Quick Create**.

Create new mobile service button

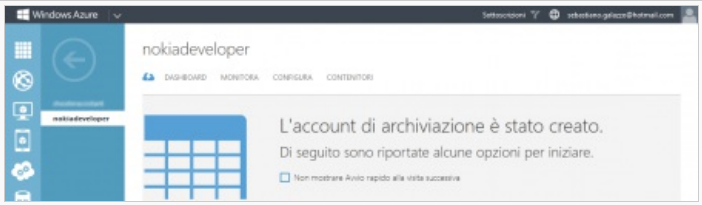


Create new mobile service button

Create a new Container

A container can be created both in code and in the Management Portal. We will use the portal as it makes sense to first create the container to host all user's avatar.

Access the main page and click on **Container** -> **Create Container**



Create new mobile service button

Create new mobile service button

Choose the name

Create new mobile service button

(C) Copyright Nokia 2014. All rights reserved.

Create new mobile service button

The container is created at the address : **http://nokiadeveloper.blob.core.windows.net/nokia**

Understanding the authentication process

Note: You can skip this section if you are not interested as the code below will manage all the following stuff.

Every request made against a storage service must be authenticated, unless the request is for a blob or container resource that has been made available for public or signed access. An authenticated request requires two headers: the Date (or x-ms-date header) and the Authorization header. The following sections describe how to construct these headers.

Beginning with version 2009-09-19, the Blob, Queue, and Table services support three following Shared Key authentication schemes:

- Shared Key for Blob and Queue Services
- Shared Key for Table Service
- Shared Key Lite

We will focus on the first one

Date Header

All authenticated requests must include the Coordinated Universal Time (UTC) timestamp for the request. You can specify the timestamp either in the x-ms-date header, or in the standard HTTP/HTTPS Date header. If both headers are specified on the request, the value of x-ms-date is used as the request's time of creation.

Authorization Header

An authenticated request must include the Authorization header. If this header is not included, the request is anonymous and may only succeed against a blob container that is marked for public access, or marked for signed access via a Shared Access Signature.

To authenticate a request, you must sign the request with the key for the account that is making the request and pass that signature as part of the request. The format for the Authorization header is as follows:

```
Authorization="[SharedKey|SharedKeyLite] <AccountName>:<Signature>"
```

Constructing the Signature String

How you construct the signature string depends on which service and version you are authenticating against, and which authentication scheme you are using. When constructing the signature string, keep in mind the following:

- The VERB portion of the string is the HTTP verb, such as GET, PUT or DELETE, and must be upper case. It represent the operation you are going to perform, to GET a Blob, to upload a new Blob (PUT) or DELETE an existing Blob.
- The values of all standard HTTP headers must be included in the string in the order shown in the signature format, without the header names. These headers may be empty if they are not being specified as part of the request; in that case, only the new line character is required.
- All new line characters (\n) shown are required within the signature string.

```
GET\n /*HTTP Verb*/
\n    /*Content-Encoding*/
\n    /*Content-Language*/
\n    /*Content-Length*/
\n    /*Content-MD5*/
\n    /*Content-Type*/
```


(C) Copyright Nokia 2014. All rights reserved.

```
\n    /*Date*/
\n    /*If-Modified-Since */
\n    /*If-Match*/
\n    /*If-None-Match*/
\n    /*If-Unmodified-Since*/
\n    /*Range*/
x-ms-date:Sun, 11 Oct 2009 21:49:13 GMT\nx-ms-version:2009-09-19\n
/*CanonicalizedHeaders*/
/myaccount/myaccount/mycontainered\ncomp:metadata\nrestype:container\ntimeout:20
/*CanonicalizedResource*/
```

Next, encode this string by using the HMAC-SHA256 algorithm over the UTF-8-encoded signature string, construct the Authorization header, and add the header to the request. The following example shows the Authorization header for the same operation:

```
Authorization: SharedKey myaccount:ctzMq410TV3wS7upTBcunJTDLEJwMAZuFPfr0mrrA08=
```

Canonicalized Headers String

To construct the `CanonicalizedHeaders` portion of the signature string, follow these steps:

- Retrieve all headers for the resource that begin with x-ms-, including the x-ms-date header.
- Convert each HTTP header name to lower-case.
- Sort the headers lexicographically by header name, in ascending order. Note that each header may appear only once in the string.
- Unfold the string by replacing any breaking white space with a single space.
- Trim any white space around the colon in the header.

Finally, append a new line character to each canonicalized header in the resulting list. Construct the `CanonicalizedHeaders` string by concatenating all headers in this list into a single string.

Code

The following code will show a **namespace** I created to easy manage the **Blob Storage** and how to use it.

Contructor

```
using System;
using System.Net;
using System.Text;
using System.IO;
using System.Globalization;

namespace AzureManager
{
    class Storage
    {
        public string Account { get; set; }
        public string BlobType { get; set; }
        public string BlobEndPoint { get; set; }
        public string Key { get; set; }
        public string SharedKeyAuthorizationScheme { get; set; }

        public Storage()
        {
            BlobType = "BlockBlob";
            SharedKeyAuthorizationScheme = "SharedKey";
```


(C) Copyright Nokia 2014. All rights reserved.

```
        }
        ...
```

Property	Description
Account	Gets or sets the account name to created to use with the Blob Storage.
BlobType	Gets or sets the BlobType, BlockBlob or PageBlob. By default is set to **BlockBlob**
BlobEndPoint	Gets or sets the endpoint to use
Key	Gets or sets the key to access the Storage. You can get it from the Management Panel
SharedKeyAuthorizationScheme	Get or set the authrization schema, here **SharedKey**

Build the Authorization Header

```
 private string CreateAuthorizationHeader(string canonicalizedstring)
 {
            string signature = string.Empty;
            using (System.Security.Cryptography.HMACSHA256 hmacSha256 = new
System.Security.Cryptography.HMACSHA256(Convert.FromBase64String(Key)))
            {
                Byte[] dataToHmac =
System.Text.Encoding.UTF8.GetBytes(canonicalizedstring);
                signature = Convert.ToBase64String(hmacSha256.ComputeHash(dataToHmac));
            }
            string authorizationHeader = string.Format(CultureInfo.InvariantCulture,
"{0} {1}:{2}", SharedKeyAuthorizationScheme, Account, signature);
            return authorizationHeader;
 }
```

Uploading a BLob

```
        public async void PutBlob(String containerName, String blobName, byte[]
blobContent, bool error = false)
        {
            String requestMethod = "PUT";
            String urlPath = String.Format("{0}/{1}", containerName, blobName);
            String storageServiceVersion = "2009-09-19";
            String dateInRfc1123Format = DateTime.UtcNow.ToString("R",
CultureInfo.InvariantCulture);

            Int32 blobLength = blobContent.Length;

            String canonicalizedHeaders = String.Format(
                "x-ms-blob-type:{0}\nx-ms-date:{1}\nx-ms-version:{2}",
                BlobType,
                dateInRfc1123Format,
                storageServiceVersion);
                String canonicalizedResource = String.Format("/{0}/{1}", Account,
urlPath);
                String stringToSign = String.Format(
                "{0}\n\n\n{1}\n\n\n\n\n\n\n\n\n{2}\n{3}",
                requestMethod,
                blobLength,
                canonicalizedHeaders,
                canonicalizedResource);
```


(C) Copyright Nokia 2014. All rights reserved.

```
            String authorizationHeader = CreateAuthorizationHeader(stringToSign);

            Uri uri = new Uri(BlobEndPoint + urlPath);
            HttpWebRequest request = (HttpWebRequest)WebRequest.Create(uri);
            request.Method = requestMethod;
            request.Headers["x-ms-blob-type"] = BlobType;
            request.Headers["x-ms-date"] = dateInRfc1123Format;
            request.Headers["x-ms-version"] = storageServiceVersion;
            request.Headers["Authorization"] = authorizationHeader;
            request.ContentLength = blobLength;

            try
            {
                using (Stream requestStream = await request.GetRequestStreamAsync())
                {
                    requestStream.Write(blobContent, 0, blobLength);
                }

                using (HttpWebResponse response = (HttpWebResponse)await
request.GetResponseAsync())
                {
                    String ETag = response.Headers["ETag"];
                    System.Diagnostics.Debug.WriteLine(ETag);
                }
                error = false;
            }
            catch (WebException ex)
            {
                System.Diagnostics.Debug.WriteLine("An error occured. Status code:" +
((HttpWebResponse)ex.Response).StatusCode);
                System.Diagnostics.Debug.WriteLine("Error information:");
                error = true;
                using (Stream stream = ex.Response.GetResponseStream())
                {
                    using (StreamReader sr = new StreamReader(stream))
                    {
                        var s = sr.ReadToEnd();
                        System.Diagnostics.Debug.WriteLine(s);
                    }
                }
            }

        }
```

Delete a Blob

```
 public async void DeleteBlob(String containerName, String blobName, bool error=false)
 {
            String requestMethod = "DELETE";

            String urlPath = String.Format("{0}/{1}", containerName, blobName);

            String storageServiceVersion = "2009-09-19";
```


(C) Copyright Nokia 2014. All rights reserved.

```
            String dateInRfc1123Format = DateTime.UtcNow.ToString("R",
CultureInfo.InvariantCulture);

            Int32 blobLength = 0; // blobContent.Length;

            String canonicalizedHeaders = String.Format(
                "x-ms-blob-type:{0}\nx-ms-date:{1}\nx-ms-version:{2}",
                BlobType,
                dateInRfc1123Format,
                storageServiceVersion);
            String canonicalizedResource = String.Format("/{0}/{1}", Account, urlPath);
            String stringToSign = String.Format(
                "{0}\n\n\n{1}\n\n\n\n\n\n\n\n\n{2}\n{3}",
                requestMethod,
                blobLength,
                canonicalizedHeaders,
                canonicalizedResource);

            String authorizationHeader = CreateAuthorizationHeader(stringToSign);

            Uri uri = new Uri(BlobEndPoint + urlPath);
            HttpWebRequest request = (HttpWebRequest)WebRequest.Create(uri);
            request.Method = requestMethod;
            request.Headers["x-ms-blob-type"] = BlobType;
            request.Headers["x-ms-date"] = dateInRfc1123Format;
            request.Headers["x-ms-version"] = storageServiceVersion;
            request.Headers["Authorization"] = authorizationHeader;
            request.ContentLength = blobLength;

            try
            {
                using (HttpWebResponse response = (HttpWebResponse)await
request.GetResponseAsync())
                {
                    String ETag = response.Headers["ETag"];
                    System.Diagnostics.Debug.WriteLine(ETag);
                    error = false;
                }
            }
            catch (WebException ex)
            {
                System.Diagnostics.Debug.WriteLine("An error occured. Status code:" +
((HttpWebResponse)ex.Response).StatusCode);
                System.Diagnostics.Debug.WriteLine("Error information:");

                error = true;

                using (Stream stream = ex.Response.GetResponseStream())
                {
                    using (StreamReader sr = new StreamReader(stream))
                    {
                        var s = sr.ReadToEnd();
                        System.Diagnostics.Debug.WriteLine(s);
                    }
                }
            }
 }
```


(C) Copyright Nokia 2014. All rights reserved.

Downloading a Blob

```
        public async void GetBlob(String containerName, String blobName, byte[] result)
        {
            String requestMethod = "GET";

            String urlPath = String.Format("{0}/{1}", containerName, blobName);

            String storageServiceVersion = "2009-09-19";

            String dateInRfc1123Format = DateTime.UtcNow.ToString("R",
CultureInfo.InvariantCulture);

            Int32 blobLength = 0; // blobContent.Length;

            String canonicalizedHeaders = String.Format(
                "x-ms-blob-type:{0}\nx-ms-date:{1}\nx-ms-version:{2}",
                BlobType,
                dateInRfc1123Format,
                storageServiceVersion);
            String canonicalizedResource = String.Format("/{0}/{1}", Account, urlPath);
            String stringToSign = String.Format(
                "{0}\n\n\n{1}\n\n\n\n\n\n\n\n\n{2}\n{3}",
                requestMethod,
                blobLength,
                canonicalizedHeaders,
                canonicalizedResource);

            String authorizationHeader = CreateAuthorizationHeader(stringToSign);

            Uri uri = new Uri(BlobEndPoint + urlPath);
            HttpWebRequest request = (HttpWebRequest)WebRequest.Create(uri);
            request.Method = requestMethod;
            request.Headers["x-ms-blob-type"] = BlobType;
            request.Headers["x-ms-date"] = dateInRfc1123Format;
            request.Headers["x-ms-version"] = storageServiceVersion;
            request.Headers["Authorization"] = authorizationHeader;
            request.ContentLength = blobLength;

            try
            {
                using (HttpWebResponse response = (HttpWebResponse)await
request.GetResponseAsync())
                {
                    String ETag = response.Headers["ETag"];
                    System.Diagnostics.Debug.WriteLine(ETag);
                    using (Stream stream = response.GetResponseStream())
                    {
                        using (StreamReader sr = new StreamReader(stream))
                        {
                            var s = sr.ReadToEnd();
                            result =  Convert.FromBase64String(s);
                        }
                    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
                }
            }
            catch (WebException ex)
            {
                System.Diagnostics.Debug.WriteLine("An error occured. Status code:" +
((HttpWebResponse)ex.Response).StatusCode);
                System.Diagnostics.Debug.WriteLine("Error information:");
                using (Stream stream = ex.Response.GetResponseStream())
                {
                    using (StreamReader sr = new StreamReader(stream))
                    {
                        var s = sr.ReadToEnd();
                        System.Diagnostics.Debug.WriteLine(s);
                    }
                }
            }
        }
```

Using the code in the real life

Imagine a scenario where you would manage all your user's avatars. You need to retrieve the image, modify or upload a new one and deleting it.

Retrieve images from the Cloud

```
private async void LoadAvatar()
{
            Uri uriResult;
            bool isUriValid = Uri.TryCreate(App.settings.user.Avatar, UriKind.Absolute,
out uriResult) && uriResult.Scheme == Uri.UriSchemeHttp;

            if (isUriValid)
            {
                var webRequest = HttpWebRequest.Create(uriResult);
                // Just request the headers, not the whole content
                webRequest.Method = "HEAD";
                try
                {
                    // Exception is thrown if 404
                    var response = await webRequest.GetResponseAsync() as
HttpWebResponse;
                    // Check if status return is 200 OK, then active.
                    if (response.StatusCode == HttpStatusCode.OK)
                    {
                        Console.WriteLine(((int)response.StatusCode).ToString());
                        Avatar.Source = new BitmapImage(uriResult);
                    }
                }
                catch (WebException e)
                {
                    Console.WriteLine(e.Message);
                }
            }
            else
            {
                Console.WriteLine("Avatar Url is not valid");
```


(C) Copyright Nokia 2014. All rights reserved.

```
            }
 }
```

As we don't know if the avatar is really available so first we send a request with only headers and check if the request returns a 200 status code. If successful then we really download the image and assign downloaded content to our image into the app.

Upload an image into Cloud

First we need to allow the user to select the image from his gallery or shoot a new one. In this example we manage the process of choosing the image from media library (the process for shooting an image is similar and well documented on this wiki and MSDN).

```
private void UploadNewAvatar_Click(object sender, RoutedEventArgs e)
{
            PhotoChooserTask photoChooserTask = new PhotoChooserTask();
            photoChooserTask.Completed += new
EventHandler<PhotoResult>(photoChooserTask_Completed);
            photoChooserTask.Show();
}
```

```
void photoChooserTask_Completed(object sender, PhotoResult e)
{
            if (e.TaskResult == TaskResult.OK)
            {

                BitmapImage bitmap = new BitmapImage();
                bitmap.SetSource(e.ChosenPhoto);

                WriteableBitmap wb = new WriteableBitmap(bitmap);

                using (MemoryStream stream = new MemoryStream())
                {
                    wb.SaveJpeg(stream, wb.PixelWidth, wb.PixelHeight, 0, 85);
                    byte[] imageBytes = stream.ToArray();

                    AzureManager.Storage storage = new AzureManager.Storage();
                    storage.Account = "YOUR BLOB STORAGE ACCOUNT"; // (i.e.
nokiadeveloper)
                    storage.BlobEndPoint = "YOUR ENDPOINT"; // (i.e.
https://nokiadeveloper.blob.core.windows.net/ )
                    storage.Key = "YOUR STORAGE KEY";

                    string fileName = "avatar-" + App.settings.user.Id + ".jpg";

                    bool error = false;
                    Dispatcher.BeginInvoke(() => ProfileBusyIndicator.Content =
"Deleting old image from Cloud...");
                    storage.DeleteBlob("shooter-assistant", App.settings.user.Avatar,
error);
                    if (!error)
                    {
                        Dispatcher.BeginInvoke(() => ProfileBusyIndicator.Content =
"Uploading image into Cloud...");
                        storage.PutBlob("shooter-assistant", fileName, imageBytes,
error);
```


(C) Copyright Nokia 2014. All rights reserved.

```
                        if (!error)
                        {
                            Dispatcher.BeginInvoke(() => RecordDataBusyIndicator.Content
= "Updating informations...");
                            UpdateAvatar(fileName, error);
                            if (!error)
                            {
                                Avatar.Source = bitmap;
                                MessageBox.Show("Avatar updated succeffuly.\n");
                            }
                            else
                            {
                                MessageBox.Show("Error updating the new Avatar.\n");
                            }
                        }
                        else
                        {
                            MessageBox.Show("Error uploading the new Avatar.\nCode:
[PUT]\n");
                        }
                    }
                    else
                    {
                        MessageBox.Show("Error uploading the new Avatar.\nCode:
[DEL]\n");
                    }
                }
                Dispatcher.BeginInvoke(() => ProfileBusyIndicator.IsRunning = false);
            }
 }
```

Once the image is selected, prepare the Blob with the needed data. It may be useful to delete old images before uploading a new one in order to save space - however this is not mandatory and depends on your workflow and needs.

Updating the table into Mobile service

If your application allows users to upload images or content using the original name, as a last step you should update the the *Users* table as is still referring to the old image name. If in your process you created a structured name (i.e <name>-<surname>-<id>.png) you can skip this step.

```
 private async void UpdateAvatar(string avatar, bool error=false)
{
            App.settings.user.Avatar = avatar;

            await
App.Azure.GetTable<Users>().UpdateAsync(App.settings.user).ContinueWith(t =>
            {
                if (t.IsFaulted)
                {
                    System.Diagnostics.Debug.WriteLine("\nAggiornamento su Azure
[Tabella Users] fallito!!!\n");
                    error = true;
                }
                else
                {
                    System.Diagnostics.Debug.WriteLine("\nAggiornamento su Azure
[Tabella Users] avvenuto con successo\n");
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    error = false;
                }
            });
    }
```

Summary

Windows Azure Blob Storage is a powerful tool to manage media content for our mobile applications. I focused the example on users avatar management, but it's also possible to store videos and perform streaming.

You can download the full code from this resource: File:Azure Storage using REST API.zip

(C) Copyright Nokia 2014. All rights reserved.