Azure Storage

## **What is Azure Storage?**

Azure Storage play an important role in the Microsoft Azure  
Platform-as-a-Service (PaaS) strategy for storage. Azure Storage enables storage and retrieval of large amounts of unstructured data. You can store content files such as documents and media in the Blob service, use the Table service for NoSQL data, use the Queue service for reliable messages, and use the File service for Server Message Block (SMB) file share  
scenarios.

Azure Storage is massively scalable cloud storage service, so you can store and process hundreds of terabytes of data to support the big data scenarios required by scientific, financial analysis and media applications.

Azure Storage currently stores tens of trillions of unique customer objects, and handles millions of requests per second on average.

Azure Storage is elastic, so you can design applications for a large global audience, and scale those applications as needed.

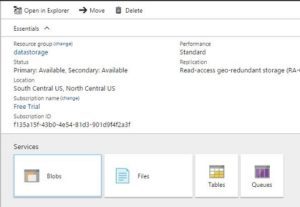
As the demands on your application grow, Azure Storage automatically allocates the appropriate resources to meet them.

Azure Storage is accessible from anywhere in the world, from any type of application, whether it’s running in the cloud, on the desktop, on an on-premises server, or on a mobile or tablet device.

Azure Storage also provides the storage foundation for Azure Virtual Machines.

## **Azure Storage services**

Azure storage provides the following four services: Blob storage, Table storage, Queue storage, and File storage.

Storage Services

* Blob Storage  
  Stores unstructured object data. A blob can be any type of text or binary data, such as a document, media file, or application installer.
* File Storage  
  Azure File storage is a service that offers file shares in the cloud using the standard[Server Message Block (SMB) Protocol](https://msdn.microsoft.com/library/windows/desktop/aa365233.aspx). On-premises applications can access file data in a share via the File service REST API.
* Table Storage  
  Stores structured datasets. Table storage is a NoSQL key-attribute data store, which allows for rapid development and fast access to large quantities of data.
* Queue Storage  
  Queue storage delivers asynchronous messaging for communication between application components, whether they are running in the cloud, on the desktop, on an on-premises server, or on a mobile device.

## **How data is stored using Storage Service?**

Following image in short shows how data is stored using azure storage service.

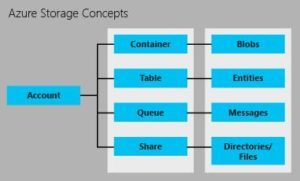
1. Create storage account first, then you get option to store data in Blobs, Table, Queues and Files. An Azure storage account provides a unique namespace to store and access your Azure Storage data objects. There are two types of storage accounts.

General-purpose Storage Accounts: A general-purpose storage account gives you access to Azure Storage services such as Tables, Queues, Files, Blobs and Azure virtual machine disks. There are two type of general purpose accounts.  
Standard storage: Allows you to store Tables, Queues, Files, Blobs and Azure virtual machine disks.  
Premium storage: supports only Azure virtual machine disks with input/output (I/O)-intensive workloads.

Blob Storage Accounts: A Blob storage account is a specialized storage account for storing your unstructured data as blobs (objects) in Azure Storage. There are two types Blob Storage that can be specified based on your data access pattern.

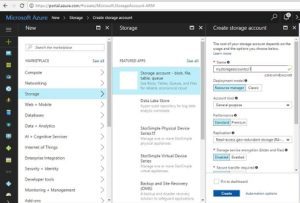
Hot access: Indicates that the objects in the storage account will be more frequently accessed. This allows you to store data at a lower access cost.  
Cool access: Indicates that the objects in the storage account will be less frequently accessed. This allows you to store data at a lower data storage cost.

1. In order to store data in blobs create containers first, within each container you can upload files (images, videos etc.).
2. To store data in NoSQL format, create table using APIs, then create entities within. Or you can use azure management studio to create data table from CSV, XML or relational database.
3. To store data in queues, create queues first from portal and then create new message within.
4. To store data in file storage, first need to create file share and directory structure in portal and then upload files within.

Azure Storage Concept

In order to use storage service you need to create storage account.

Navigate to new (+) -> Storage -> Storage Account -> Provide the details of storage account and click on create button.

Create new storage account

Once storage account is created you can see the same in the list of all resources, if you open it you will able to see different storage services available for you to consume as stated above.

Storage Services in Storage Account

# BLOB Storage

Cloud computing enables new scenarios for applications, which require scalable, durable, and highly available storage for their data, which is supported by Azure Storage, It makes possible for developers to build large-scale applications to support new scenarios.

## **What is BLOB Storage?**

BLOB is an acronym for Binary Large Object, which is a collection of binary data stored as a single entity.

Azure Blob storage is a service that stores unstructured data in the cloud as objects/blobs. Blob storage can store any type of text or binary data, such as a document, media file, or application installer. Blob storage is also referred to as object storage.

BLOB storage can be accessed from anywhere in the world via HTTP or HTTPS. You can use Blob storage to expose data publicly to the world, or to store application data privately.

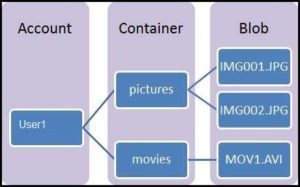
## **What is USE of Blob storage?**

Common uses of Blob storage is as follows.

* Storing Documents
* Social data such as photos, videos, music, and blogs
* Backups of files, computers, databases, and devices
* Images and text for web applications
* Configuration data for cloud applications
* Big data, such as logs and other large datasets

## **What are the components of BLOB Storage?**

Following are basic components of BLOB Storage Service

Components of BLOB Storage

Account: All access to Azure Storage is done through a storage account. For more details please refer article [Azure Storage – Part 1](http://www.sharecareinspire.com/azure-storage/)

Container: A container provides a grouping of a set of blobs.

Blob: A file of any type and size. Azure Storage offers two main types of blobs: block blobs, page blobs.

Block blobs are ideal for storing text or binary files, such as documents and media files. A single block blob can contain up to 50,000 blocks of up to 100 MB each, for a total size of slightly more than 4.75 TB (100 MB X 50,000).

Append blobs are similar to block blobs in that they are made up of blocks, but they are optimized for append operations, so they are useful for logging scenarios.

Page blobs can be up to 1 TB in size, and are more efficient for frequent read/write operations. Azure Virtual Machines use page blobs as OS and data disks.

## **How data is structured in BLOB storage?**

Every blob is organized into a container. Containers also provide a useful way to assign security policies to groups of objects. A storage account can contain any number of containers, and a container can contain any number of blobs, up to the 500 TB capacity limit of the storage account.

## **BLOB storage data access scenarios**

During creation of storage account you can specify what type of data (access tier) will be stored in BLOB storage, there are two types of access tiers that can be specified based on data access pattern:

Hot: indicates that the objects in the storage account will be more frequently accessed. This allows you to store data at a lower access cost.

Cool: indicates that the objects in the storage account will be less frequently accessed. This allows you to store data at a lower data storage cost.

If there is a change in the usage pattern of your data, you can also switch between these access tiers at any time.

## **How to Create BLOB storage using ARM Portal**

### **Step 1: Create Storage Account of type BLOB**

Follow the link on portal to create new storage account. New (+) -> Storage -> Storage Account.

Provide the name of storage account; you need to make two important selections in the properties below.

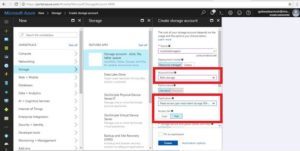
Account kind: This field has two options, Blob Storage and General Purpose. This article focused on Blob storage, hence select Blob storage value from dropdown. For more details about these two types please refer article [Azure Storage – Part 1](http://www.sharecareinspire.com/azure-storage/)

Replication: This is a solution for high availability and disaster recovery of your data, what is your primary data center fails because of some natural disaster or disks/racks on which your data was stored fails. There are three options for replication as follows.

Read-application-geo-redundant storage (RA-GRS): This is default selection. This is geo-redundant storage plus the ability to read the data in the secondary data center. You can change your application to read your data from the secondary data center, in case primary data center is not responding. Also, if you have an application where only a few users can write data, but lots of people read the data, you could point the application that writes to storage at the primary data center, and then do all the reads from the secondary.  
Geo-redundant-storage (GRS):  This replicates your data three times in your chosen data center, and then replicates it three times in a secondary data center that is far away.

Locally-redundant-storage (LRS): Three copies of your blobs are stored in a single facility in a single region. The replicas reside in separate fault domains and upgrade domains. This means that data is available even if the rack where your data is stored fails or is taken offline to be updated. Data is always updated in all three copies.

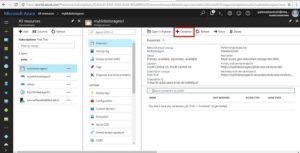
Access Tier: This field has two options, cool and hot. Default is hot, you make selection are per your requirement. For more details about these two types please refer article [Azure Storage – Part 1](http://www.sharecareinspire.com/azure-storage/)

Create Storage Account

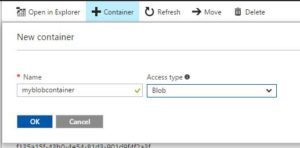
Click on Create button and storage account will be provisioned and will be shown in all resources list.

### **Step 2: Create Container**

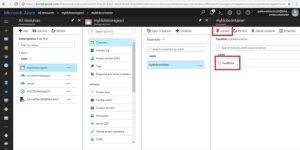
A container provides a grouping of a set of blobs. All blobs must be in a container. To create container, open properties blade for blob storage and select overview.

Blob Storage Containers

Add new container with access type as Blob, click on ok to create new container.

Add Blob Storage Container

All available containers will be shown in the list, select the container and you will be able to upload blob data in the container. Following screen snap shows option to upload data and one text file is already uploaded in the container.

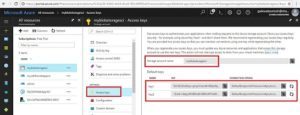
Upload Files in Container

## **Access Keys**

Azure provides APIs to work with storage account programmatically. The account name and key values are used to create a StorageCredential class instance, which is then used to create an instance of the CloudStorageAccount. The CloudStorageAccount object represents a storage account that you will be accessing. From this object you can obtain references to BLOBs  within the storage account.

StorageCredentials cred = new StorageCredentials(accountName, accessKey);

CloudStorageAccount account = new CloudStorageAccount(cred, useHttps: true);

Blob Storage Access Keys

# Read and Edit BLOB Storage Data from Azure Web App

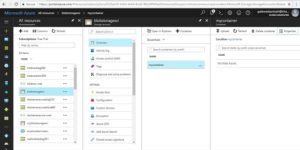
## **How to read data programmatically from Blob storage?**

Generally you would not be using Blob storage to upload data manually, there would be a system which will write data to and read data from blob storage.

This article focuses on reading, updating, deleting data from Blob storage using .NET code.

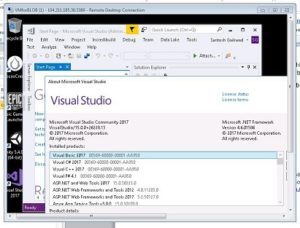
### **Step 1: Create Azure Blob Storage**

Login to Azure ARM portal and create Blob storage account and container. To know about how to create Blob storage account and container, please refer article [Azure Blob Storage](http://www.sharecareinspire.com/azure-storage-part-2-blob-storage/)

Blob Storage

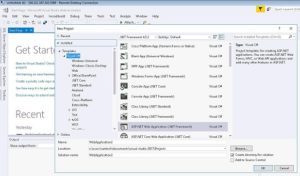
### **Step 2: Setup development environment**

You need to install Visual Studio 2017; I am using Azure VM with Visual Studio 2017 for development environment. If you want to know more about how to create VMs with Specific software image in Azure please refer article [Virtual Machine in Azure](http://www.sharecareinspire.com/virtual-machine-vm-in-azure/).

Development Environment

### **Step 3: Create new ASP.NET Web App**

I decided to create ASP.NET Web app, but you can even select console app or windows form app, which ever you are comfortable with.

Create Web App

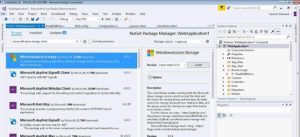
I have selected MVC template, it’s up to you which template you want to select. Code to read and update Azure Blob storage remains same irrespective of template.

### **Step 4: Use NuGet to install the required packages**

There are two packages you need to reference in your project to complete this tutorial:

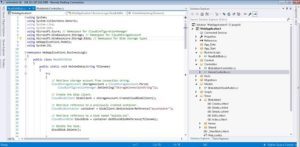
[Microsoft Azure Storage Client Library for .NET](https://www.nuget.org/packages/WindowsAzure.Storage/): This package provides programmatic access to data resources in your storage account.

[Microsoft Azure Configuration Manager library for .NET](https://www.nuget.org/packages/Microsoft.WindowsAzure.ConfigurationManager/): This package provides a class for parsing a connection string in a configuration file, regardless of where your application is running.

Install Nuget Packages

### **Step 5: Create Views, Controller for Blob Details**

Create views and controller for blob details, assumption is that you are aware of MVC pattern, hence skipping the steps for creating views, controller and making changes. Views are used to CRUD operations.

MVC Solution Structure

### **Step 6: Configure your storage connection string**

The Azure Storage Client Library for .NET supports using a storage connection string to configure endpoints and credentials for accessing storage services. The best way to maintain your storage connection string is in a configuration file.

Add StorageConnectionString in web.config, make sure you copy connection string from the BLOB you created in Azure Portal,  following screen snap shows how to get the connection string.

  <appSettings>

    <add key="StorageConnectionString" value="DefaultEndpointsProtocol=https;AccountName=blobstoragesci;AccountKey=qLYzEE72GWvhunLPkkZt9I2mPo5NP3HOaLDuqTIzXh/kqqrSr8a9ZAuvZKzwOnm7estxeE2u8aJEMULY0vRnfA==;EndpointSuffix=core.windows.net" />

  </appSettings>

Navigate to the Blob -> Access Keys (under settings section in properties blade), copy connection string, you can copy from either of the keys; they act as a backup keys, if one is compromised for some reason you can immediately use another one.

BLOB Access Keys and Connection String

### **Step 7: Create class for model and business logic to Read and Edit Blob Storage data**

Create following classes, which acts as a business layer which interacts with Azure Blob Storage, better not to add code directly to controller to achieve separation of concerns. Call methods of this class from controller of Blobdetails for actions like read, upload, delete, and download.

Create View model class for holding data to transfer from business layer to presentation layer.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace WebApplication1.Models

{

    [Table("Blobdetails")]

    public class BlobdetailsViewModels

    {

        public BlobdetailsViewModels()

        {

        }

        public BlobdetailsViewModels(string blobname, DateTime blobdate, string blobtype, long blobsize)

        {

            this.BlobName = blobname;

            this.BlobModifiedData = blobdate;

            this.BlobType = blobtype;

            this.BlobSize = blobsize;

        }

        [Key]

        public string BlobName { get; set; }

        public DateTime BlobModifiedData { get; set; }

        public string BlobType { get; set; }

        public long BlobSize { get; set; }

    }

}

Create class for business logic to perform actions on Azure BLOB storage, this is the important class file which interacts with Azure BLOB.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using Microsoft.Azure; // Namespace for CloudConfigurationManager

using Microsoft.WindowsAzure.Storage; // Namespace for CloudStorageAccount

using Microsoft.WindowsAzure.Storage.Blob; // Namespace for Blob storage types

using WebApplication1.Models;

using System.IO;

namespace WebApplication1.BusinessLogic

{

    public class ReadEditBlob

    {

        public static void DeleteData(string filename)

        {

            try

            {

                // Retrieve storage account from connection string.

                CloudStorageAccount storageAccount = CloudStorageAccount.Parse(

                    CloudConfigurationManager.GetSetting("StorageConnectionString"));

               // Create the blob client.

                CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

               // Retrieve reference to a previously created container.

                CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

               // Retrieve reference to a blob named "myfile.txt".

                CloudBlockBlob blockBlob = container.GetBlockBlobReference(filename);

               // Delete the blob.

                blockBlob.Delete();

            }

            catch (Exception)

            {

                throw;

            }

        }

       public static List<BlobdetailsViewModels> ReadBlobData()

        {

            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("StorageConnectionString"));

           // Create the blob client.

            CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

           // Retrieve reference to a previously created container.

            CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

           List<BlobdetailsViewModels> blobdetailsViewModels = new List<BlobdetailsViewModels>();

           // Loop over items within the container and output the length and URI.

            foreach (IListBlobItem item in container.ListBlobs(null, false))

            {

                if (item.GetType() == typeof(CloudBlockBlob))

                {

                    CloudBlockBlob blob = (CloudBlockBlob)item;

                    blobdetailsViewModels.Add(new BlobdetailsViewModels(Path.GetFileName(blob.Uri.AbsolutePath), blob.Properties.LastModified.Value.LocalDateTime, blob.BlobType.ToString(), blob.Properties.Length));

              }

            }

            return blobdetailsViewModels;

        }

       public static void UploadFile(string filelocation)

        {

            // Retrieve storage account from connection string.

            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(

                CloudConfigurationManager.GetSetting("StorageConnectionString"));

           // Create the blob client.

            CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

           // Retrieve reference to a previously created container.

            CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

           // Retrieve reference to a blob named "newfile".

            CloudBlockBlob blockBlob = container.GetBlockBlobReference("newfile");

           // Create or overwrite the "myblob" blob with contents from a local file.

            using (var fileStream = System.IO.File.OpenRead(filelocation))

            {

                blockBlob.UploadFromStream(fileStream);

            }

        }

       public static string DownloadBlobToString(string blobname)

        {

            // Retrieve storage account from connection string.

            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(

                CloudConfigurationManager.GetSetting("StorageConnectionString"));

           // Create the blob client.

            CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

           // Retrieve reference to a previously created container.

            CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

           CloudBlockBlob blockBlob2 = container.GetBlockBlobReference(blobname);

           string text;

            using (var memoryStream = new MemoryStream())

            {

                blockBlob2.DownloadToStream(memoryStream);

                text = System.Text.Encoding.UTF8.GetString(memoryStream.ToArray());

            }

            return text;

        }

    }

}

### **Step 8: Testing web application**

Deploy web app to Azure and its ready for testing.

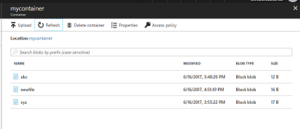
Navigate to Home Page of web app

Home Page

Click on the blob details link to read data from blob storage and list on web page

BLOB Data

Following image shows data in Blob storage in Azure.

BLOB Data

# BLOB Shared Access Signature

## **Why Secure Access?**

In previous article ([Part 3](http://www.sharecareinspire.com/azure-storage-part-3-read-and-edit-blob-storage-data-from-azure-web-app/)) we have seen that you can use keys and connection string to access and modify data in blobs. But those keys and connection string are very sensitive and if compromised, can be misused by any one which may lead to destruction of you data in Azure Blobs.

OK, so it is understood that those keys need to be protected, but what if you have your partners who want to access some data and store some data in your Azure Blobs.

Very simple, share the keys with them right? Ah wait No ….

Microsoft Azure has given you more secure ways to deal with such scenarios which is called Shared Access Signature, this allows the client to access the containers and blobs in your storage account without knowing your storage account keys.

Secure access to blob storage means a secure connection for data transfer, and access to the Blobs would be given through authentication and authorization.

Blobs support both HTTP and secure HTTPS requests. It is recommended to use HTTPS connections for data transfer.

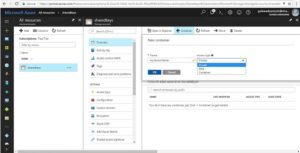
## **What are default permissions on Blobs?**

At the time of creation of Blob container, you can set the permissions for each container in blob storage to one of three values:

Private – no public access to blobs or containers, you can only access the blobs and containers if you have the storage account name and key, or you use a shared access signature.

Blob – public read access for blobs, anybody with the URL to a blob in the container can read the blob and the blob properties and metadata.  shared access signature is required with appropriate permissions to read or write content to blob.

Container – Public read access for blob containers and blobs, you can read properties of container, blob and also can read the content of content and blob. However can’t modify, upload, or delete blobs.

Storage Access Types

## **How to provide secure access to Blobs?**

Using Shared Access Signature you can Grant restricted access rights to containers and blobs. You can provide a shared access signature to users you don’t trust with your storage account key or your clients. You can give them a shared access signature that will grant them specific permissions to the resource for a specified amount of time.

## **How a shared access signature works?**

A shared access signature is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters. The token indicates how the resources may be accessed by the client. One of the query parameters, the signature, is constructed from the SAS parameters and signed with the account key. This signature is used by Azure Storage to authenticate.

The SAS token is a string you generate on the client side. A SAS token you generate with the storage client library. You can create an unlimited number of SAS tokens on the client side.

When a client provides a SAS URI to Azure Storage as part of a request, the service checks the SAS parameters and signature to verify that it is valid for authenticating the request. If the service verifies that the signature is valid, then the request is authenticated. Otherwise, the request is declined with error code 403 (Forbidden).

## **How to Create Share Access Signature?**

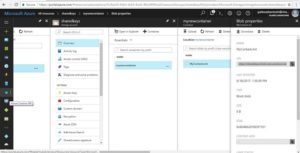
You as owner of the storage account need to generate a shared access signature using account key and provide it to the clients for consumption.

Will see example in upcoming articles.

## **How does SAS URI looks like?**

Let’s consider I have a blob entry with text file which has all my contacts

<https://sharedkeys.blob.core.windows.net/mynewcontainer/MyContacts.txt>

Blob URI

If this resource has to be accessed by the client’s applications, they need to access this resource with URI which looks something like blow RUI.

<https://sharedkeys.blob.core.windows.net/mynewcontainer/MyContacts.txt?sv=2015-04-05&st=2015-04-29T22%3A18%3A26Z&se=2015-04-30T02%3A23%3A26Z&sr=b&sp=rw&sip=168.1.5.60-168.1.5.70&spr=https&sig=Z%2FRHIX5Xcg0Mq2rqI3OlWTjEg2tYkboXr1P9ZUXDtkk%3D>

Does it look wacky?  Hmmm, let’s have a close look at it, and see what all things it contains.

* Blob URI: This is the address of the blob as stated above.  
  [https://sharedkeys.blob.core.windows.net/mynewcontainer/MyContacts.txt](https://sharedkeys.blob.core.windows.net/mynewcontainer/MyContacts.txt%C2%A0)
* Storage services version (sv): This tells which version of the storage services to use, in above example: sv=2015-04-05**,**this is optional, and will be set to the newest version available if not specified.
* Start Time (st): This is the time the SAS becomes valid. If omitted, the SAS is effective immediately. This must be in ISO 8061 format. In above example: st=2015-04-29T22%3A18%3A26Z
* Expiration Time (se): This is the time after which the SAS is no longer valid. In above example: se=2015-04-30T02%3A23%3A26Z
* Storage Resource (sr): This tells if the resource is a blob, queue, etc. in above example: sr=b, which says the resource is a blob, (sr=c for container).
* Permissions (sp): This defines what operations can be performed against the storage resource. In above example: sp=rw, which grants Read (r) and Write (w) access.
* Shared IP range: The range of IP addresses from which a request will be accepted in above example, sip=168.1.5.60-168.1.5.70
* Shared Protocol: Protocol on which data will be exchanged, only requests using HTTPS are permissible.In above example, spr=https
* Signature (sig): This is used to authenticate the blob. Signature uses the SHA256 algorithm, then encoded using Base64 encoding. (That sounds hard, doesn’t it? Fortunately, the storage client library has a call to create this for you, unless you just like to do things the hard way.) in above example: sig=Z%2FRHIX5Xcg0Mq2rqI3OlWTjEg2tYkboXr1P9ZUXDtkk%3D

# Read BLOB data using SAS Token

## **How use Share Access Signature?**

As owner of the blob, you need to create shared access signature and provide the same to your clients.

### **Step 1: Create SAS token**

The following code example creates an account SAS that is valid for the Blob and File services, and gives the client permissions read, write, and list permissions to access service-level APIs. The account SAS restricts the protocol to HTTPS, so the request must be made with HTTPS.

Create console application and paste following code, to know about how to get the connection string of blob and installing nuget package, please refer article Part 3. Run the application to get SAS token.

Following application creates SAS token for the BLOB file MyContacts.txt

Blob File MyContacts

Following is the content of file.

MyContacts File

using Microsoft.WindowsAzure.Storage;

using Microsoft.WindowsAzure.Storage.Blob;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

    class Program

    {

        static void Main(string[] args)

        {

           Console.WriteLine(GetBlobSasUri());

            Console.ReadLine();

        }

        static string GetBlobSasUri()

        {

            // To create the account SAS, you need to use your shared key credentials. Modify for your account.

            const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=sharedkeys;AccountKey=oO/+Pe+K4AvHL+R/svio2NcNrcZ0gDg/ag+oDWI5KfOslZgyqAsMe5RWzKL5tKzqCjGKU9RG2a0oyhksAqH3NA==;EndpointSuffix=core.windows.net";

            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);

            CloudBlobClient blobClientWithSAS = storageAccount.CreateCloudBlobClient();

            CloudBlobContainer container = blobClientWithSAS.GetContainerReference("mynewcontainer");

            //Get a reference to a blob within the container.

            CloudBlockBlob blob = container.GetBlockBlobReference("MyContacts.txt");

            //Set the expiry time and permissions for the blob.

            //In this case, the start time is specified as a few minutes in the past, to mitigate clock skew.

            //The shared access signature will be valid immediately.

            SharedAccessBlobPolicy sasConstraints = new SharedAccessBlobPolicy();

            sasConstraints.SharedAccessStartTime = DateTimeOffset.UtcNow.AddMinutes(-5);

            sasConstraints.SharedAccessExpiryTime = DateTimeOffset.UtcNow.AddHours(24);

            sasConstraints.Permissions = SharedAccessBlobPermissions.Read | SharedAccessBlobPermissions.Write;

            //Generate the shared access signature on the blob, setting the constraints directly on the signature.

            string sasBlobToken = blob.GetSharedAccessSignature(sasConstraints);

           //Return the URI string for the container, including the SAS token.

            return blob.Uri + sasBlobToken;

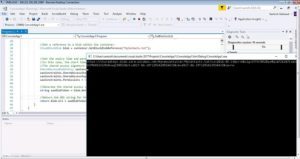
        }

    }

}

Application returns following URI

<https://sharedkeys.blob.core.windows.net/mynewcontainer/MyContacts.txt?sv=2016-05-31&sr=b&sig=a3AqebhPpI%2FnpTTYHUEzNAR4WWMYf099vxWvAGEUoP8%3D&st=2017-06-18T12%3A07%3A33Z&se=2017-06-19T12%3A12%3A33Z&sp=rw>

Application returns SAS URI

### **Step 2: Use SAS token in Client’s or 3rd party application code**

Once you create SAS token at your end, you need to provide that storage account name and token to your clients or 3rd parties who want to consume your blob. I have used console application to read content of blob using SAS token.

using Microsoft.WindowsAzure.Storage;

using Microsoft.WindowsAzure.Storage.Auth;

using Microsoft.WindowsAzure.Storage.Blob;

using Microsoft.WindowsAzure.Storage.Shared.Protocol;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ThirdPartyApplication

{

    class Program

    {

        static void Main(string[] args)

        {

            string sasToken = "https://sharedkeys.blob.core.windows.net/mynewcontainer/MyContacts.txt?sv=2016-05-31&sr=b&sig=a3AqebhPpI%2FnpTTYHUEzNAR4WWMYf099vxWvAGEUoP8%3D&st=2017-06-18T12%3A07%3A33Z&se=2017-06-19T12%3A12%3A33Z&sp=rw";

            UseAccountSAS(sasToken);

            Console.ReadLine();

        }

        static void UseAccountSAS(string sasToken)

        {

            CloudBlob blobsas = new CloudBlob(new Uri(sasToken));

            MemoryStream msRead = new MemoryStream();

            using (msRead)

            {

                blobsas.DownloadToStream(msRead);

                msRead.Position = 0;

                using (StreamReader reader = new StreamReader(msRead, true))

                {

                    string line;

                    while ((line = reader.ReadLine()) != null)

                    {

                        Console.WriteLine(line);

                    }

                }

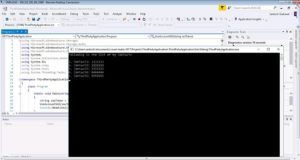
            }

        }

    }

}

Run the application and you will see the content of file on the screen.

Client Application returns content of BLOB file