Managing Scientific Data with Microsoft Azure Storage

Introduction

Microsoft Azure Storage is a cloud-based distributed storage service that you can use to store many different types of data, ranging from images and videos to text files, log files, and sensor data. The Microsoft Azure Storage service provides numerous benefits. It scales to accommodate huge amounts of data, permits access to that data easily from any location at any time, and provides load balancing to support variable traffic volumes. Furthermore, Microsoft Azure Storage makes three copies of your data to ensure that it is always accessible if a storage node fails or a data center becomes unavailable. Optionally, it provides geo-redundant storage by replicating your data to an alternate data center region hundreds of miles away, but in the same geo-political area (known as a *geo*). You can also set up your storage account with the Microsoft Azure Content Delivery network to cache data closer to your users and thereby provide a consistent high-performance user experience regardless of their distance from a data center.

Microsoft Azure Storage is an ideal storage solution for scientific data. It is well suited for a variety of scenarios, such as:

- Storing instrumentation data that is growing exponentially and rapidly
- Providing data to other researchers with high-performance connectivity, thereby reducing the time required to move large data sets.
- Accessing data from anywhere in the world with an Internet connection, whether in the field or in the laboratory, when getting to access to a local drive is not practical
- Using virtual resources on demand as needed, without the need to plan for building and maintaining a scalable infrastructure
- Co-locating data with Microsoft Azure compute resources, such as a scientific application running in Microsoft Azure or data analysis with HDInsight, Microsoft's distribution of Hadoop on Microsoft Azure

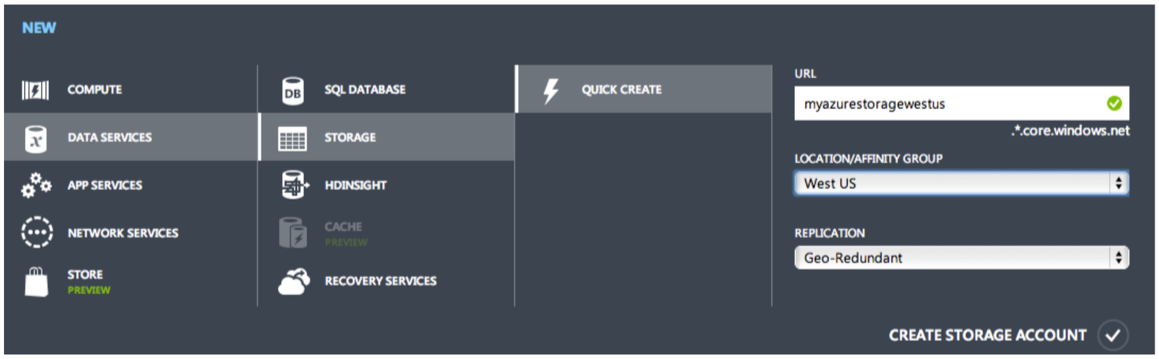
Microsoft Azure storage types

To start working with the Microsoft Azure Storage service, you create a storage account to serve as the top-level namespace that you use to access storage. By default, you can set up a maximum of five storage accounts for each Microsoft Azure subscription; additional accounts can be enabled through a support request. Within each account, you can store up to 200 TB of data.

To create a storage account in the Microsoft Azure Management Portal:

1. Click the **New** button, click **Data Services**, click **Storage**, and then click **Quick Create**.
2. Type a URL for your storage account, select a location or affinity group, and specify whether to enable geo-replication

3. Click **Create Storage Account**. The URL you select here becomes host name in the URI that you use to access your data in storage.



URL: myazurestoragewestus .*.core.windows.net

LOCATION/AFFINITY GROUP: West US

REPLICATION: Geo-Redundant

Figure 1: Creating a new storage account

After you create your storage account, your next step is to load data into storage. You can choose from the following options of storage types:

- Use blob storage to store any type of file.
- For structured, non-relational tabular data, you can use table storage.
- If you are running applications in Microsoft Azure, you can use queue storage to pass messages between a Microsoft Azure web role and worker role, or to store messages to process asynchronously. Queue storage is not the focus of this paper, but you can read more about it at Queue Service Concepts and Queue Service REST API.

Microsoft Azure Storage explorer tools

Although the Management Portal provides you with access to many administrative activities, you can use public HTTP or HTTPS REST endpoints to interact with storage, write an application to transfer data and manage your storage, or acquire a third-party tool. Experiment with some of the following tools to see which you prefer:

- AzCopy (Windows)
- Azure Storage Explorer (Windows)
- Azure Management Studio (Windows)
- CloudBerry Explorer (Windows)
- CloudXplorer by ClumsyLeaf (Windows)
- Azure Tools for Microsoft Visual Studio (Windows)
- Zudio (any operating system, web-based)

You can view a list of other storage explorer tools in the Microsoft Azure Storage team blog. If you prefer to create your own application, see How to use the Microsoft Azure Blob Storage Service in .NET and How to: Programmatically access table storage.

Note: Rather than use an application to load blob data into storage, you can take advantage of the Microsoft Azure Import/Export service. This might be a more time- and cost-effective option when you have extremely large amounts of data. You simply create a job to notify the data center and then send one or more hard drives loaded with encrypted data to a data center where the data is uploaded to your storage account. Conversely, you can send empty drives to the data center to have data downloaded to the drives and then returned to you. To learn more, see Using the Microsoft Azure Import/Export Service to Transfer Data to Blob Storage.

Microsoft Azure Blob service

A blob is simply a file. It can be binary data, such as an image or an audio file, or it can be text data. You organize blobs inside of containers that are associated with your storage account, as shown in Figure 2. There is no limit to the number of containers you can create in an account, nor is there a limit to the number of blobs that you can store in a container (other than the maximum size of the storage account).

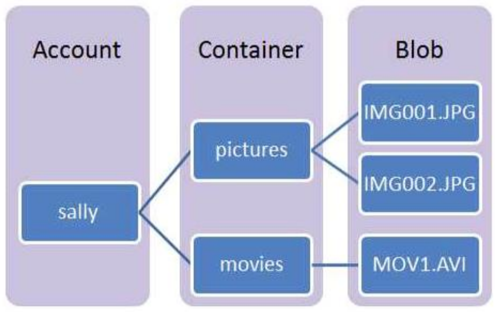


Figure 2: Blob storage components

Blobs

Although a blob can be any type of file, it falls into one of the following categories: block blob or page blob. The type of blob determines how Microsoft Azure Storage manages both storage of the blobs and operations involving the blobs.

Block blob

You use this type of blob for efficient upload and download (streaming) workloads of audio or video files and for most document, text, and image file types. Essentially, the client application performing the upload deconstructs a large blob into a sequence of blocks, with each block having a unique block ID. Each block can be no larger than 4 MB. The total size of a single blob cannot exceed 200 GB and must contain no more than 50,000 blocks. A client application can upload the blocks in parallel to minimize the time needed to transfer the entire blob. In addition, it can upload the blocks out of sequence. That way, if a block fails to upload correctly, the client application can retry the upload of that block only and not force the entire upload to restart.

Page blob

You use this type of blob for files that you want to optimize for random read-write operations such as a database or a file system. Each page blob contains a collection of 512-byte pages with a maximum size of 1 TB. As you create the page blob or update its contents, you write one or more pages to storage and specify an offset and a range that corresponds to 512-byte page boundaries. Existing pages remain unchanged.

Blob snapshot
You can create a snapshot of a blob at any time to save its current state as a read-only blob. You can interact with a blob snapshot just like a blob, except you cannot modify it. A blob snapshot has the same URI as the parent blob, but has the date and time of creation appended to the URI.

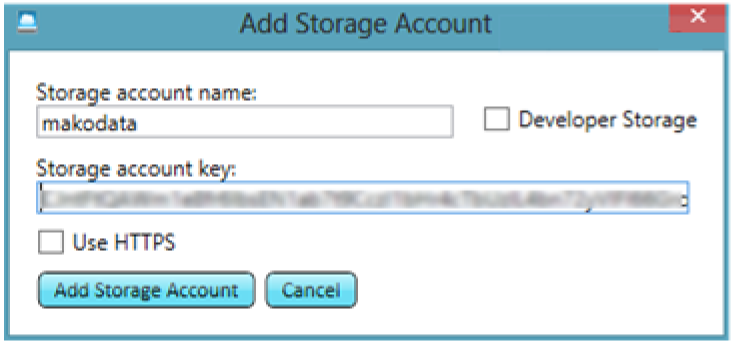
Blob storage and Azure Storage Explorer
An easy way to work with your storage account is to download and install Azure Storage Explorer. When you launch the application:

1. Click the **Add Account** button, type in your storage account name and storage account key, as shown in Figure 3.
2. Click **Add Storage Account**.
3. Optional: you can select the **Use HTTPS** check box to use a secure connection for sensitive data.

To locate your storage account key:

- Visit the Microsoft Azure Management Portal and access the **Storage** page.
- Select the storage account, and then click the **Manage Access Keys** button at the bottom of the page.
- Click the icon to the right of the **Primary Access Key** to copy the key to your clipboard,

Once you have the key, you can paste it into the **Add Storage Account** dialog box in Azure Storage Explorer.



Storage account name: makodata

Storage account key:

Figure 3: Add storage account in Azure Storage Explorer

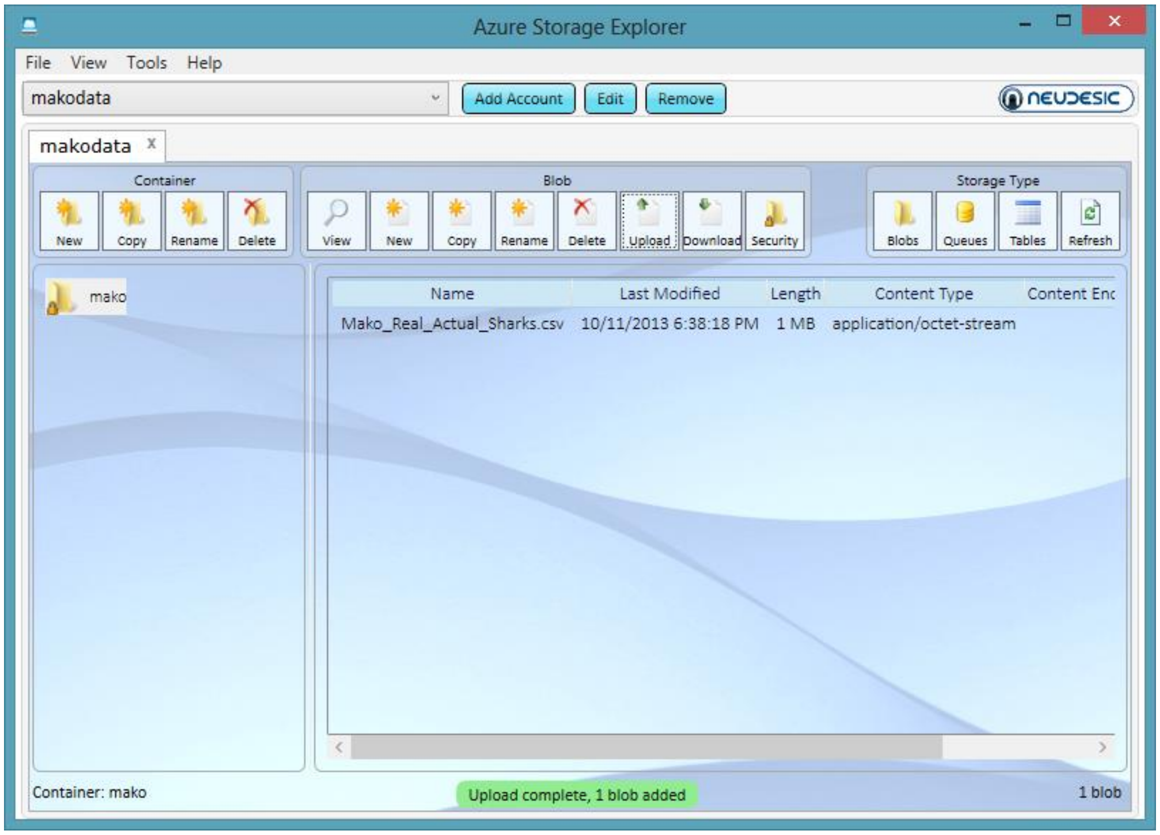
Next, you need to create a container in your storage account to hold one or more blobs. To create the blob container:

1. Click the **New** button in the **Container** section of the ribbon.
2. Type a name for the container.
3. Click **Create Container**.

Next, you need to access data to put into storage. For this demonstration:

1. Download the Mako_Real_Actual_Sharks CSV file to your computer.
2. In Azure Storage Explorer, select the container you created, click the **Upload** button, and select the CSV file that you just downloaded.

When the upload completes, you can see the metadata for the newly created blob in the selected container, as shown in Figure 4, including its name, modification date, length, content type, content encoding, and content language.



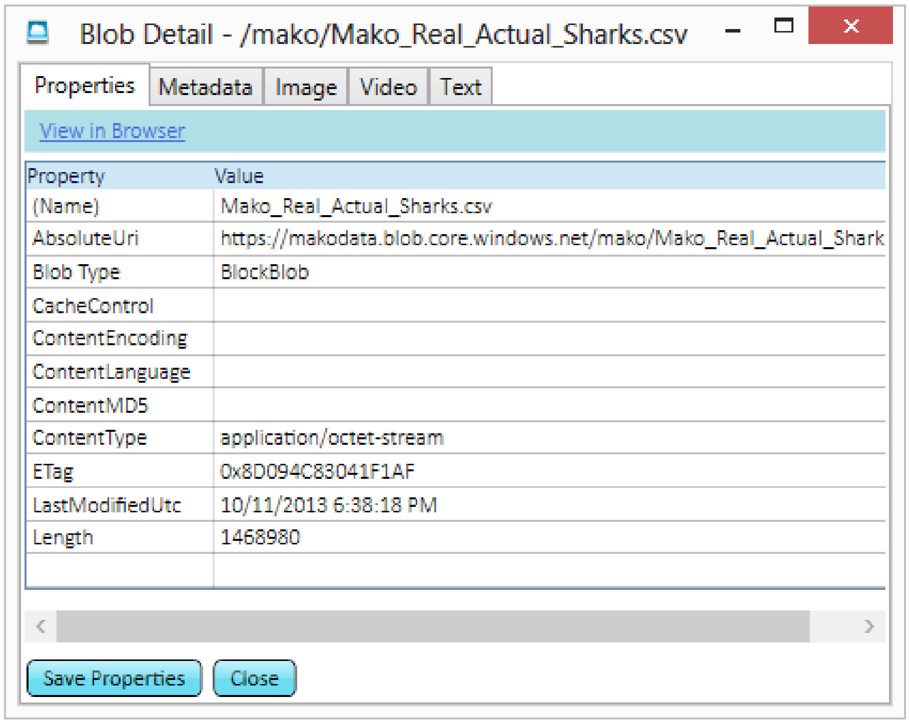
Name	Last Modified	Length	Content Type	Content Enc
Mako_Real_Actual_Sharks.csv	10/11/2013 6:38:18 PM	1 MB	application/octet-stream	

Container: mako
Upload complete, 1 blob added
1 blob

Figure 4: Blob in Microsoft Azure Storage container

Azure Storage Explorer allows you to manage all types of storage components: containers, blobs, queues, and table. You can create new components, make copies of existing components, or delete components, in addition to renaming them. In addition, you have the option to configure security on blobs and containers by setting the access level at minimum and creating shared access signatures and shared access policies—all of which are described later in this document in the "Security and Microsoft Azure Storage" section.

A viewer in Azure Storage Explorer allows you to see the properties and contents of your blobs. Double-click the blob name in the list or select it and click the **View** button to view the **Blob Detail** dialog box, as shown in Figure 5. You can add properties as name/value pairs on the **Metadata** tab to give context to the blob. For example, you could have an Author or Project property. You can also add metadata to containers.



Property	Value
(Name)	Mako_Real_Actual_Sharks.csv
AbsoluteUri	https://makodata.blob.core.windows.net/mako/Mako_Real_Actual_Shark
Blob Type	BlockBlob
CacheControl	
ContentEncoding	
ContentLanguage	
ContentMD5	
ContentType	application/octet-stream
ETag	0x8D094C83041F1AF
LastModifiedUtc	10/11/2013 6:38:18 PM
Length	1468980

Figure 5: Blob Detail in Azure Storage Explorer

Blob storage and IPython

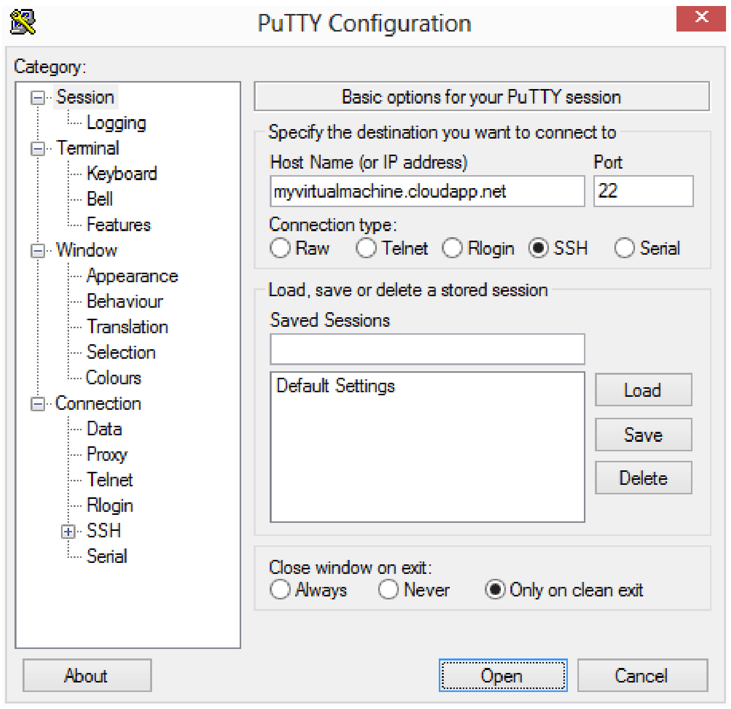
For greater control over blob management, you can write your own storage logic for your application by using one of many client libraries that implement the REST API for Microsoft Azure Storage Services, including Python. To minimize the number of tools you need to install on your local computer, you can use an IPython Notebook server for development and write your code in a web application. The IPython Notebook server can even be hosted in a Microsoft Azure Virtual Machine.

Running an IPython Notebook server on a Microsoft Azure Virtual Machine

A quick way to get started with development using an IPython Notebook server is to use an image from VM Depot that has the necessary components installed. You can view a list of available images on the VM Depot website and use the search bar to filter the list for images tagged with a specific keyword, such as IPython. For instructions on how to create a virtual machine from an image, such as the Azure Data Analysis image, refer to "Using and Contributing Virtual Machines to VM Depot" (available from the Microsoft Azure for Research Technical Papers website).

Once you have the Azure Data Analysis virtual machine provisioned and started, you can connect to it with a PuTTY client.

1. In the **Host Name (or IP address)** box, type the DNS name for your virtual machine, as shown in Figure 6.
2. Click **Yes** to trust the connection, and then type in the user name and password that you established when creating the virtual machine.



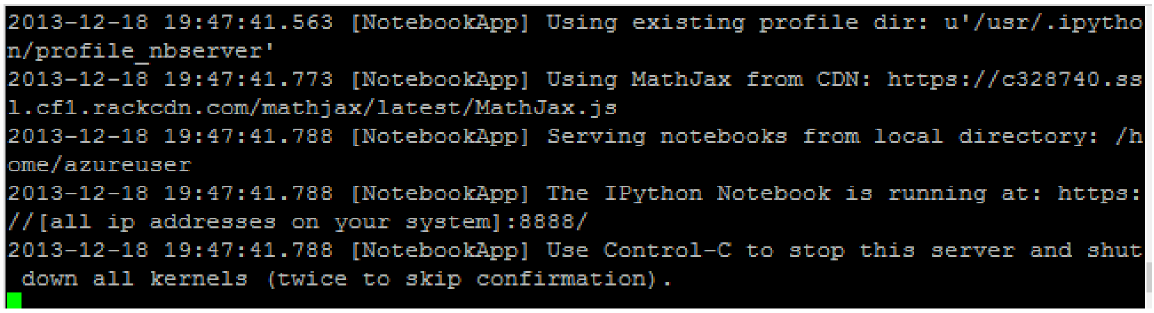
Specify the destination you want to connect to
Host Name (or IP address): myvirtualmachine.cloudapp.net
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Saved Sessions
Close window on exit: Always Never [✓ Only on clean exit]

Figure 6: SSH connection to Linux virtual machine

3. In the PuTTY window, type the following command to start the IPython Notebook server:

```
sudo ipython notebook –profile=nbserver
```

You might need to supply your password again to execute this command. The notebook server is successfully started when you see the message to use Control-C to stop the server, as shown in Figure 7.



```
2013-12-18 19:47:41.563 [NotebookApp] Using existing profile dir: u'/usr/.ipytho
n/profile_nbserver'
2013-12-18 19:47:41.773 [NotebookApp] Using MathJax from CDN: https://c328740.ss
l.cf1.rackcdn.com/mathjax/latest/MathJax.js
2013-12-18 19:47:41.788 [NotebookApp] Serving notebooks from local directory: /h
ome/azureuser
2013-12-18 19:47:41.788 [NotebookApp] The IPython Notebook is running at: https:
//[all ip addresses on your system]:8888/
2013-12-18 19:47:41.788 [NotebookApp] Use Control-C to stop this server and shut
 down all kernels (twice to skip confirmation).
```

Figure 7: IPython Notebook Server successfully started

Now you are ready to start coding. To do this, open your browser and use the DNS address for your virtual machine, https://<virtual-machine>.cloudapp.net. Be sure to use the HTTPS protocol. You will see a warning about the security, which you can ignore for this demonstration, and then you must respond to the prompt for the login password for the IPython Notebook on first load. For the IPython Notebook server, the password is **Elastacloud123**.

Writing data directly into blob storage

Using IPython, you can develop an application that connects to storage, writes data into a local file, and then uploads the file as a blob into a container. In the sample code below, these steps are achieved by using the **BlobService** object and the **put_blob** method. For other code samples demonstrating how to download (get_blob) and perform other operations, see How to Use the Blob Storage Service from Python.

Click the **New Notebook** button in the top right corner of the IPython notebook window, and then copy the following code sample and paste it into the gray box in the browser. Be sure to replace the account, key, and container name variables with values specific to your storage account.

```
#first import required modules
from azure.storage import *
import os
import csv
import numpy
from collections import defaultdict
import time

#replace the following values with your own account name, primary access key, and
container name
account = 'myazurestoragewestus'
key = 'mdJnNXPYS1cC5T4pf+9yva/ …. etc'
containername = 'mystorage'

#get a handle to your account
blob_service = BlobService(account_name=account, account_key=key)

#create a new file a blob into a container
open(r'mytextfile.txt', 'w').write("This is a sample text file.")

#upload the blob into the container
textblob = open(r'mytextfile.txt', 'r').read()
blob_service.put_blob(containername, 'mytextfile.txt', textblob,
x_ms_blob_type='BlockBlob')

#use azure storage explorer to find the mytextfile.txt file
```

To execute the code, click the **Run Cell** button (with the arrow icon), as shown in Figure 8. Then use another tool like Azure Store Explorer to view the uploaded file.

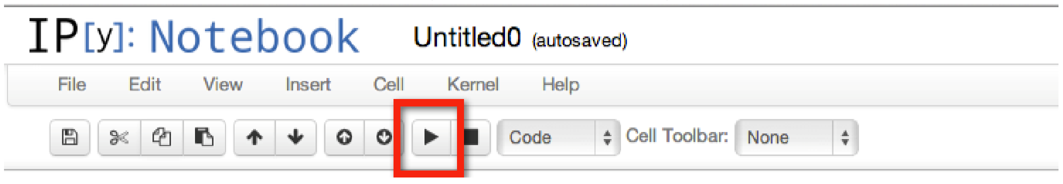


Figure 8: Run Cell button executes code

Removing a blob from storage

You can also use IPython to remove a blob from storage, as shown in the next code sample. In this example, you use the **delete_blob** method. Paste this code into the notebook document, and then execute it. When it completes, confirm that the blob no longer exists in storage.

```
#Remove mytextfile.txt
os.remove(r'mytextfile.txt')

#delete the blob remotely
blob_service.delete_blob(containername, 'mytextfile.txt')

#check the azure storage explorer again, the file is removed
```

Blob storage and AzCopy

AzCopy is a command-line utility that you can use to copy blobs between storage accounts or to a different container in the same storage account. This is a much faster process than downloading the blob locally and then uploading it again to a different target location. To use this utility, you must know the URI for the source and target accounts and containers in addition to the keys for the source and target. Open a command prompt window and type the respective commands shown below after replacing the source and target names and keys:

- You can move all blobs from a source container in a source account to a target container in a different storage account.

  ```
  AzCopy https://<sourceaccount>.blob.core.windows.net/<sourcecontainer>/
  https://<targetaccount>.blob.core.windows.net/<targetcontainer>/
  /sourcekey:<key> /targetkey:<key> /S
  ```

- By adding the /MOV flag to the command, you can delete blobs from the source after they have been successfully copied.

  ```
  AzCopy https://<sourceaccount>.blob.core.windows.net/<sourcecontainer>/
  https://<targetaccount>.blob.core.windows.net/<targetcontainer>/
  /sourcekey:<key> /targetkey:<key> /MOV /S
  ```

- When you have blobs with snapshots, you can use the utility with the /SNAPSHOT flag to move not only the base blobs, but also the blob snapshots. The snapshots will be renamed to conform to this pattern: [blob-name][snapshot-time][extension].

  ```
  AzCopy https://<sourceaccount>.blob.core.windows.net/<sourcecontainer>/
  https://<targetaccount>.blob.core.windows.net/<targetcontainer>/
  /sourcekey:<key> /targetkey:<key> /S /SNAPSHOT
  ```

- If you repeatedly execute the utility with the same source and target, you can put the URIs in a text file like this:

  ```
  #URI of Source Container
  https://<sourceaccount>.blob.core.windows.net/<sourcecontainer>/
  ```

```
#URI of Destination Container
https://<targetaccount>.blob.core.windows.net/<targetcontainer>/
```

Then you can reference the file in the command line in lieu of the URIs:

```
AzCopy /@:C:\myAzCopy.txt /sourcekey:<key> /targetkey:<key> /MOV /S
```

Blob storage and Node.js

Similar to Python and AzCopy, CLI and Node.js can be used to copy local files up to Blob Storage, this works on Windows, Linux, and Mac.

CLI – Copy a local file to blob

```
azure storage blob upload -a "japandata" -k "xdJ…storage-key…==" "foo.gz"
"backups" "foo.gz" -q
```

See **azure storage blob upload -h** for details on individual parameters.

Node.js – Copy a local file to blob

```
var azure = require('azure');
var fs = require('fs');

var storageAccountName = 'myazurestorageaccountname';
var storageAccountKey = 'myazurestorageaccountkey';
var storageContainerName = 'mystoragecontainername';

var blobService = azure.createBlobService(
                    storageAccountName, storageAccountKey,
                    storageAccountName + '.blob.core.windows.net');

var fileNameList = [ 'localfile1.json', 'localfile2.json' ];
for (var i=0; i<fileNameList.length; i++) {
    var fileName = fileNameList[i];
    console.log('=> ' + fileName);
    blobService.createBlockBlobFromFile(
        storageContainerName, fileName, fileName,
        {
            contentType: 'application/json',
            cacheControl: 'public, max-age=3600' // max-age in seconds
        },
        function(error) {
            if (error) {
                console.error(error);
            }
    });
}
```

Asynchronous copy

Microsoft Azure has asynchronous copy support for files that are already in blob storage. You can use it to make a copy of a blob within the same storage account, or to copy blobs across storage accounts, even if the data is being copied between data centers. Once you initiate a copy, Microsoft Azure handles all of the activity as a background process. This means that you can initiate the copy of a very large blob from anywhere, including a computer running outside the cloud, but rather than first downloading the blob to the initiating computer and then uploading to the new destination, the Microsoft Azure Blob service can pull it directly into the designated storage account. Consequently, data transfers are as efficient as possible, and client code does not need to wait for transfers to finish. The status of the transfer can be monitored, and in-progress transfers can be canceled. This feature is especially useful for large blobs that are gigabytes or terabytes in size.

Asynchronous copy with PowerShell

```
# SOURCE DATA (public blob, in data center in US East)
# sourceBlob can be a HUGE FILE, gigabytes or terabytes
$sourceBlob = 'https://azuremap.blob.core.windows.net/maps/azuremap.geojson'

# DESTINATION (private container, in data center in Japan West)
$japanAcct = 'japandata'
$japanContainer = 'backups'
$japanKey = 'xdJ…storage-key-goes-here…RA=='$destinationContext = New-
AzureStorageContext `
                      -StorageAccountName $japanAcct `
                      -StorageAccountKey $japanKey

# In this example, the source blob is addressed by a URL that is publicly
# accessible; once the copy request is accepted by the Microsoft Azure Blob
# service, it takes over and handles all the processing.

Start-AzureStorageBlobCopy -AbsoluteUri $sourceBlob `
                           -Context $destinationContext `
                           -DestContainer $japanContainer `
                           -DestBlob 'new.file.name'
```

You can learn more at Start-AzureStorageBlobCopy.

Asynchronous copy with Node.js

After executing “npm install azure”, the following Node.js code will start an asynchronous copy, just like in the PowerShell example.

```
var azure = require('azure');

// SOURCE DATA (public blob, in data center in US East)
// sourceBlob can be a HUGE FILE, gigabytes or terabytes
var sourceBlob =
'https://azuremap.blob.core.windows.net/maps/azuremap.geojson';

// DESTINATION (private container, in data center in Japan West)
var japanAcct = 'japandata';
var japanContainer = 'backups';
var japanKey = 'xdJ…storage-key-goes-here…RA=='
// In this example, the source blob is addressed by a URL that is publicly
// accessible; once the copy request is accepted by the Microsoft Azure Blob
// service, it takes over and handles all the processing.

var blobService = azure.createBlobService(japanAcct, japanKey);
blobService.copyBlob(sourceBlob, japanContainer, 'new.file.name.node', function
(error) {
    if (error != null) {
        console.log(error);
    }
});
```

Microsoft Azure Table storage

Use table storage for high volumes of structured, non-relational data. An analogous type of data is a Microsoft Excel worksheet containing multiple rows of data that can have a different number of populated columns per row and even different data types in the same column across multiple rows. Table storage provides flexibility to your applications without requiring you to define and maintain a strict data model. It is highly scalable with fast query response times.

Table storage components

You can add a table of up to 200 TB in size to your storage account and then associate entities with tables, as shown in Figure 9. An entity is like a row in a table and represents a collection of up to 255 properties. It has a maximum size of 1 MB.

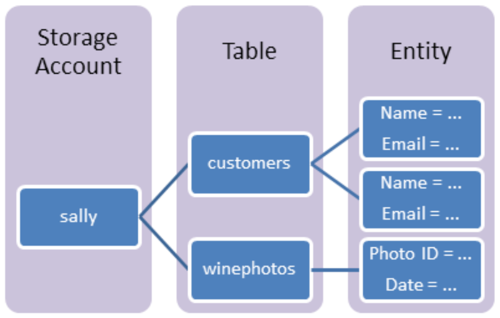


Figure 9: Table storage components

Each property is a name-value pair. There are three required properties: PartitionKey, RowKey, and Timestamp. PartitionKey and RowKey are set by the developer, and the system sets Timestamp. PartitionKey and RowKey form a single clustered index for the table to enable fast query performance. Entities having the same PartitionKey are stored together physically, whereas partitions in the same table can be located on separate storage nodes. Inserts, updates, and deletes in the same partition can be handled as part of the same transaction.

Table storage and Azure Management Studio

Although you can use Azure Storage Explorer to manage tables, another option to consider is Cerebrata's Azure Management Studio. When you open Azure Management Studio, you need to connect to your Microsoft Azure subscriptions.

1. Sign in to your Microsoft Azure account to download your publishsettings file.
2. Select the subscription file to download.
3. In Azure Management Studio, open the **File** menu, point to **Import**, click **Publish Settings**, and select the publishsettings file you just downloaded.

Your subscription displays in the **Connection Group** pane on the left side of the screen.

4. You can open the **Storage Account** node to access existing storage accounts, or right-click the **Storage Account** node and select **New Storage Account** if you need to create a storage account.

5. In the **Connection Group** pane, expand the storage account that you want to use for table storage, right-click **Tables**, point to **New**, and then choose one of the options shown in Figure 10. (For example, you could use the **Table from CSV** option to upload the Mako_Real_Actual_Sharks file into your storage account.)

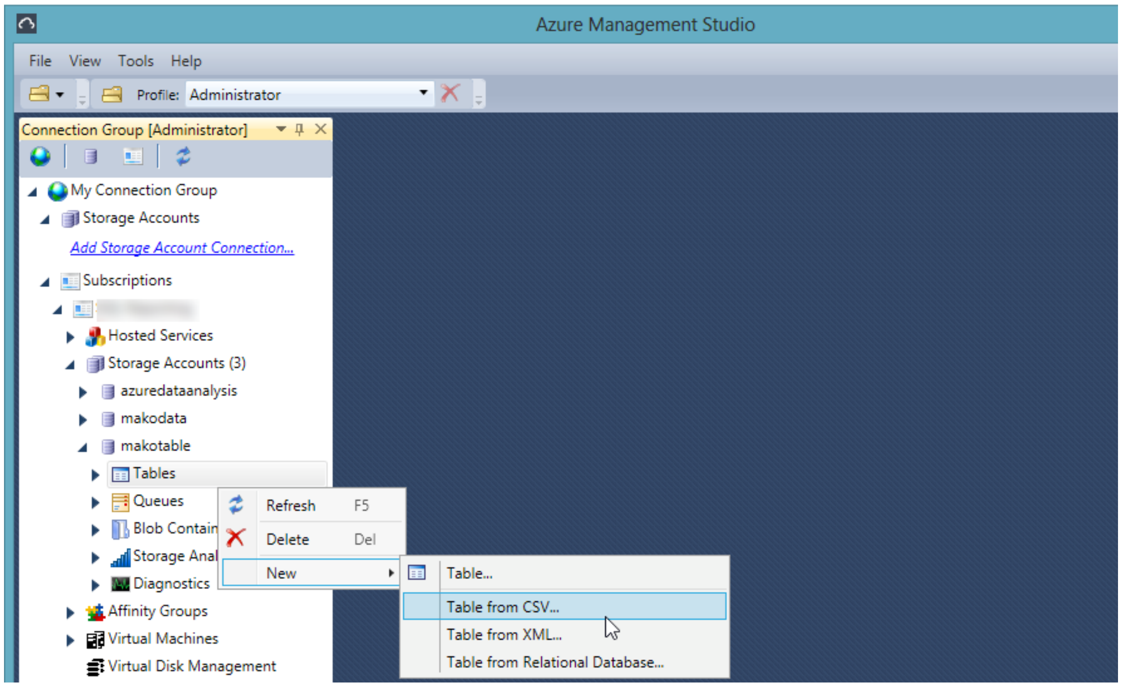
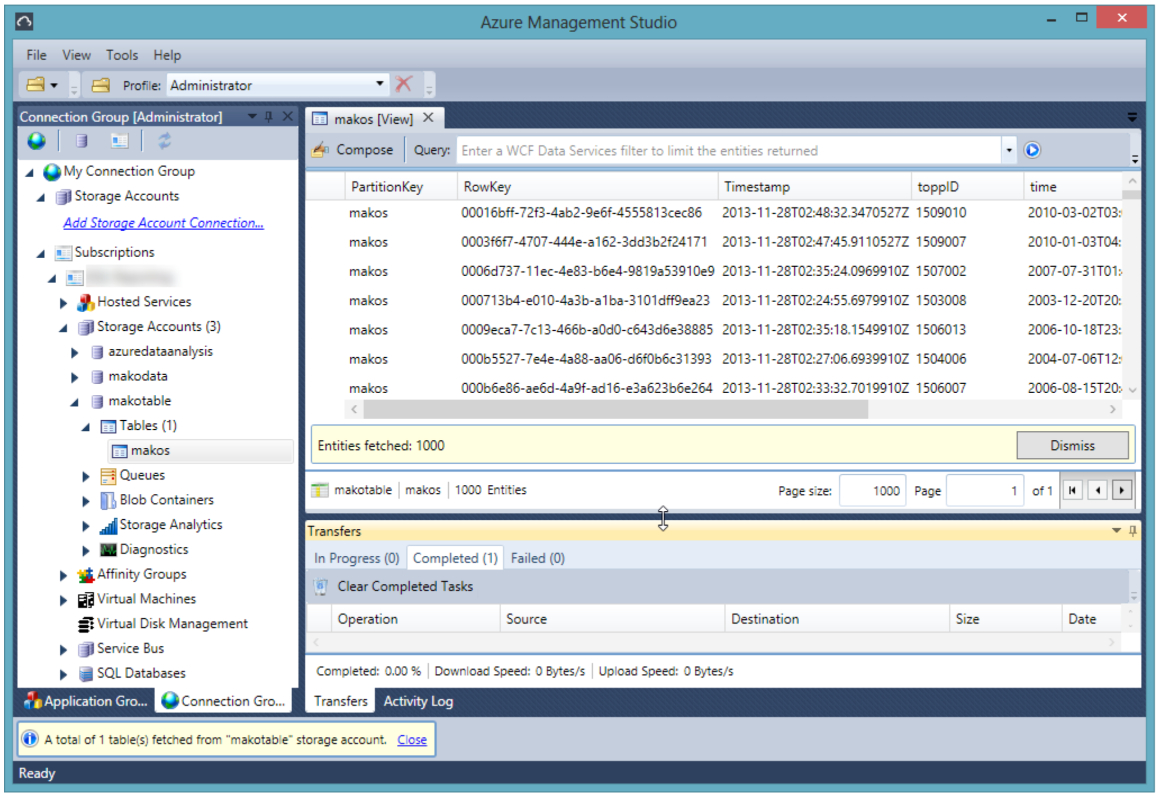


Figure 10: New table creation in Azure Management Studio

After you specify the file name to upload and configure any additional properties specific to your data, such as renaming a column name, the upload begins. You can monitor the upload progress in the **Transfers** pane that displays at the bottom of Azure Management Studio. When the upload is complete, you can view the data in Azure Management Studio, as shown in Figure 11:

1. Expand the **Tables** node.
2. Right-click your table.
3. Select **Open**.



PartitionKey	RowKey	Timestamp	toppID	time
makos	00016bff-72f3-4ab2-9e6f-4555813cec86	2013-11-28T02:48:32.3470527Z	1509010	2010-03-02T03:
makos	0003f6f7-4707-444e-a162-3dd3b2f24171	2013-11-28T02:47:45.9110527Z	1509007	2010-01-03T04:
makos	0006d737-11ec-4e83-b6e4-9819a53910e9	2013-11-28T02:35:24.0969910Z	1507002	2007-07-31T01:
makos	000713b4-e010-4a3b-a1ba-3101dff9ea23	2013-11-28T02:24:55.6979910Z	1503008	2003-12-20T20:
makos	0009eca7-7c13-466b-a0d0-c643d6e38885	2013-11-28T02:35:18.1549910Z	1506013	2006-10-18T23:
makos	000b5527-7e4e-4a88-aa06-d6f0b6c31393	2013-11-28T02:27:06.6939910Z	1504006	2004-07-06T12:
makos	000b6e86-ae6d-4a9f-ad16-e3a623b6e264	2013-11-28T02:33:32.7019910Z	1506007	2006-08-15T20:

Entities fetched: 1000

makotable | makos | 1000 Entities   Page size: 1000 Page 1 of 1

Completed: 0.00 % | Download Speed: 0 Bytes/s | Upload Speed: 0 Bytes/s

A total of 1 table(s) fetched from "makotable" storage account. Close

Figure 11: Data in table storage accessible in Azure Management Studio

Table storage and IPython

Just as you can use IPython to manage blob storage, you can also use it to manage table storage. The following code sample works if you have your environment set up as described in the "Running an IPython Notebook Server on a Microsoft Azure Virtual Machine" section of this document.

Writing data to a table

To demonstrate how to write data to a table, you must have the Mako_Real_Actual_Sharks file on your computer. Then you can write an application that reads a subset of the file, assigns a column value as the PartitionKey, assigns an index as the RowKey, and assigns the other columns in the file as entities in the table. These steps are performed in the sample code below by using the **TableService** and **Entity** objects and the **create_table** and **insert_entity** methods. For other code samples demonstrating how to download (get_blob) and perform other operations, see How to Use the Table Storage Service from Python.

Click the **New Notebook** button in the top right corner of the notebook window, and then copy the following code sample and paste it into the gray box in the browser.
Note: Be sure to replace the account and key with values specific to your storage account.

```
#Next we will demonstrate the table storage management in Microsoft Azure
#we can add top 100 rows of the mako_real_actual_sharks csv to a table storage
#get a handle to your account
table_service = TableService(account_name=account, account_key=key)
table_name = 'makotable';
```

```
csv_file = 'Mako_Real_Actual_Sharks.csv'

#delete the table for temporary data
result = table_service.delete_table(table_name)

# create a new table to save all entities.
result = table_service.create_table(table_name)

#attention: more code need to write if you want to execute all the time at the
same time
#when you call delete_table or create_table, it takes some time for Microsoft
Azure to finish the operation
#you must check the operation status before you can execute the following code!
#then we insert the top 1000 rows into the table, we set PartitionKey to be
each color and RowKey to be the index
index = 0
with open(csv_file) as f:
    reader = csv.DictReader(f) #create a reader which represents rows in a
dictionary form
    for row in reader: #this will read a row as {column1: value1, column2:
value2,...}
        entity = Entity()
        entity.PartitionKey = row['color']
        entity.RowKey= str(index)
        entity.ToppID = row['toppID']
        entity.Time = row['time']
        entity.Longitude = row['Longitude']
        entity.Latitude = row['Latitude']
        table_service.insert_entity(table_name, entity)
        print row
        index=index+1
        if index >= 1000:
            break
```

Reading data from a table

One of the advantages of using table storage is the ability to query data quickly, even though it is not stored in relational tables. By using the **query_entities** method, you can filter your query to retrieve data from your table that match certain criteria:

```
#we can check the azure storage explore to query all inserted entities
#we can also query all table entities with diamonds' color = 'yellow'
makos = table_service.query_entities(table_name, "PartitionKey eq 'yellow'")
for m in makos:
    print(str(m.ToppID),str(m.PartitionKey),str(m.Time),str(m.Longitude),
str(m.Latitude))
```

Other programmatic options for managing storage

You have additional options for developing applications that manage your storage accounts:

- Microsoft Azure Storage Client Library
- Storage Services REST APIs
- LINQ for Microsoft Azure Table service

If you perform a task once or infrequently, you can use the Management Portal or execute a PowerShell or CLI script. For automating frequent or complex tasks, you might prefer to use an

SDK. For example, if you already have experience with Python or Node, you can develop code in that language and bypass the need to learn PowerShell.

Security and Microsoft Azure Storage

Your storage account is secure by default. As the owner of the storage account, only you have access until you grant others permissions to your resources. You have the following options for managing security:

- Grant permissions at container and blob level to allow anonymous access.
- Allow access to a container, blob, table, or queue by using a shared access signature for a limited time
- Shared access signatures can optionally include a stored access policy for even more flexible management

Container and blob permissions

In the Microsoft Azure Management Portal, you specify the access permissions when you create a new container. To do this:

1. Click the storage account on the **Storage** page of the Management Portal.
2. Click the **Containers** link at the top of the page, and then click **Create a Container**, if no containers exist, or click the **Add** button at the bottom of the page.
3. Type a container name in the **NAME** field, and the select one of the following **ACCESS** options, as shown in Figure 8:
    - **Private**: The owner of the storage account is the only user with access to the container.
    - **Public Container**: The storage service responds to anonymous requests to access metadata for the current container and the blobs it contains.
    - **Public Blob**: Anonymous requests can access only the blob data.

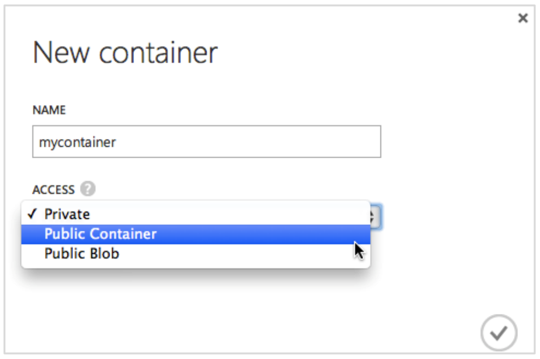


Figure 12: Access permissions selection at container level

For a full list of REST operations and .NET methods and the effect of access setting, see [Restrict Access to Containers and Blobs]().

Shared access signature

To grant permissions on an *ad hoc* basis for a limited time, you can issue a shared access signature to a user or a client application. You can specify a shared access signature for a blob or a container. Either way, in addition to the level of permissions, you can specify the start and end time during which access is granted. Access can include browsing the contents of a container or reading and writing to blob data, table entities, or queue messages. You can see code examples at Create and Use a Shared Access Signature.

Stored access policy

To implement security that is more flexible, create a stored access policy to define permissions and the time range during which access to storage is granted, and reference the policy from a group of shared access signatures. However, you are not obligated to define the stored access policy parameters when you set it up. You can use any combination of parameters when configuring the stored access policy, and then include any remaining parameters directly in the shared access signatures. When you use this approach, you can update the policy when you need to shorten or extend the lifetime of a shared access signature, rather than revoke the signature and issue a new one. You can establish up to five stored access policies per container, table, or queue and use each policy with an unlimited number of shared access signatures. For more information, see Use a Stored Access Policy.

For a closer look at security concerns related to identity and access in the cloud and considerations for building security into your cloud applications, see Microsoft Azure Security Guidance.

Learn more

- Understanding Block Blobs and Page Blobs (http://msdn.microsoft.com/en-us/library/ee691964.aspx)
- Blob Service Concepts (http://msdn.microsoft.com/en-us/library/dd179376.aspx)
- Delivering High-Bandwidth Content with the Microsoft Azure CDN (http://msdn.microsoft.com/en-us/library/ee795176.aspx)
- Understanding the Table Storage Data Model (http://msdn.microsoft.com/en-us/library/dd179338.aspx)
- Table Service Concepts (http://msdn.microsoft.com/en-us/library/dd179463.aspx)
- Querying Tables and Entities (http://msdn.microsoft.com/en-us/library/dd894031.aspx)
- Microsoft Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency (http://sigops.org/sosp/sosp11/current/2011-Cascais/printable/11-calder.pdf)