Contents

[Initial Steps 1](#_Toc364324902)

[Setting up your environment 1](#_Toc364324903)

[Adding certificates to the Windows Azure portal 1](#_Toc364324904)

[Setup the software locally 1](#_Toc364324905)

[Service Management and Windows Azure Storage 1](#_Toc364324906)

[Creating a Windows Store Storage account programmatically 1](#_Toc364324907)

[Manipulating Blobs 2](#_Toc364324908)

[Creating a blob container and uploading your first blob 2](#_Toc364324909)

[Download the blob that was just uploaded 2](#_Toc364324910)

[Get the Uri of the blob that was uploaded 3](#_Toc364324911)

[Get the shared access signature of the blob and download 3](#_Toc364324912)

[Uploading a large blob 4](#_Toc364324913)

[Taking a blob snapshot 4](#_Toc364324914)

[Copying blobs between containers (not supported in Python copying blobs between accounts) 5](#_Toc364324915)

[Deleting containers and blobs 5](#_Toc364324916)

[Manipulating Tables 5](#_Toc364324917)

[Tidying up all storage accounts 5](#_Toc364324918)

[Deleting storage accounts 5](#_Toc364324919)

# Introduction

The purpose of this demo script is to show the attendee a good breadth of functionality of Windows Azure Storage. The accompanying python files storagetools.py and the associated package storagetoolspackage need to be copied to the demo laptop. This can be obtained from the Day 1 – Windows Azure Storage MSR course module.

The accompanying slides should be used in conjunction to talk through the theory of each of the operations performed in the demo and the single line of code in python should be replicated on the slides as a talking point.

# Initial Steps

## Setting up your environment

In order to use the demo it’s necessary to be able to authenticate to windows azure. To enable this ensure that you have OpenSSL installed on the machine you’re running the demo from.

* *openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout msrcert.pem -out msrcert.pem*

The above command will generate a 1024bit RSA public/private key with an X509 certificate called msrcert.pem. We can use this in all of our subsequent demos using the Service Management API to Windows Azure.

The following will generate an X509 certificate called msrcert.cer in the current directory.

* *openssl x509 -inform pem -in msrcert.pem -outform der -out msrcert.cer*

## Adding certificates to the Windows Azure portal

Login to Windows Azure with a valid account and upload the msrcert.cer to the **ManagementCertificates** section.

***[Insert image of management certificates in the Azure Portal]***

To proceed with the demo the Python SDK for Windows Azure should be installed from here:

<http://www.windowsazure.com/en-us/downloads/?sdk=python>

## Setup the software locally

Once the certificate has been uploaded to the portal the code for “module 3 – azure storage” should be downloaded so that the demo script can be run as per this document.

Copy the .pem file that was created earlier into the directory with *storagetools.py*. In the subdirectory storagetoolspackage edit credentials.py and replace the subscription id and certificate path your Windows Azure subscription id and the path to your .pem certificate like so:

* subscriptionid = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
* certificate = '/Users/research/Documents/msr.pem'

# Service Management and Windows Azure Storage

## Creating a Windows Store Storage account programmatically

To begin create a storage like so:

* storagetools.py --storage --container msrwmtest --create

Navigate to the portal as per the following image and show the users the storage account created in the portal with the associated label and description and the location.

***[Insert image of storage account]***

Run the above command again to show that storage names are unique. The following response message should show up on screen:

**Storage account msrwmtest exists already**

# Manipulating Blobs

## Creating a blob container and uploading your first blob

Issue the following command to the storagetools.py script:

* python storagetools.py --blobs msrwmtest --container msrwmtest --create --upload myblob.txt

This should give a response something to this:

**Getting storage keys for account msrwmtest**

**primary key is u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**secondary key is CraIFCC61NgR+WuZydXY4wt7NLXY6G3+78zdItrcQ8Gt0S4rE3x4g5w9mU7jN9QyXly+pIjXg9su9njkkql0eg==**

**Creating local file with content - some text!!**

**Creating storage account container msrwmtest**

**Setting container ACL to private for msrwmtest**

**Creating block blob with name myblob.txt in container msrwmtest**

Keys will be returned for the named storage account msrwmtest and the primary key will be used to access the storage account. The container will be created called msrwmtest and a text file created locally (on the fly) and then uploaded to the container.

Check the container to make sure that the file myblob.txt exists in the container.

***[Show image of Azure Management Studio with the container or the Windows Azure portal]***

## Download the blob that was just uploaded

In order to download the blob enter the following command:

* python storagetools.py --blobs msrwmtest --container msrwmtest --download myblob.txt

This should put a file in the current directory called “download.txt”. The keys will of course vary.

**Getting storage keys for account msrwmtest**

**primary key is u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**secondary key is CraIFCC61NgR+WuZydXY4wt7NLXY6G3+78zdItrcQ8Gt0S4rE3x4g5w9mU7jN9QyXly+pIjXg9su9njkkql0eg==**

**Found myblob.txt in container msrwmtest**

**Written blob to local file download.txt**

***[Show image of the local filesystem]***

## Get the Uri of the blob that was uploaded

Now that we’ve showed an upload and a download of the file we can get a public uri to show that the file cannot be downloaded directly despite the HTTP endpoint, without the key.

To do this, the following should be entered:

* python storagetools.py --blobs msrwmtest --container msrwmtest --public myblob.txt

**Getting storage keys for account msrwmtest**

**primary key is u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**secondary key is CraIFCC61NgR+WuZydXY4wt7NLXY6G3+78zdItrcQ8Gt0S4rE3x4g5w9mU7jN9QyXly+pIjXg9su9njkkql0eg==**

[**https://msrwmtest.blob.core.windows.net/msrwmtest/myblob.txt**](https://msrwmtest.blob.core.windows.net/msrwmtest/myblob.txt)

The last line is the HTTPS uri of the blob. Open up a browser window and enter this Uri into the addresses bar.

You should see something like the following since the ACL has been set with private.

<Error>

<Code>ResourceNotFound</Code>

<Message>

The specified resource does not exist. RequestId:3e15b1b7-e9ed-4a24-aa04-b4593c84a0c4 Time:2013-08-13T05:52:42.4152283Z

</Message>

</Error>

## Get the shared access signature of the blob and download

Enter the following command to generate the shared access signature:

* python storagetools.py --blobs msrwmtest --container msrwmtest --signature myblob.txt

**storagetools.py --blobs msrwmtest --container msrwmtest --signature myblob.txt**

**Getting storage keys for account msrwmtest**

**primary key is u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**secondary key is CraIFCC61NgR+WuZydXY4wt7NLXY6G3+78zdItrcQ8Gt0S4rE3x4g5w9mU7jN9QyXly+pIjXg9su9njkkql0eg==**

**[<azure.storage.sharedaccesssignature.Permission instance at 0x105e7edd0>]**

**u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**https://msrwmtest.blob.core.windows.net/msrwmtest/myblob.txt**

**https://msrwmtest.blob.core.windows.net/msrwmtest/myblob.txt?st=2013-08-13T05:57:17Z&se=2013-08-13T06:57:17Z&sp=r&sr=b&sig=lVgh%2BwPf20wQAjC2cj%2Bmz0xHsO1m0hD27V/uKS5GhJs%3D&**

As you can see the last line is a different Uri which contains a signed request to the storage service for the myblob.txt file. This request is only valid for an hour as per the noticeable time difference. Cut and paste this into the browser address bar and see the resource downloaded.

## Uploading a large blob

In order to upload a large blob of 64MB or more use the following:

* python storagetools.py --blobs msrwmtest --container msrwmtest --large /Users/azurecoder/Pictures/haunted-house.jpg

The following output should be visible describing the blob upload.

**Getting storage keys for account msrwmtest**

**primary key is u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**secondary key is CraIFCC61NgR+WuZydXY4wt7NLXY6G3+78zdItrcQ8Gt0S4rE3x4g5w9mU7jN9QyXly+pIjXg9su9njkkql0eg==**

**Added empty blob called AzureHpc.zip**

**Total file size: 2423K**

**adding 2 MB to storage**

**Total file size: 0K**

**completed storage upload**

## Taking a blob snapshot

In order to take a snapshot of a blob use the following:

* python storagetools.py --blobs msrwmtest --container msrwmtest --snapshot myblob.txt

This will snapshot the blob and list the blobs in the account container including the snapshot.

**Getting storage keys for account msrwmtest**

**primary key is u7fPme7hBSYxY5chfIbi/aT2GjlalPek+Y/uzj1e3ctv83zKe7VhNpBc7WEF8KYIkKX2n1sddRE2ll09N5pUwg==**

**secondary key is CraIFCC61NgR+WuZydXY4wt7NLXY6G3+78zdItrcQ8Gt0S4rE3x4g5w9mU7jN9QyXly+pIjXg9su9njkkql0eg==**

**Taken snapshot of blob myblob.txt**

**Listing all blobs in container msrwmtest**

**myblob.txt**

**http://msrwmtest.blob.core.windows.net/msrwmtest/myblob.txt?snapshot=2013-08-13T13%3a42%3a05.1702593Z**

**myblob.txt**

**http://msrwmtest.blob.core.windows.net/msrwmtest/myblob.txt**

Note the snapshot Uri with the date and time added to it.

## Copying blobs between containers (not supported in Python copying blobs between accounts)

Create the new container without the uploading a blob to the account:

* python storagetools.py --blobs msrwmtest --container msrwmtest2 --create

When this is complete issue the following command to copy the blob between accounts.

* python storagetools.py --blobs msrwmtest --container msrwmtest2 --copyaddress /msrwmtest/msrwmtest/myblob.txt

Check the Windows Azure Management Portal to ensure that the blob has been copied to the new container.

## Deleting containers and blobs

Use this command to delete the blobs in the container and the container itself:

* python storagetools.py --blobs msrwmtest --container msrwmtest –delete

Show the portal so that they can see that the blob and container has been deleted.

If you have made a snapshot then this should fail unless the snapshot blob has been deleted first. Talk through that snapshots cannot exist without the original.

**Traceback (most recent call last):**

**File "storagetools.py", line 180, in <module>**

**main(sys.argv[1:])**

**File "storagetools.py", line 107, in main**

**blobManager.TidyUp()**

**File "/Users/azurecoder/StorageTools/storagetoolspackage/Blobs.py", line 114, in TidyUp**

**self.blob\_service.delete\_blob(self.container\_name, blob.name)**

**File "/Library/Python/2.7/site-packages/azure/storage/blobservice.py", line 713, in delete\_blob**

**response = self.\_perform\_request(request)**

**File "/Library/Python/2.7/site-packages/azure/storage/storageclient.py", line 140, in \_perform\_request**

**\_storage\_error\_handler(e)**

**File "/Library/Python/2.7/site-packages/azure/storage/\_\_init\_\_.py", line 766, in \_storage\_error\_handler**

**return \_general\_error\_handler(http\_error)**

**File "/Library/Python/2.7/site-packages/azure/\_\_init\_\_.py", line 644, in \_general\_error\_handler**

**raise WindowsAzureConflictError(\_ERROR\_CONFLICT)**

**azure.WindowsAzureConflictError**

Delete the snapshot through Azure Management Studio or zud.io and then run the command again.

# Manipulating Tables

## Creating a table

When the storage account has been created issue the following command to create the table.

* python storagetools.py --tables msrwmtest --tablename secureaudit --create

In order to check the table exists use Azure Management Studio and navigate to the tables section to see the secureaudit table.

## Adding the first test entity

This is the first of two test entities that will be added to the table.

* python storagetools.py --tables msrwmtest --tablename secureaudit --entity 1 --create

You can use Azure Management Studio and navigate to that table and the see the new record.

**Run this command again and show the error to illustrate the rowkeys and partitionkeys should be unique.**

## Adding the second test entity

This is the first of two test entities that will be added to the table.

* python storagetools.py --tables msrwmtest --tablename secureaudit --entity 2 --create

You can use Azure Management Studio and navigate to that table and the see the second record. This has an extra column in it which should show the dynamic nature of the table storage schema.

## Update the second test entity

This will update entity 2.

* python storagetools.py --tables msrwmtest --tablename secureaudit --entity 2

Check the second row in the table and ensure that the attempts column has been incremented by 1 to 2.

## Showing entities in the table

List all of the entities in a table.

* python storagetools.py --tables msrwmtest --tablename secureaudit --list

Check to see that the values correlate with the input values in code

## Deleting a table

To delete a table use the following:

* python storagetools --tables msrwmtest --tablename secureaudit --delete

Check in Azure Management Studio to ensure that the table has disappeared.

# Tidying up all storage accounts

## Deleting storage accounts

In order to delete a storage account use the following:

* python storagetools.py --storage –container msrwmtest --delete

Show the portal deleting the account whilst this is occurring.