Manage Windows Azure Storage

Overview

Windows Azure Storage is designed for cost-effectively storing and retrieving large volumes of data while providing ease of access and durability. It offers non-relational data storage including Blob, Table, Queue and Drive storage. In this lab, you will learn to use different tools to manage Windows Azure Storage Service.

Objectives

In this hands-on lab, you will learn how to:

- Use Azure Storage Explorer to manage your storage accounts.
- Use IPython notebook to run storage commands.
- Use AzCopy to Copy files between different storage accounts.(optional)

Prerequisites

The following is required to complete this hands-on lab:

- A Windows Azure subscription - sign up for a free trial
- You **must** use one of the following **browsers**: Latest version of **Firefox or Chrome, IE 9, 10, 11.** Browsers like Safari, 360 may have issues with IPython or RDP download.

Exercises

This hands-on lab includes the following exercises:

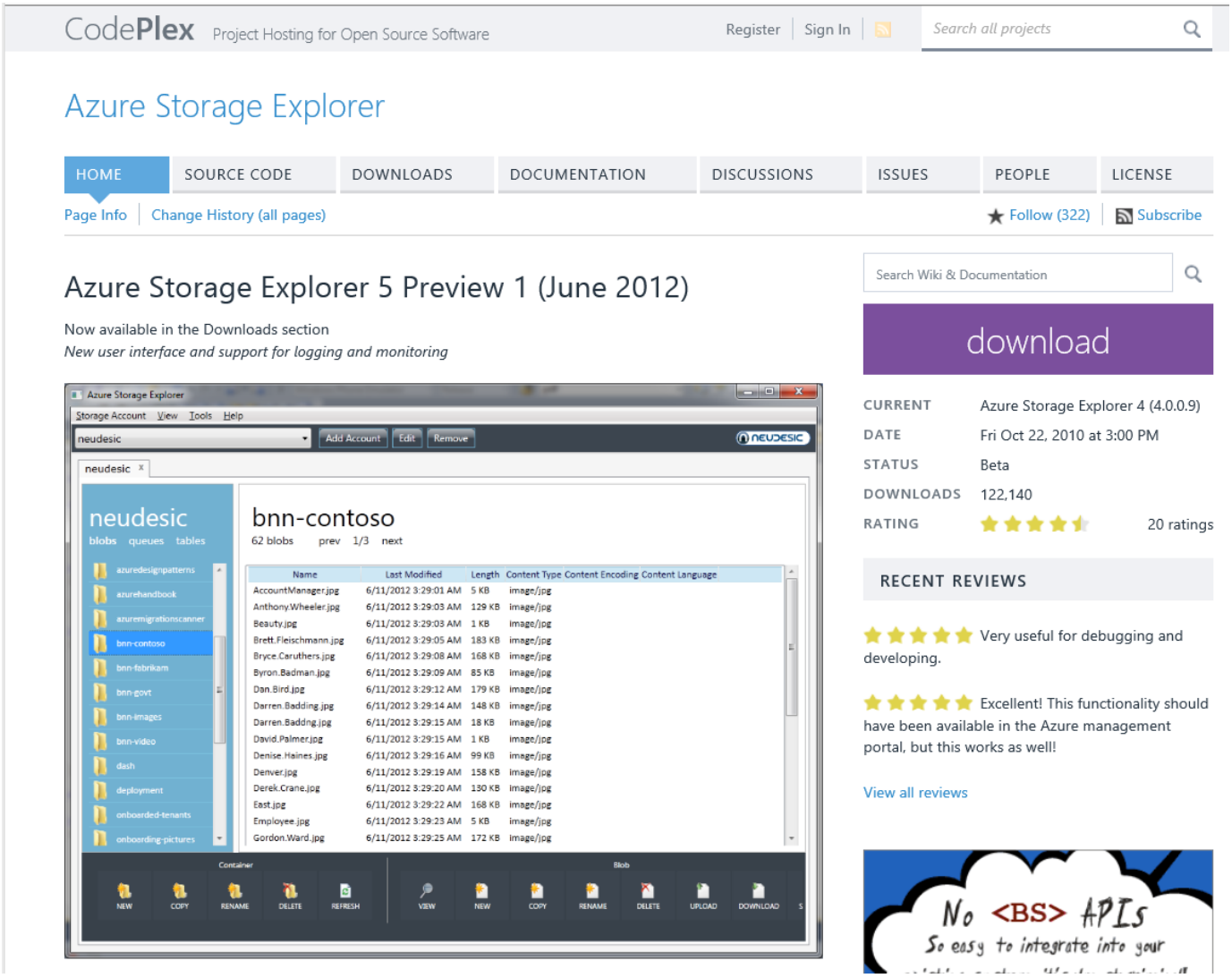
1. Use Azure Storage Explorer to manage your storage accounts.
2. Use IPython notebook to run storage commands.
3. Use AzCopy to Copy files between different storage accounts.

Estimated time to complete this lab: **60** minutes.

Excercise 1: Use Azure Storage Explorer to manage your storage accounts.

Azure Storage Explorer is a useful GUI tool for inspecting and altering the data in your Windows Azure Storage storage projects including the logs of your cloud-hosted applications. All 3 types of cloud storage can be viewed and edited: blobs, queues, and tables.

1. Azure Storage Explorer can be downloaded for free from CodePlex.

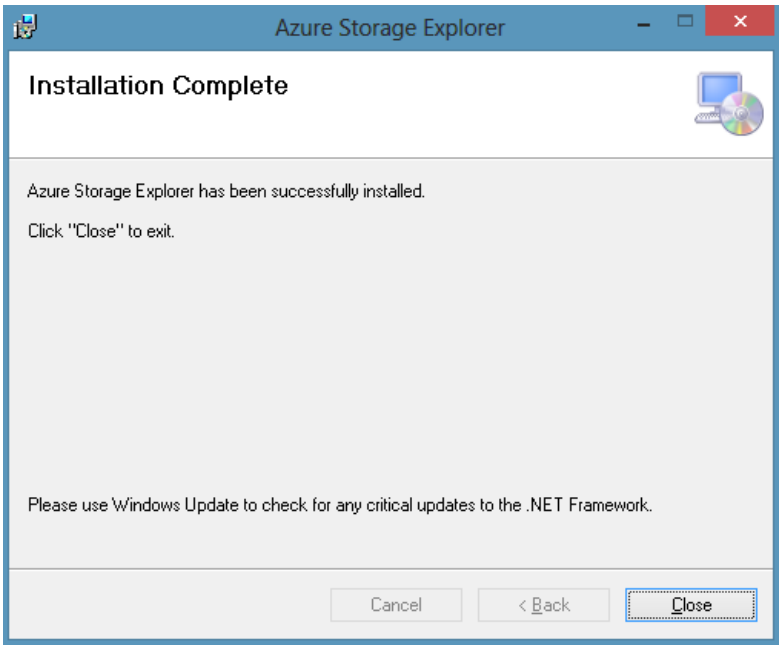


Azure Storage Explorer Page

The latest version is 5.0 Preview, but you can also download the previous released version.

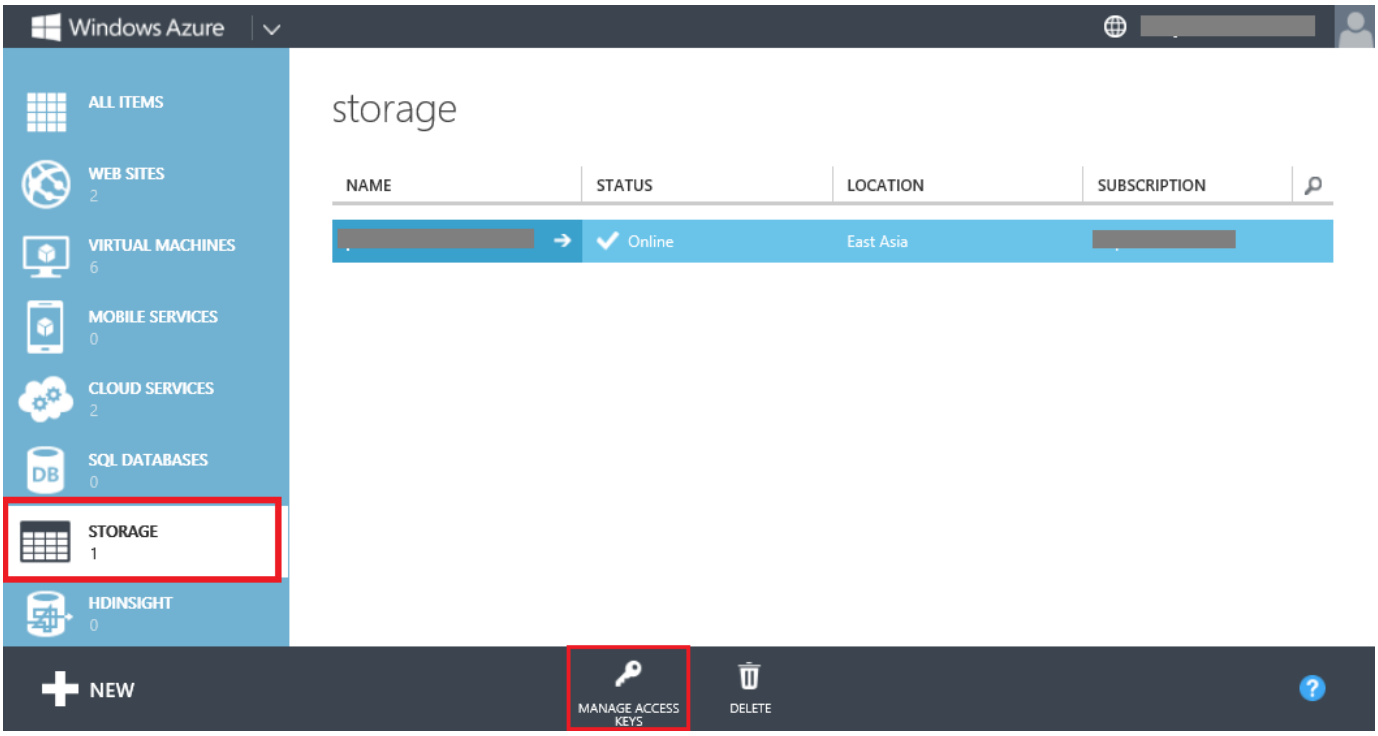
2. Download *Azure Storage Explorer 4* or better and install it.

> The Azure Storage Explorer can only be installed on Windows machines.



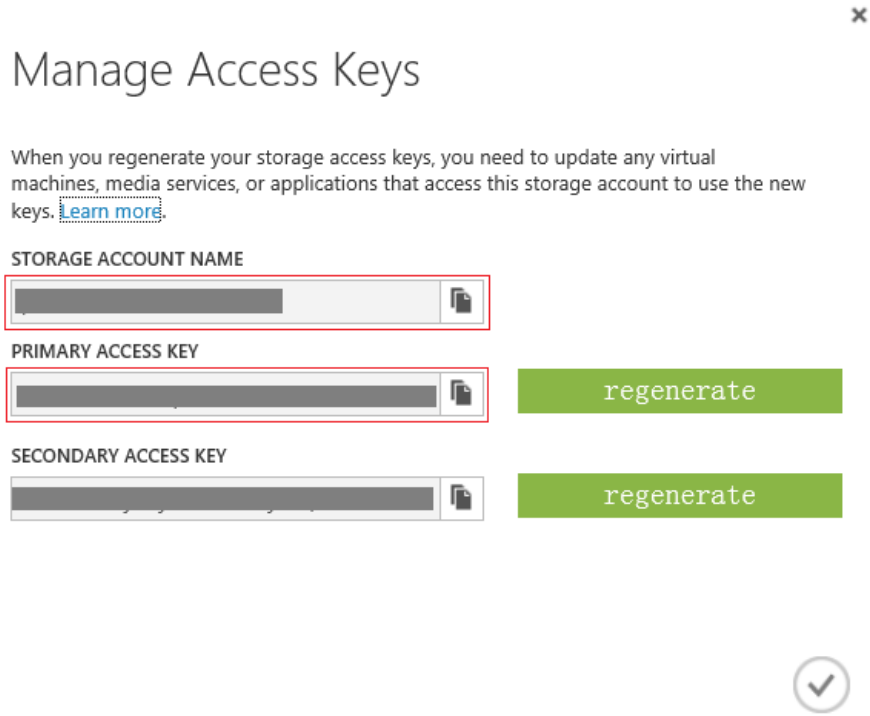
Install Azure Storage Explorer

3. Before you launch the tool, you need to know your storage account's name add access key. You can find your storage account name and acess key in your Windows Azure Management Portal.

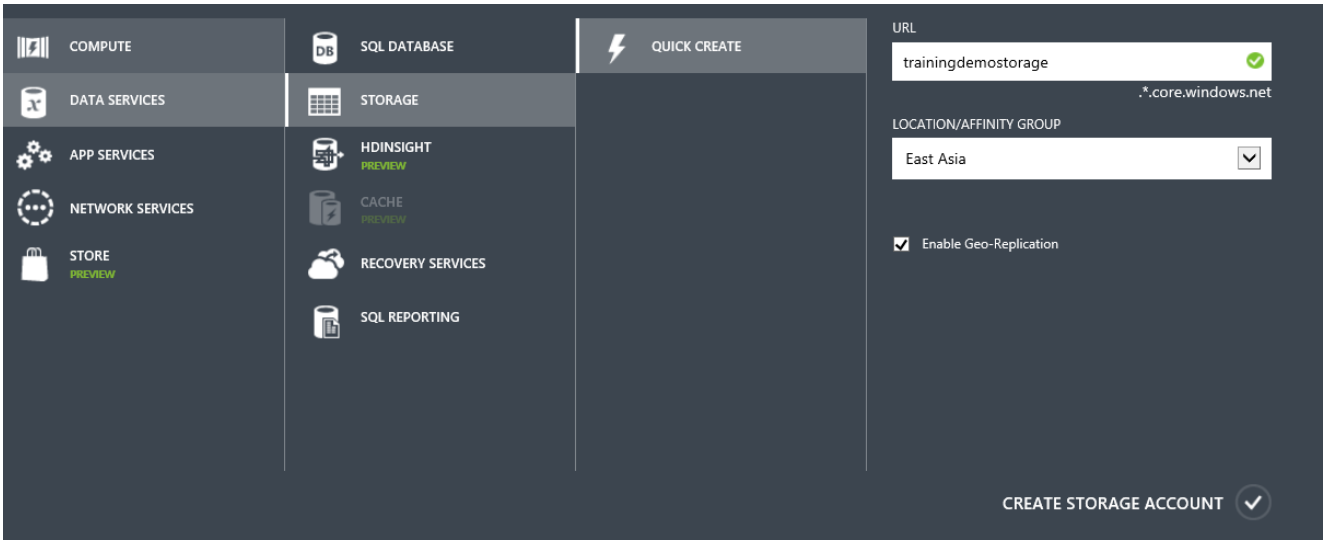


Windows Azure Storage Account

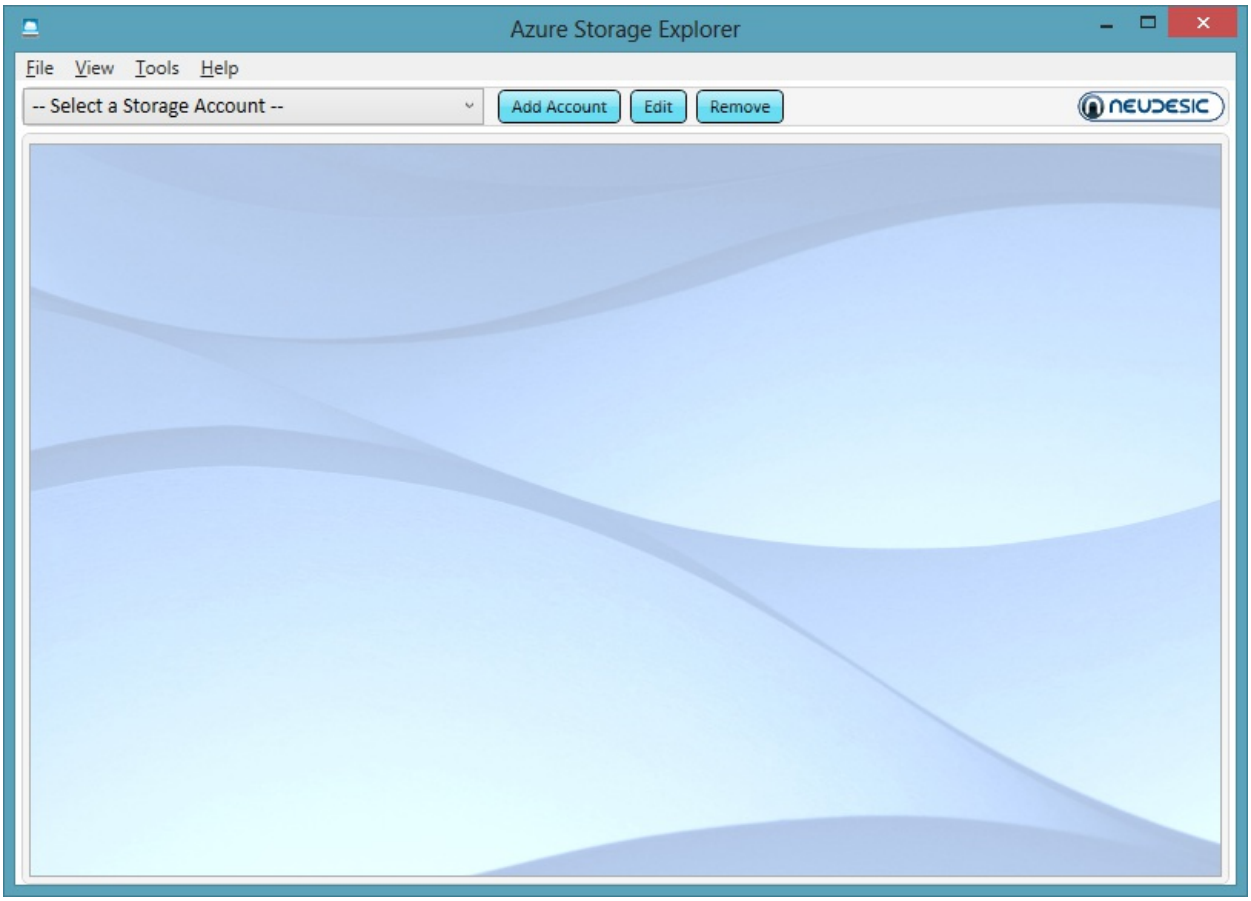
Click the "Manage Access Keys" button under the page to display the storage account name and access keys for the currently selected storage account.



Manage Access Key

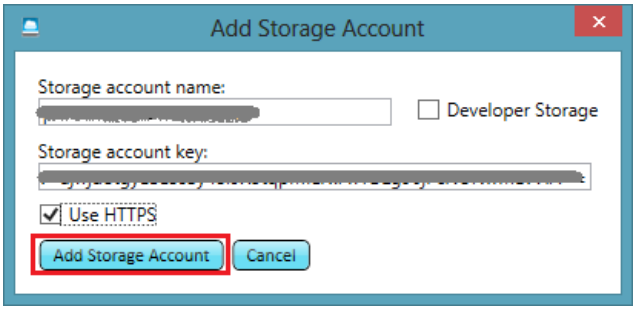
> If there is no storage account under your subscription, you can just click **New** -> **Data Service** -> **Storage** -> **Quick Create** to create one.
>
> 
>
>
> *Create A Storage Account*

4. Launch *Azure Storage Explorer*.



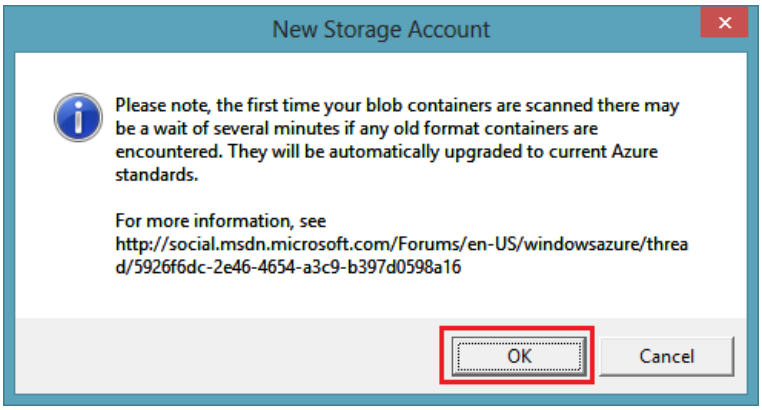
Azure Storage Expolor

5. Then Click "Add Account" button on the top and input your account name and access key in the form, check the **Use HTTPS** and click "Add Storage Account".

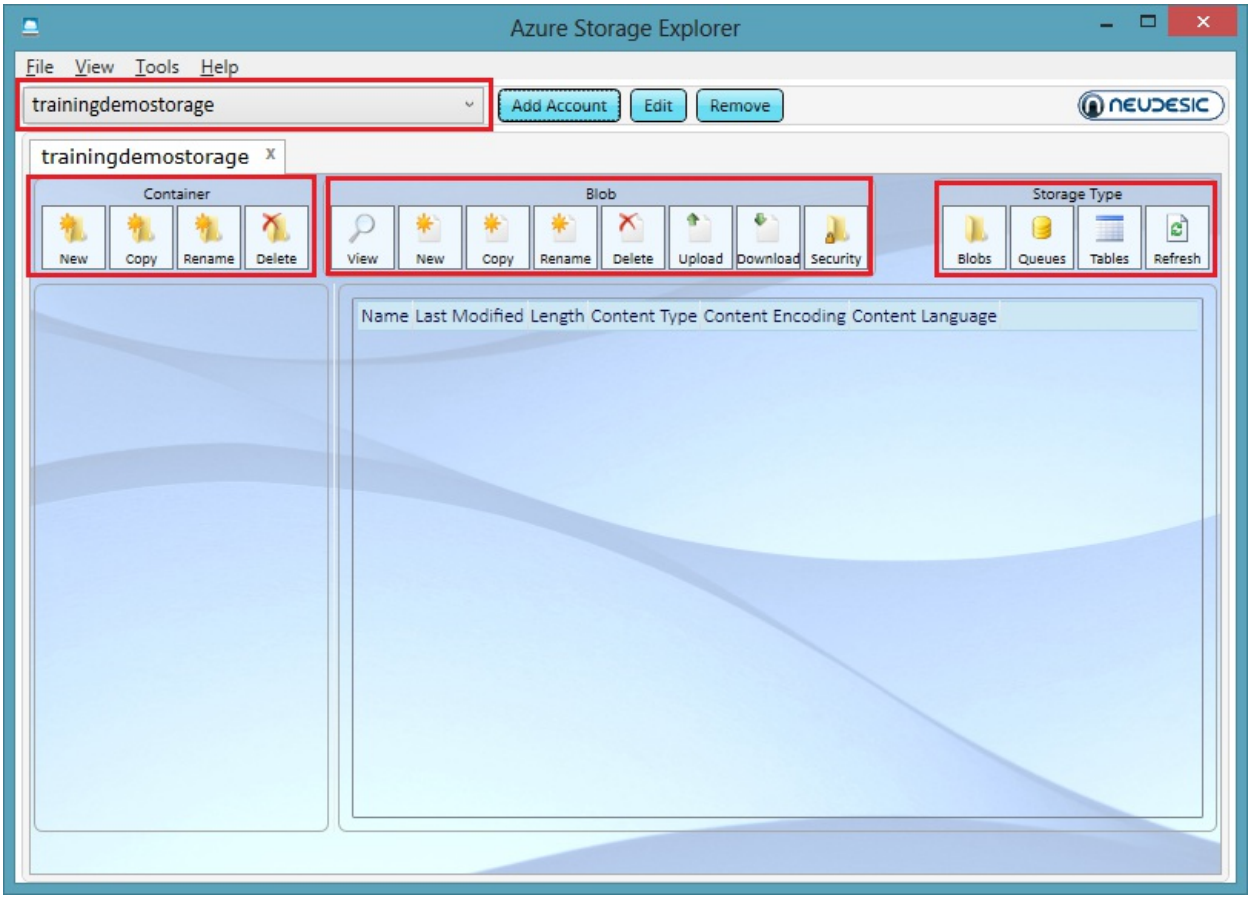


Add Account Information

6. A new form will pop up to notify you it takes some time to scan the storage account if it is the first time you add the storage account. Click OK and after several seconds Azure Storage Explorer will be ready to help you viewing the blob, table and queue data in your storage account.



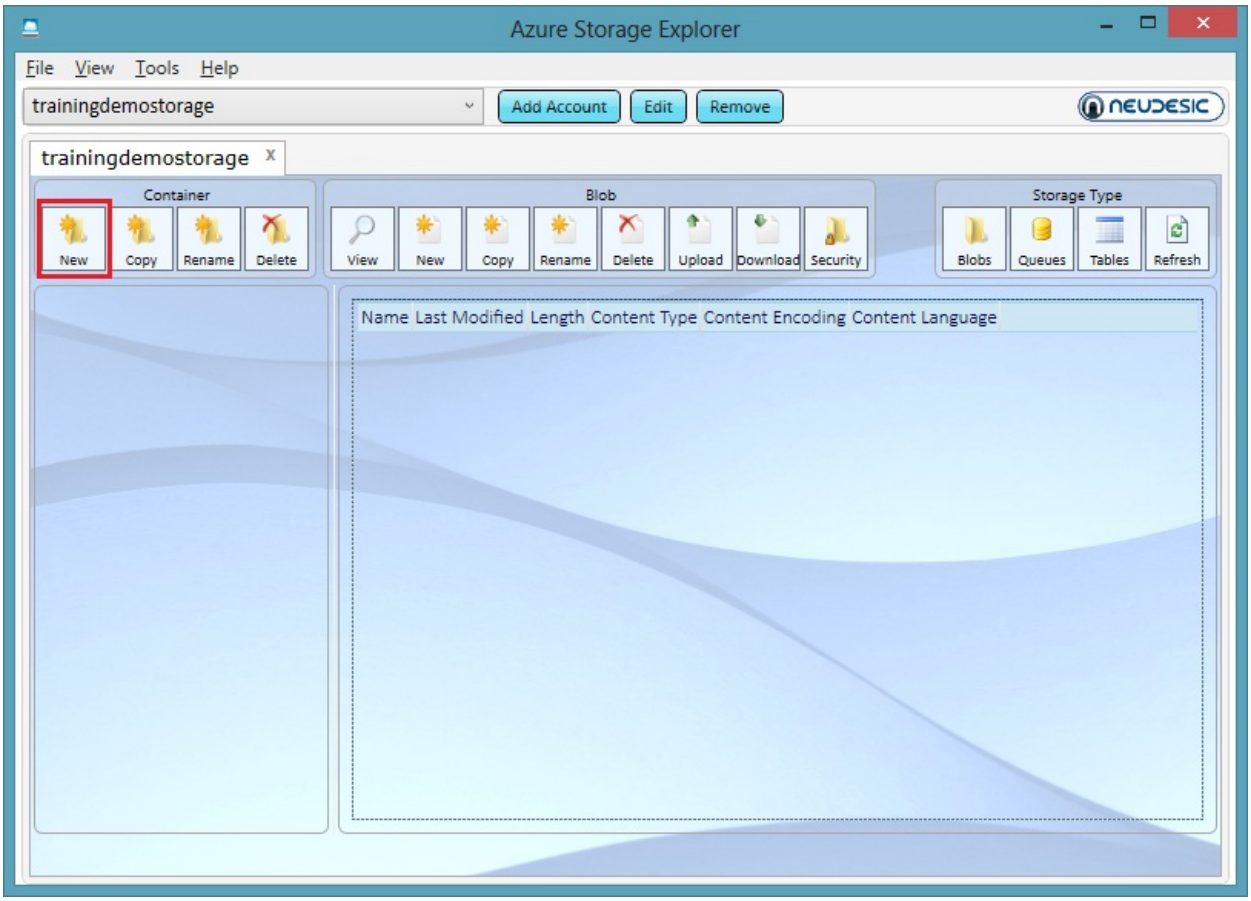
New Storage Account Confirmation



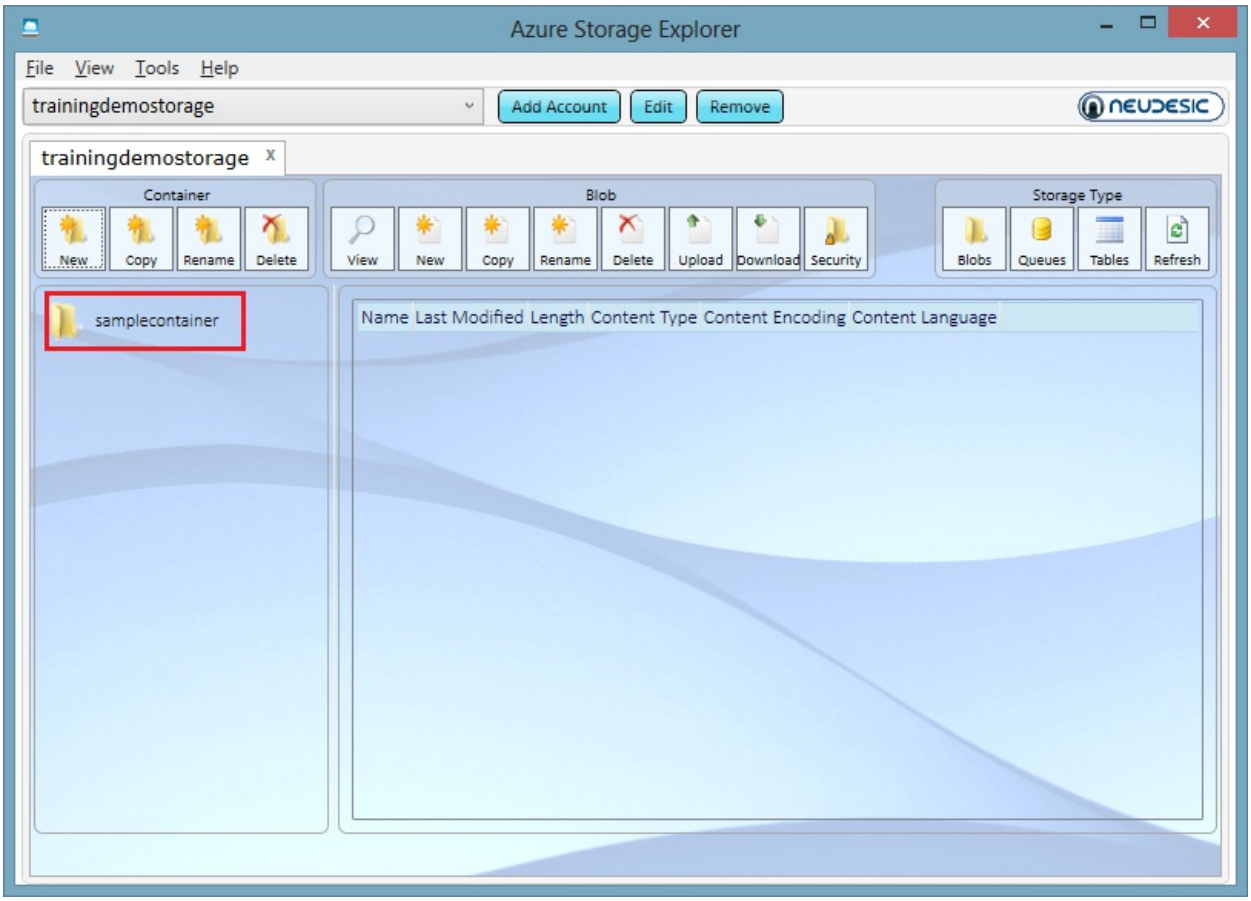
Manage Storage Account

Azure Storage Explorer displays information about Blob storage by default. From here, you can create, copy, rename or delete a container with the button on left top. Now let's create a sample container and upload a file to your storage account.

7. Click *New*, input the container name *samplecontainer* and set the *Accessibility* to be *Public Container*. Click *Create Container* and you will create a new container under the storage account. The "Public Container" means that everyone can access the file through its fully qualified URL (via http or https). If you don't want anyone knowing your URLs to be able to access files in the container, set the accessibility to be private instead of public.

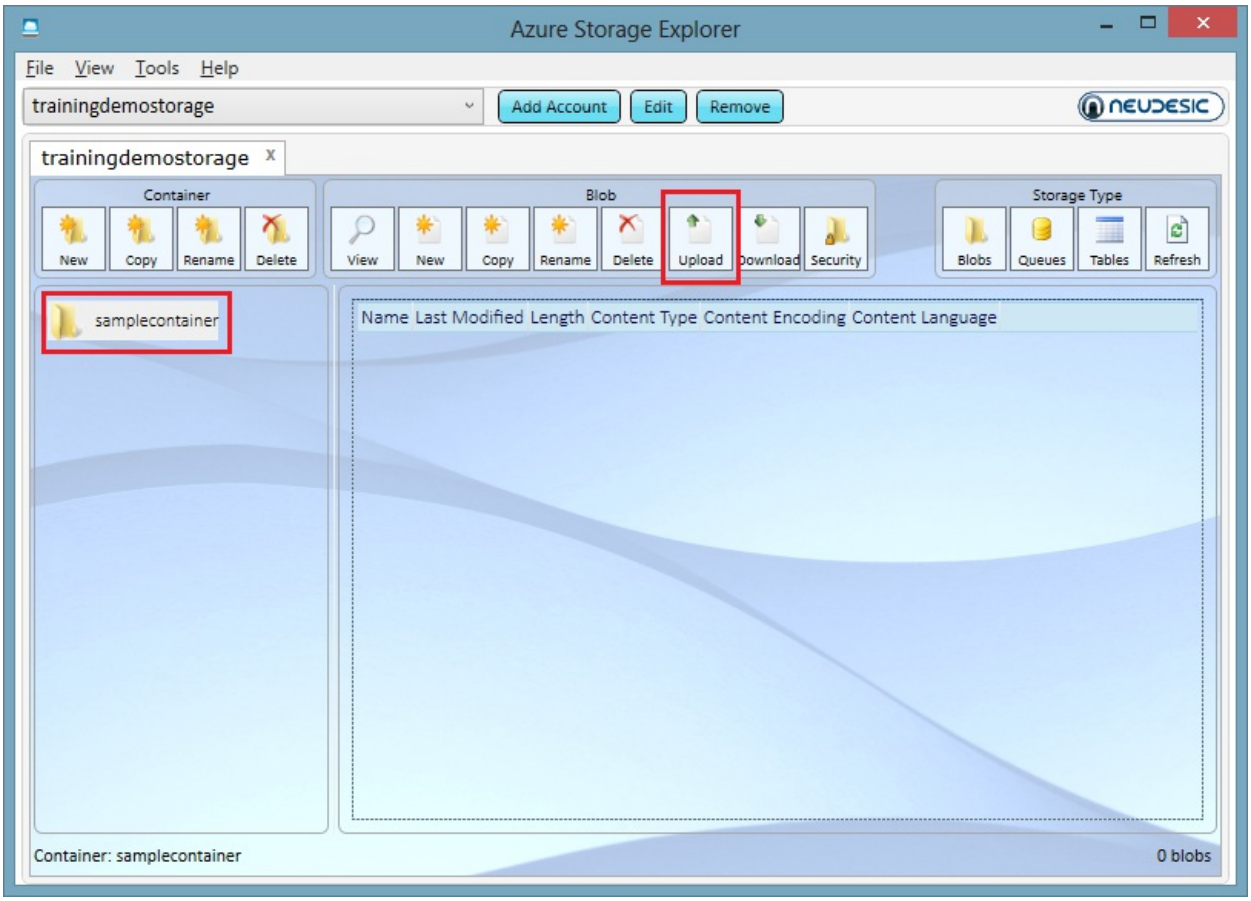

Azure Storage Explorer
File View Tools Help
trainingdemostorage
Add Account
Edit
Remove
NEUDESIC
trainingdemostorage
Container
New
Copy
Rename
Delete
Blob
View
New
Copy
Rename
Delete
Upload
Download
Security
Storage Type
Blobs
Queues
Tables
Refresh
Name Last Modified Length Content Type Content Encoding Content Language


New Container
Container name:
samplecontainer
Accessibility:
Public Container
Public Blob
Private
Create Container
Cancel



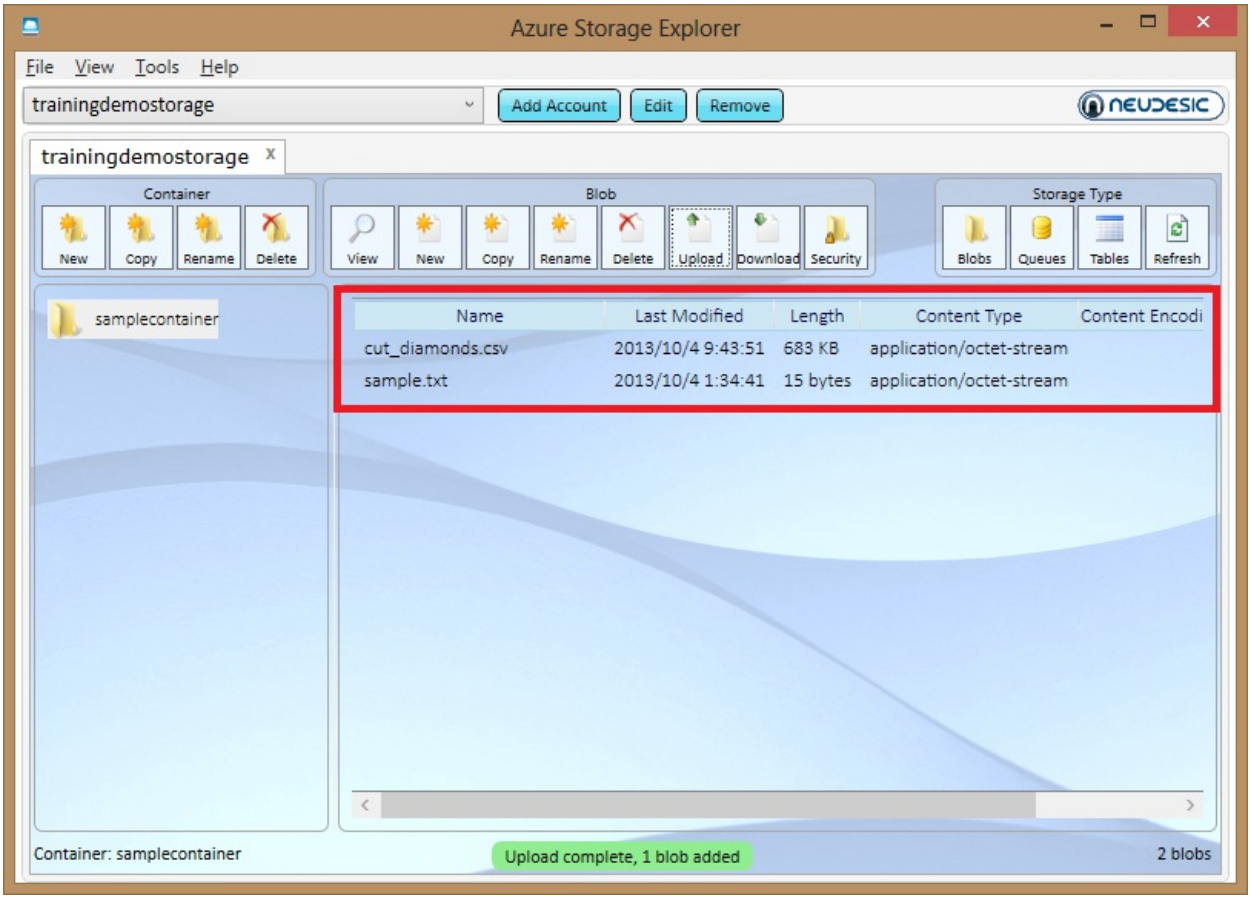
New Container

8. Next, we will upload a new file to the new container. Click the new container on the left and click *Upload* button on the top.



Upload File

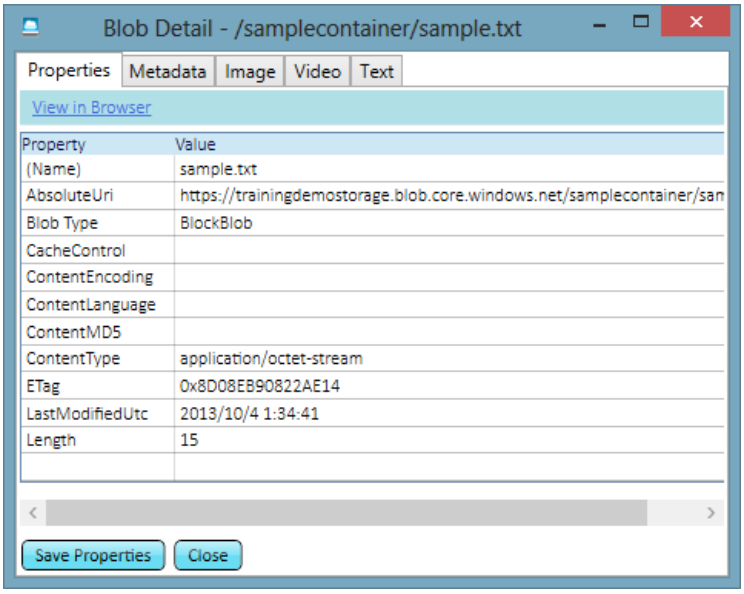
9. You can locate any files on your drives. We have some sample files under **Source\Exercise1** which you can upload. Please make sure you upload the file **cut_diamonds.csv** which will be used Exercise 2



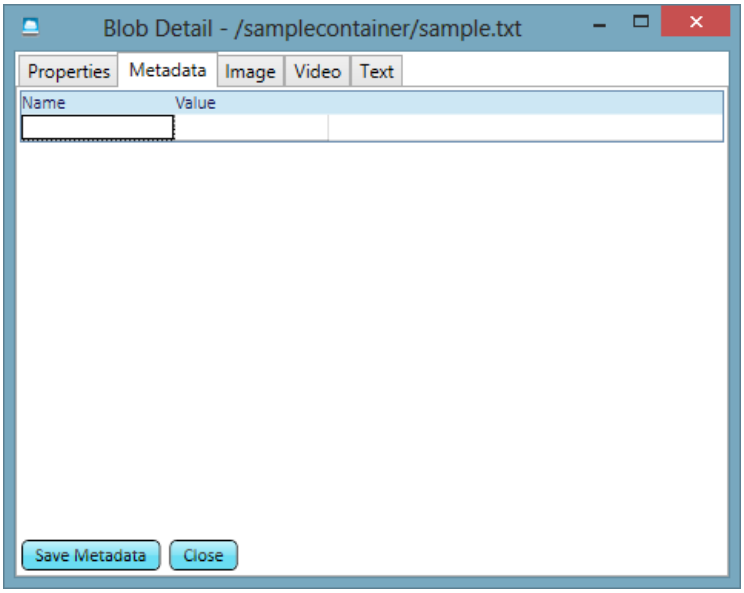
Uploaded Files

Files stored in the Blob Storage Service can simply referred to as blobs. You can see basic information about the blob such as its name, when last modified, length and content type. Each storage account in Windows Azure can hold up to 200TB which could consist of many large blobs, or even one 200GB blob.

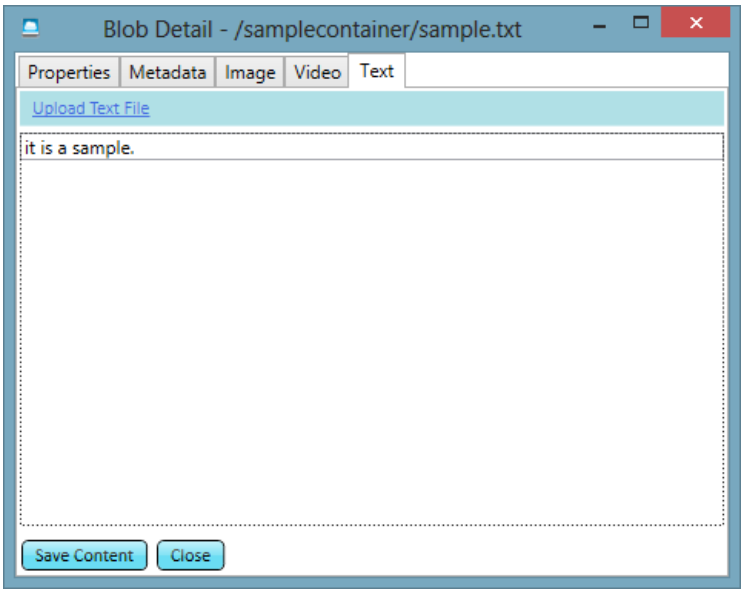
10. Double click the file **Sample.txt** and you will find more information. You can find the properties and metadata. Since the file is a text file, Azure Storage Explorer displays its contents.



File Property

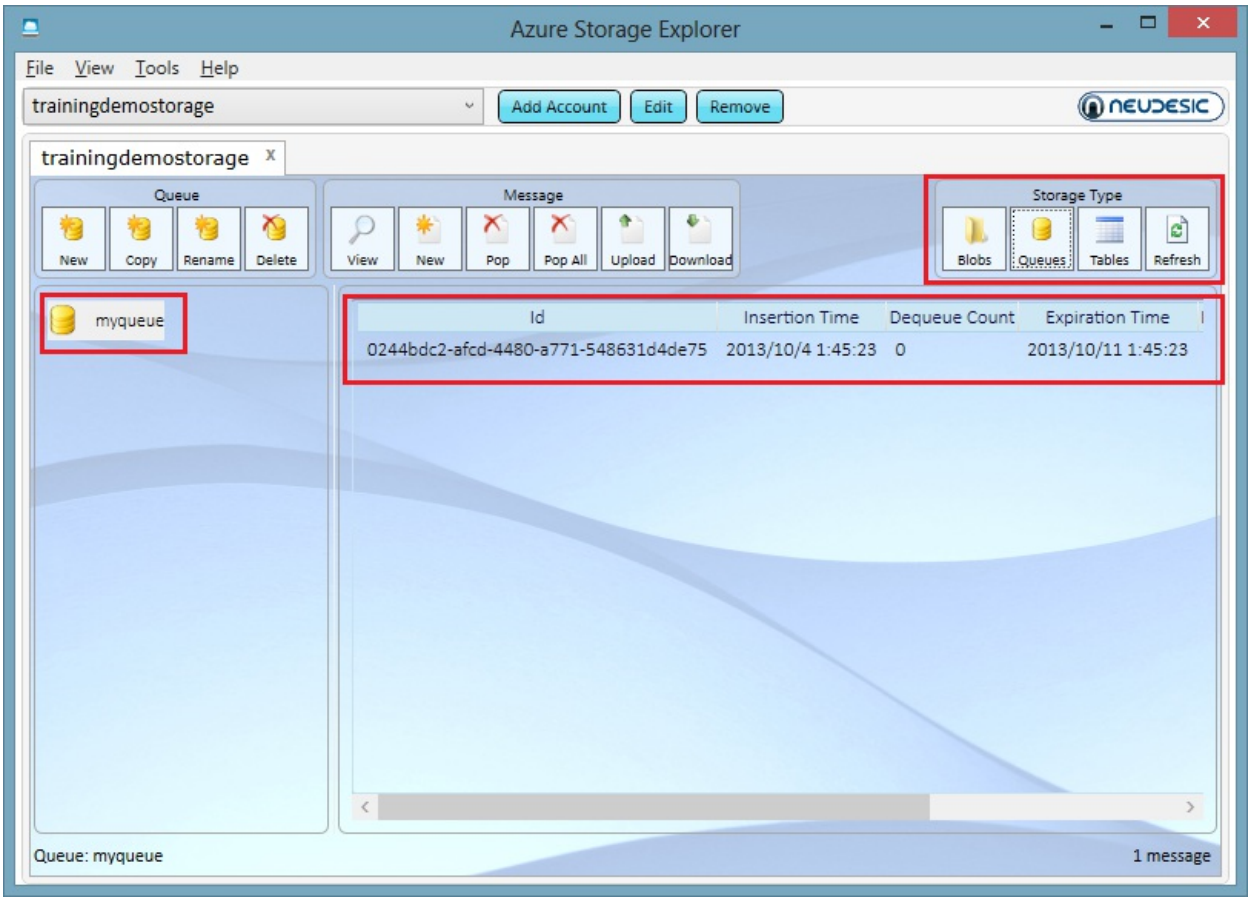


File Metadata

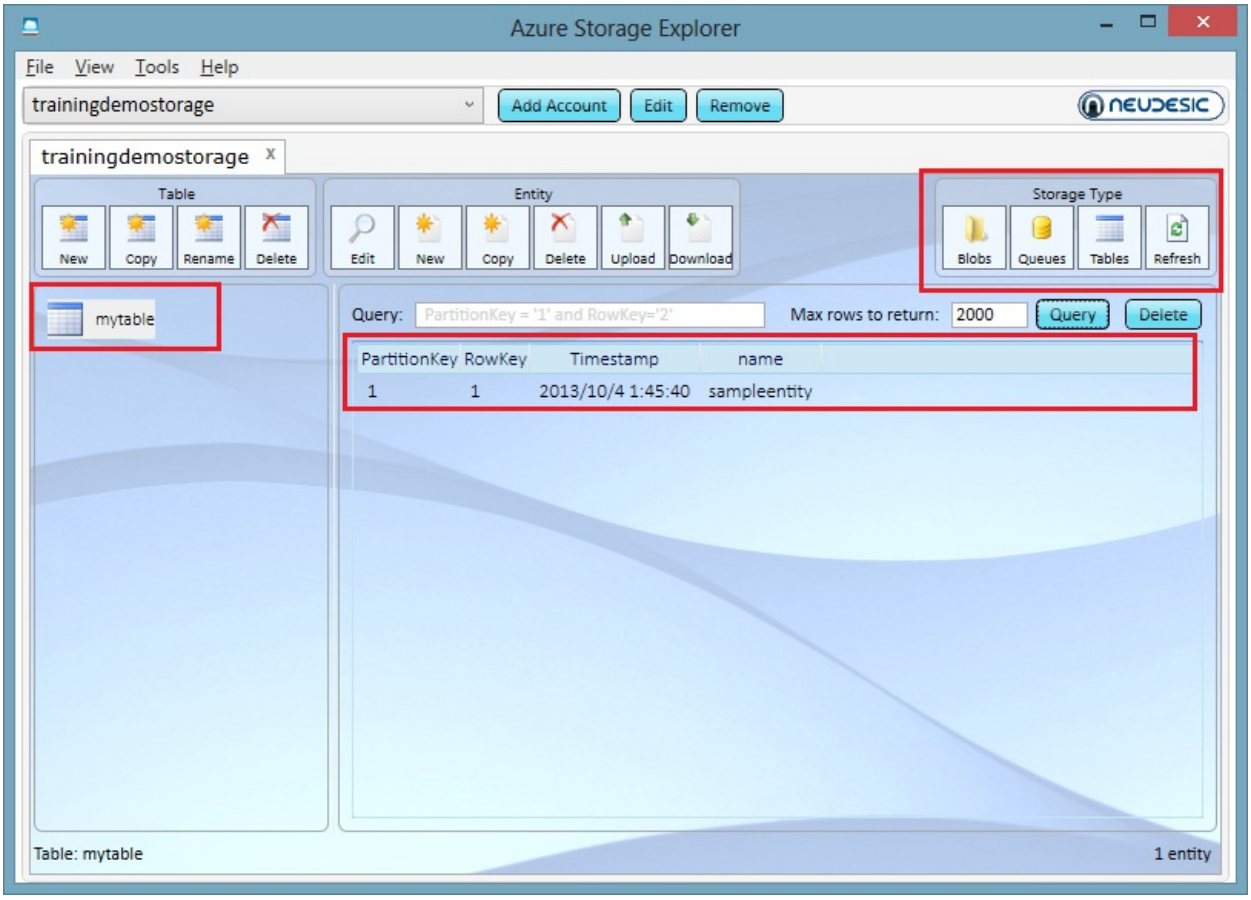


The Content of a File

11. On the top right side, you can also manage the Windows Azure Table and Queue data in your storage account. With Tables, you can add or remove entities along with other Table management features. With Queues, you can push and pop messages along with other Queue management features. For more instructions on interacting with Queue and Table data using Azure Storage Explorer, you can visit AzureStorageExplorer4UserGuide.pdf.



Azure Storage Explorer Queue Management



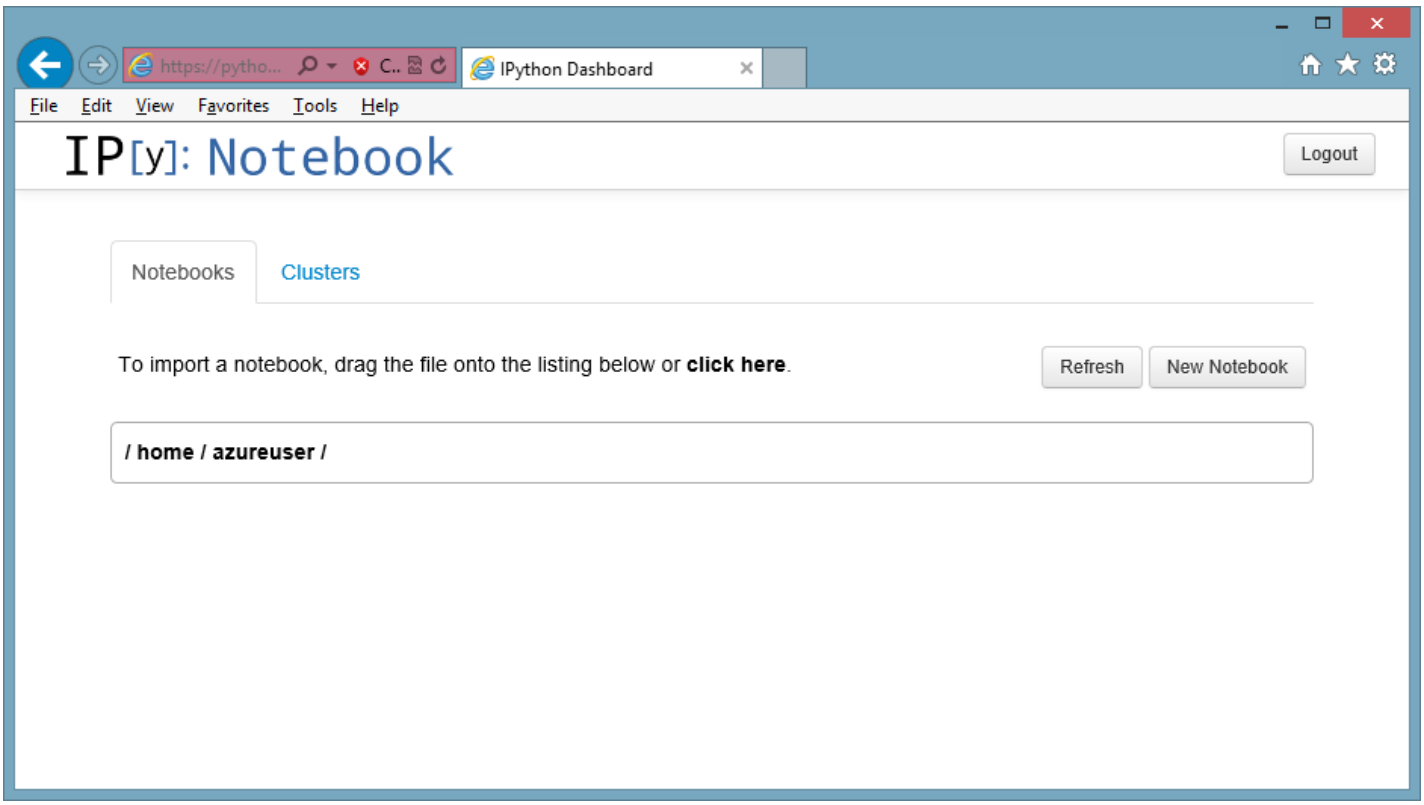
Azure Storage Explorer Table Management

Excercise 2: Use IPython notebook to run storage commands.

Use the IPython notebook you have already created in the previous exercise **Using Windows Azure Virtual Machines.** for this exercise. You can manage Windows Azure Storage Account in IPython.

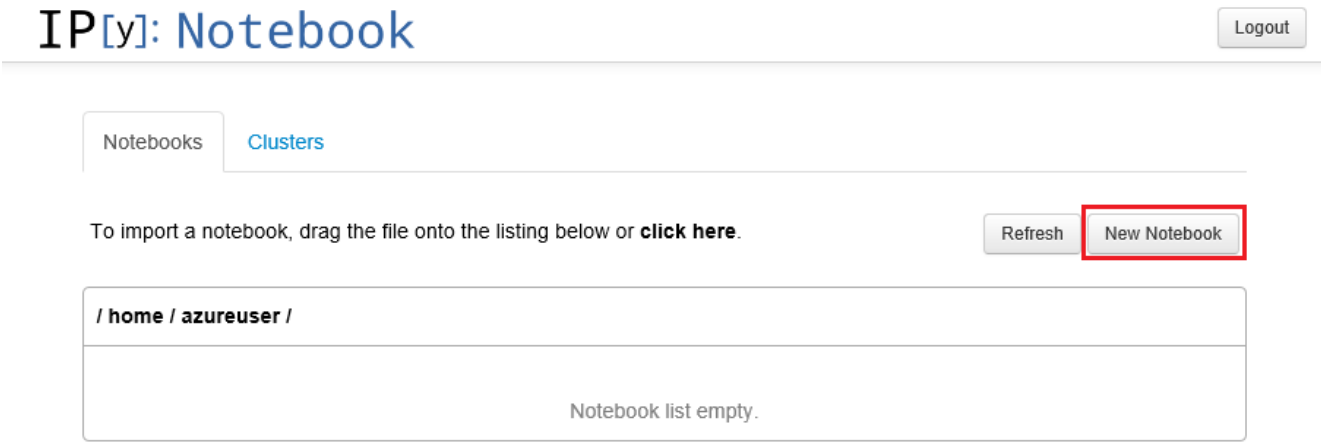
> Note: If you have not completed the Virtual Machines lab, please note that IPython notebook is an interactive Python framework which makes Python project development and management much more easier.
>
> Build an IPython environment on windows azure, you can read http://www.windowsazure.com/en-us/develop/python/tutorials/ipython-notebook/

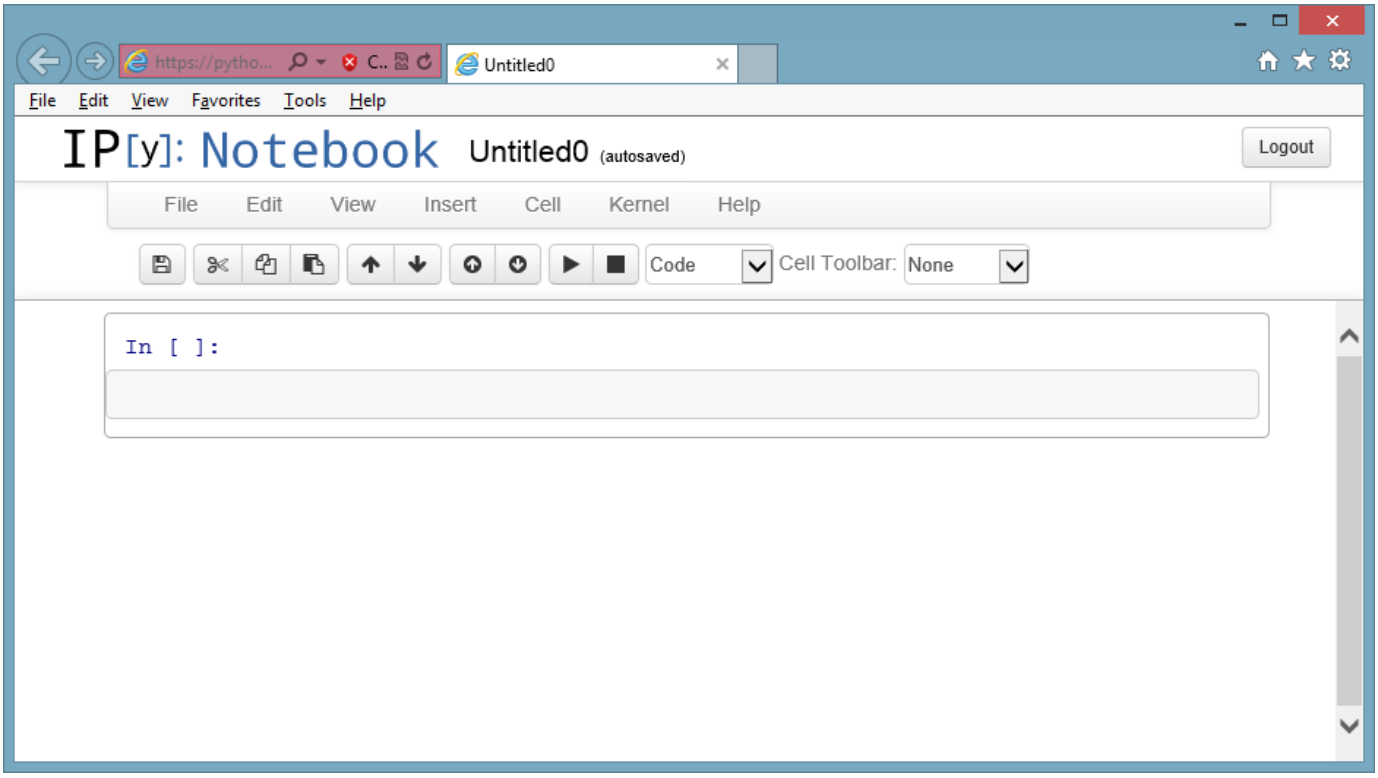
After the IPython Notebook is deployed, you can open the IPython Notebook in your Explorer:



IPython Notebook

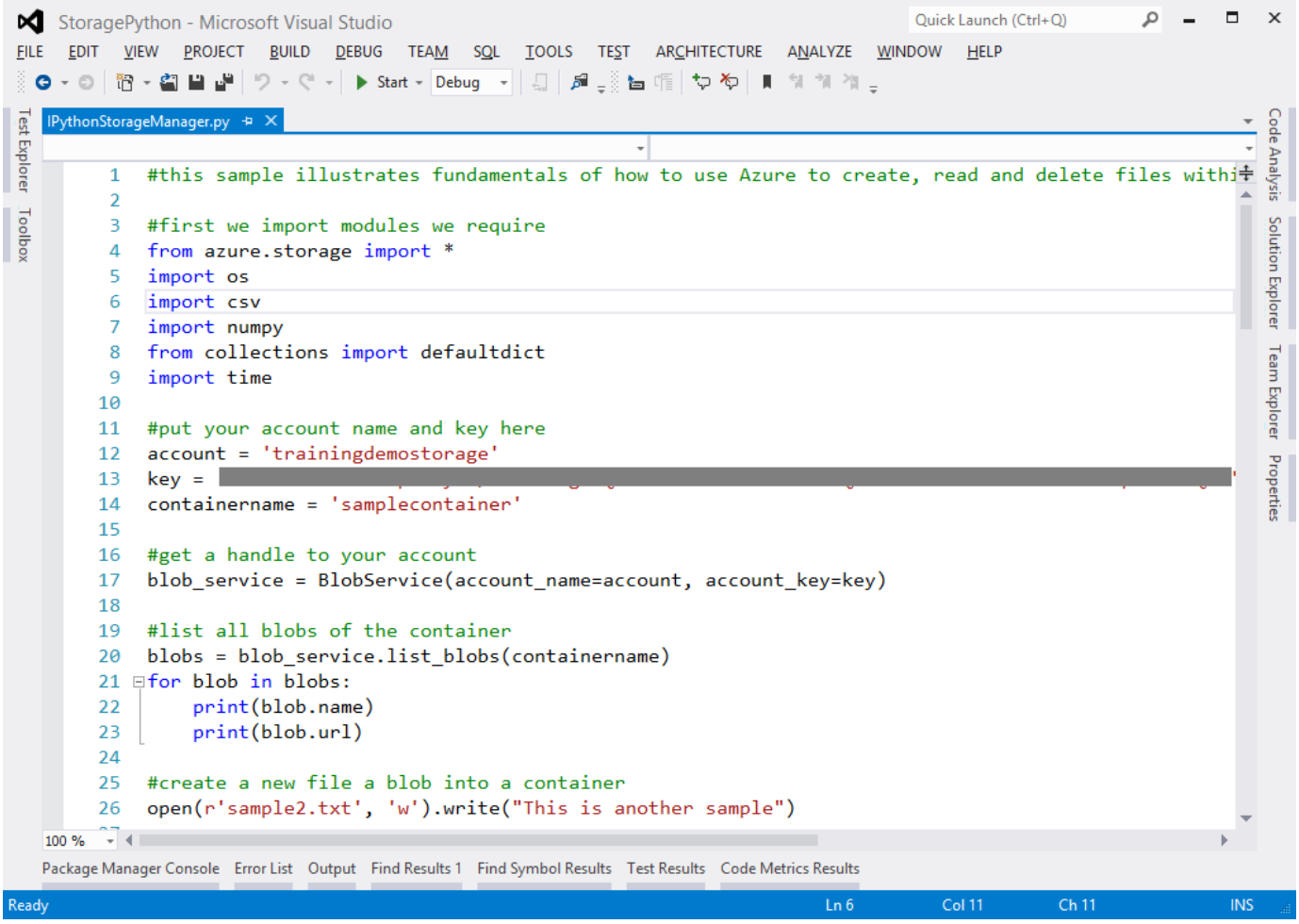
1. Create the button **New Notebook** on the top right,





Create a New Notebook

2. Next we are going to go use some Python code to manage the storage account that we created in Azure Storage Explorer. Open the file **Source\Exercise2\IPythonStorageManager.py** in a text editor and we will execute those commands step by step.

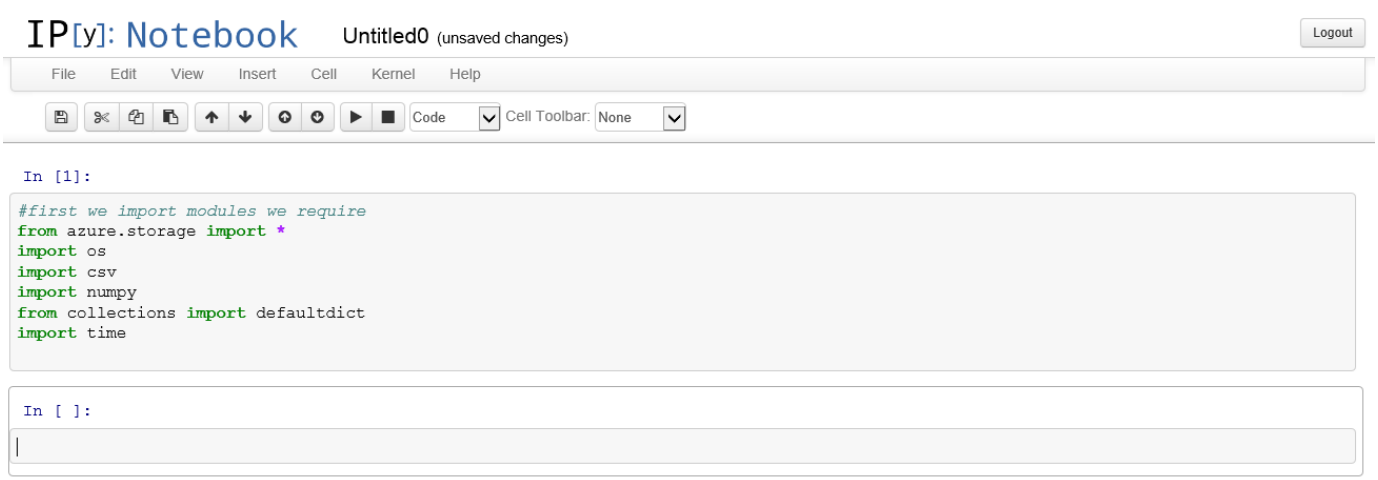


IPythonStorageManager Code

3. First we need to set the *account* and *key* variable in the code. We've just learnt how to get those information from Windows Azure Management Portal in Exercise 1. Then we will run those code in IPython Notebook.

4. Excute the following code to imports all required libraries.

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



Import Libraries

5. Then we set the private variables for the storage account

```
#put your account name and key here
account = '[Your Storage Account Name]'
key = '[You Storage Accunt Access Key]'
containername = 'samplecontainer'
```

In [2]:

```
#put your account name and key here
account = 'trainingdemostorage'
key = [redacted]
containername = 'samplecontainer'
```

Set Variables

6. We create a BlobService to manage all blobs in the storage account

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

In [3]:

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

Create Blob Service

7. Now we will list all blobs in the current storage account and container.We will print all blobs' name and full urls.

```
#list all blobs of the container
blobs = blob_service.list_blobs(containername)
for blob in blobs:
    print(blob.name)
    print(blob.url)
```

In [4]:

```
#list all blobs of the container
blobs = blob_service.list_blobs(containername)
for blob in blobs:
    print(blob.name)
    print(blob.url)
```

```
cut_diamonds.csv
http://trainingdemostorage.blob.core.windows.net/samplecontainer/cut_diamonds.csv
sample.txt
http://trainingdemostorage.blob.core.windows.net/samplecontainer/sample.txt
```

List All Blobs

You can see that we get all files that we uploaded to the container in Exercise 1.

8. Next we are going to create a new file locally and upload the file to my storage account. We create a text file *sample2.txt* and then write *This is another sample* into it.

```
#create a new file a blob into a container
open(r'sample2.txt', 'w').write("This is another sample")
#upload the blob into the container
sampleblob2 = open(r'sample2.txt', 'r').read()
blob_service.put_blob(containername, 'sample2.txt', sampleblob2, x_ms_blob_type='BlockBlob')
```

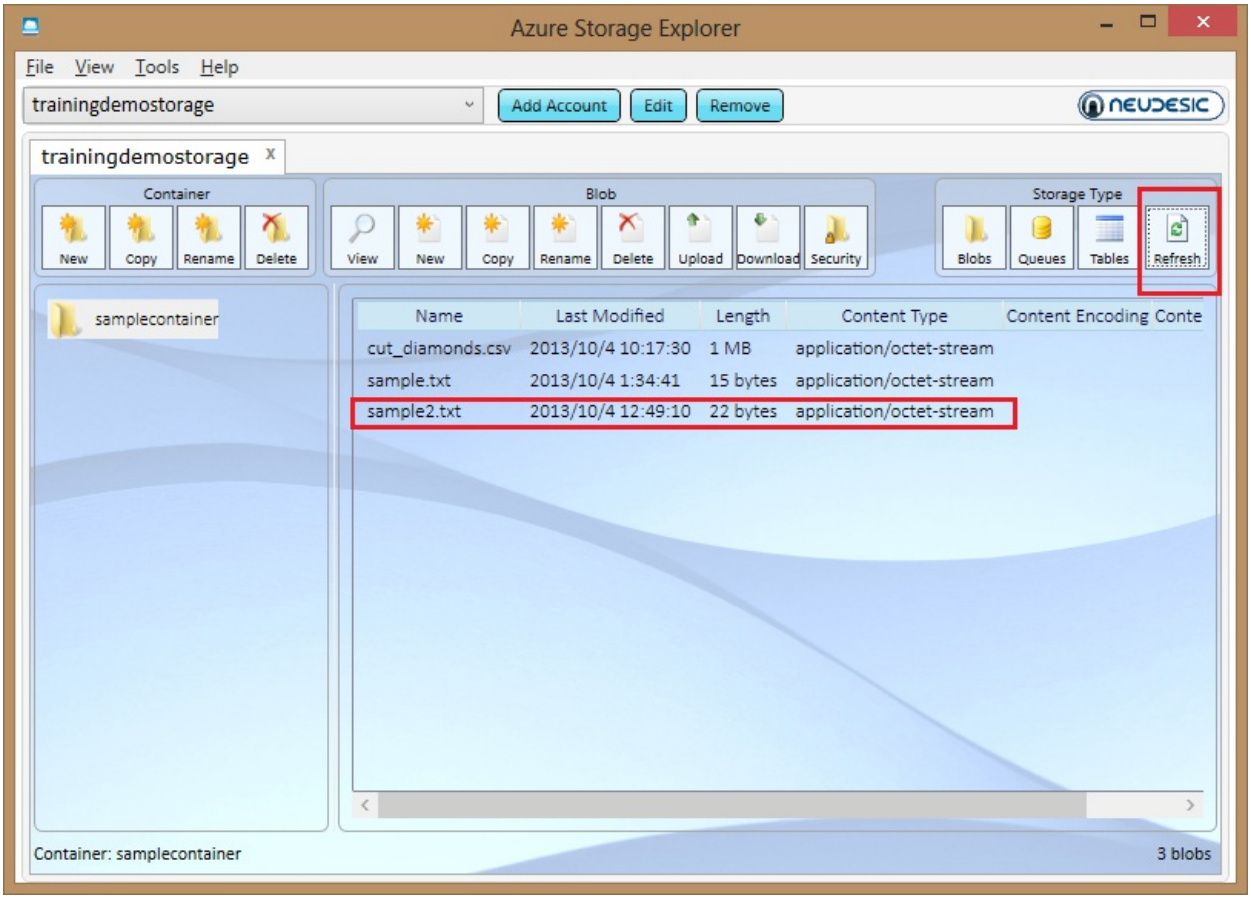
In [5]:

```
#create a new file a blob into a container
open(r'sample2.txt', 'w').write("This is another sample")

#upload the blob into the container
sampleblob2 = open(r'sample2.txt', 'r').read()
blob_service.put_blob(containername, 'sample2.txt', sampleblob2, x_ms_blob_type='BlockBlob')
#you can check the azure explorer to find the sample2.txt file
```

Upload Blob

When the upload is done, we launch Azure Storage Explorer again and refresh current container. We can see a new file *Sample2.txt" appears in the container.



The Sample2.txt is Uploaded

9. We can also delete the file in the container by following code.

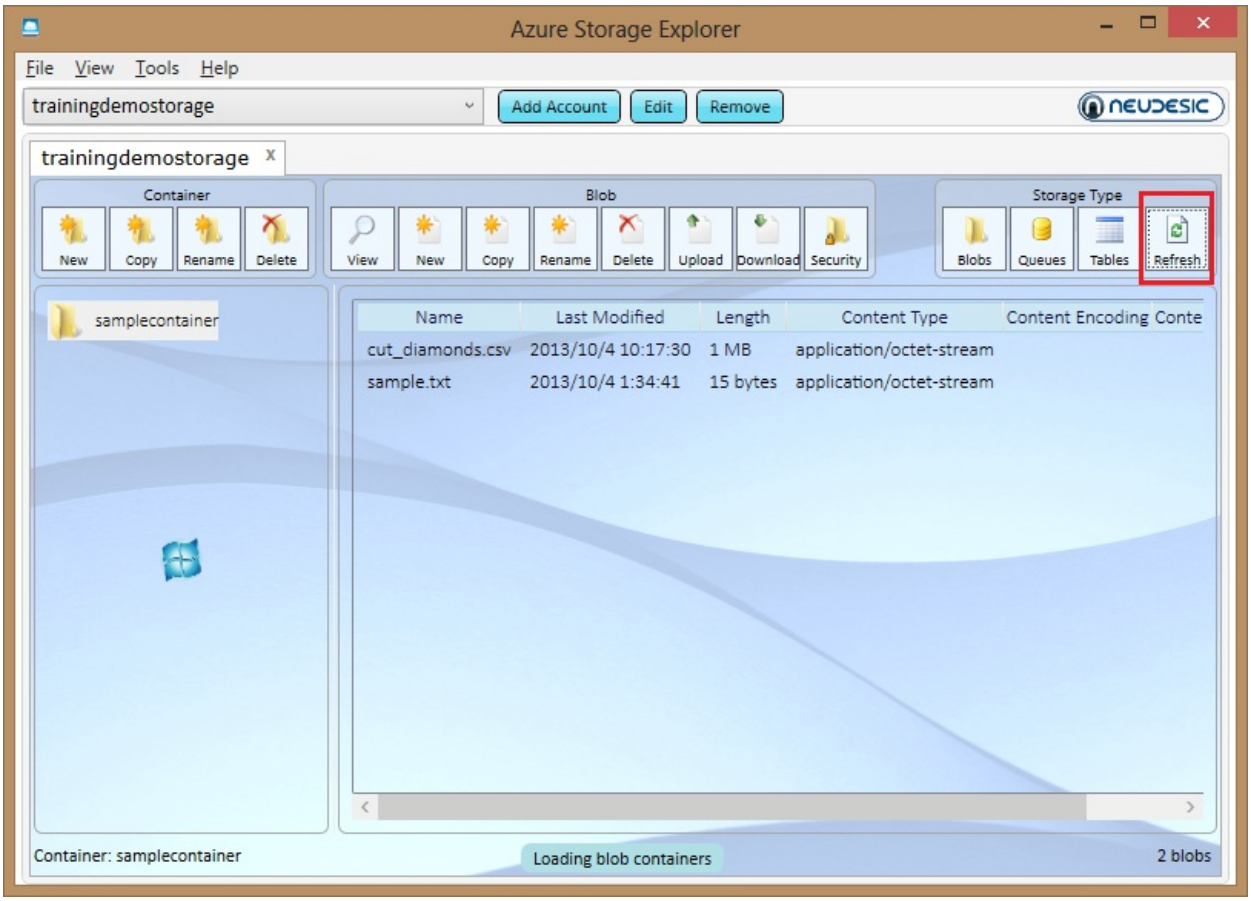
```
#then we can remove sample2.txt
os.remove(r'sample2.txt')
#delete the blob remotely
blob_service.delete_blob(containername, 'sample2.txt')
#check the azure storage explorer again, the file is removed.
```

In [6]:

```
#then we can remove sample2.txt
os.remove(r'sample2.txt')
#delete the blob remotely
blob_service.delete_blob(containername, 'sample2.txt')
#check the azure storage explorer again, the file is removed.
```

Delete Blob

Again, in the Azure Storage Explorer again and refresh current container. We can see the file *Sample2.txt" disappears.



The Sample2.txt is Deleted

10. The let's download the csv file to local and we can draw a scatter figure from the data.

```
#we can also download a csv file to local
csv_file = 'cut_diamonds.csv'
csvblob = blob_service.get_blob(containername, csv_file)
with open(csv_file, 'w') as f:
 f.write(csvblob)
```

In [7]:

```
#we can also download a csv file to local
csv_file = 'cut_diamonds.csv'
csvblob = blob_service.get_blob(containername, csv_file)
with open(csv_file, 'w') as f:
    f.write(csvblob)
```

Download Blob

11. Then we load the data in csv from the csv library and draw a scatter plot based on its carat and price.

```
#then we draw a scatter from the csvfile
columns = defaultdict(list) #we want a list to append each value in each column to
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,...}
        for (k,v) in row.items(): #go over each column name and value
            columns[k].append(v) #append the value into the appropriate list based on column name k
carat = np.array(columns['Carat'])
price = np.array(columns['Price'])
scatter(carat,price,marker ='o',color='#ff0000')
```

In [8]:

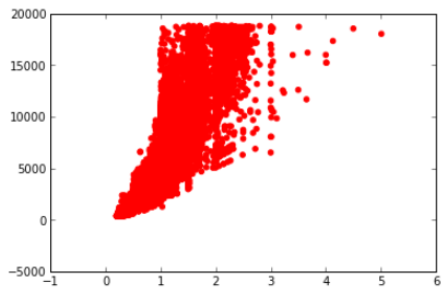
```
#then we draw a scatter from the csvfile
columns = defaultdict(list) #we want a list to append each value in each column to

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,...}
        for (k,v) in row.items(): #go over each column name and value
            columns[k].append(v) #append the value into the appropriate list based on column name k

carat = np.array(columns['Carat'])
price = np.array(columns['Price'])
scatter(carat,price,marker ='o',color='#ff0000')
```

Out[8]:

```
<matplotlib.collections.PathCollection at 0x532fa90>
```



The carat and price scatter diagram

12. Next we will also manage some table storage operation. Windows Azure Table storage is used to save many entities with different partition key and row key. It can be used as a NoSQL storage repository. First we are going to create a TableService object with the same account name and key name. We will also set the private variable to save a table name.

```
#Next we are going to demostrate the table storage management in Windows Azure
#we can add top 100 rows of the cut_diamond csv to a table storage
#get a handle to your account
table_service = TableService(account_name=account, account_key=key)
table_name = 'diamondtable';
```

In [9]:

```
#Next we are going to demostrate the table storage management in Windows Azure
#we can add top 100 rows of the cut_diamond csv to a table storage

#get a handle to your account
table_service = TableService(account_name=account, account_key=key)
table_name = 'diamondtable';
```

Create Table Service

13. Then we create a new table. First we will delete the table in case the table exists.

```
#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)
```

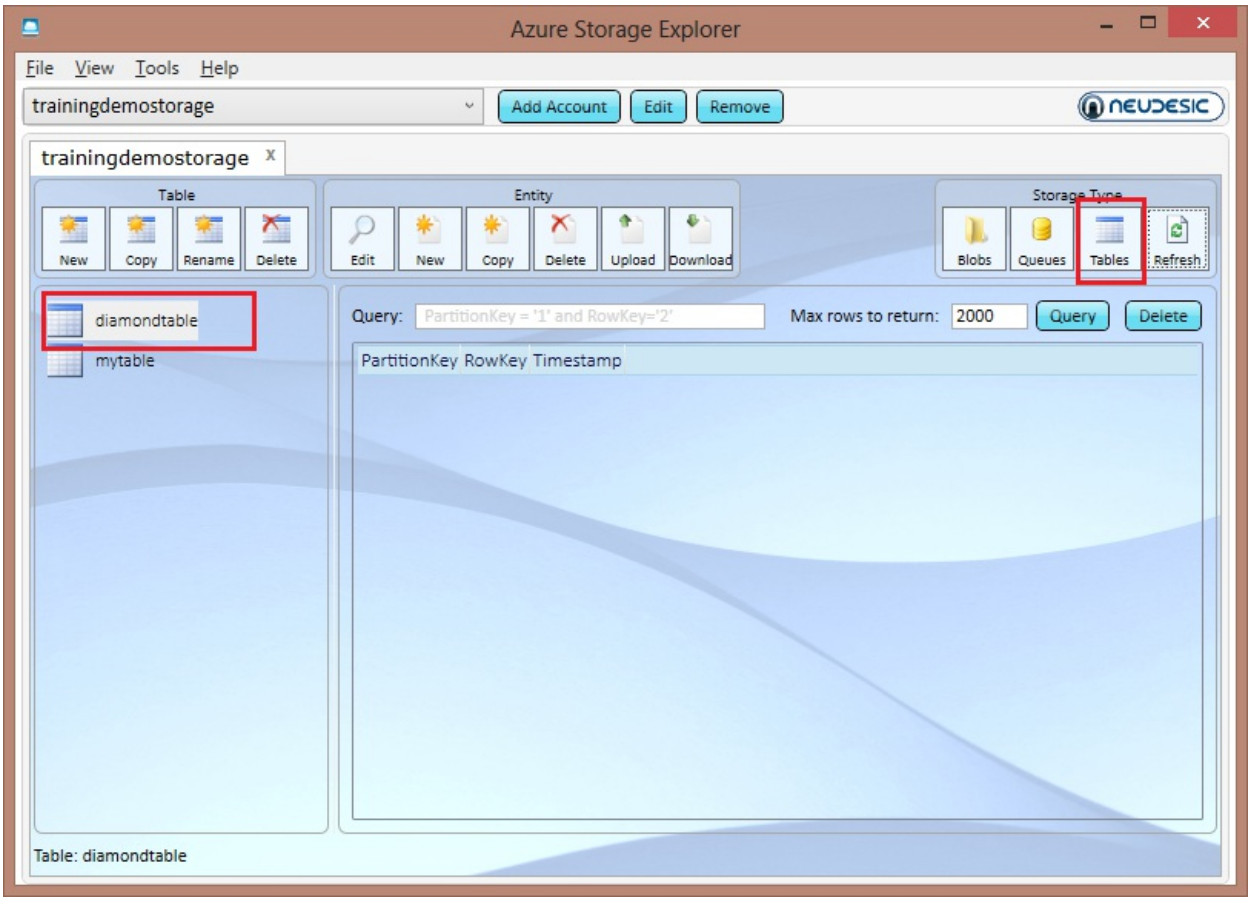
In [10]:

```
#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)
```

Create New Table

We goto the Azure Storage Explorer again, click *Tables* and check the new table.



New Table is Created

14. Now we will create 100 top entities and insert those entities into the new table. We will set each entity's partition key to be the diamond's color and row key is the index.

```
#then we insert the top 100 diamond into the table, we set PartitionKey to be each diamonds' 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.Clarity = row['Clarity']
        entity.Cut = row['Cut']
        entity.Carat = row['Carat']
        entity.Price = row['Price']
        table_service.insert_entity(table_name, entity)
        print row
        index=index+1
        if index >= 100:
            break
#we can check the azure storage explore to query all entities that we inserted.
```

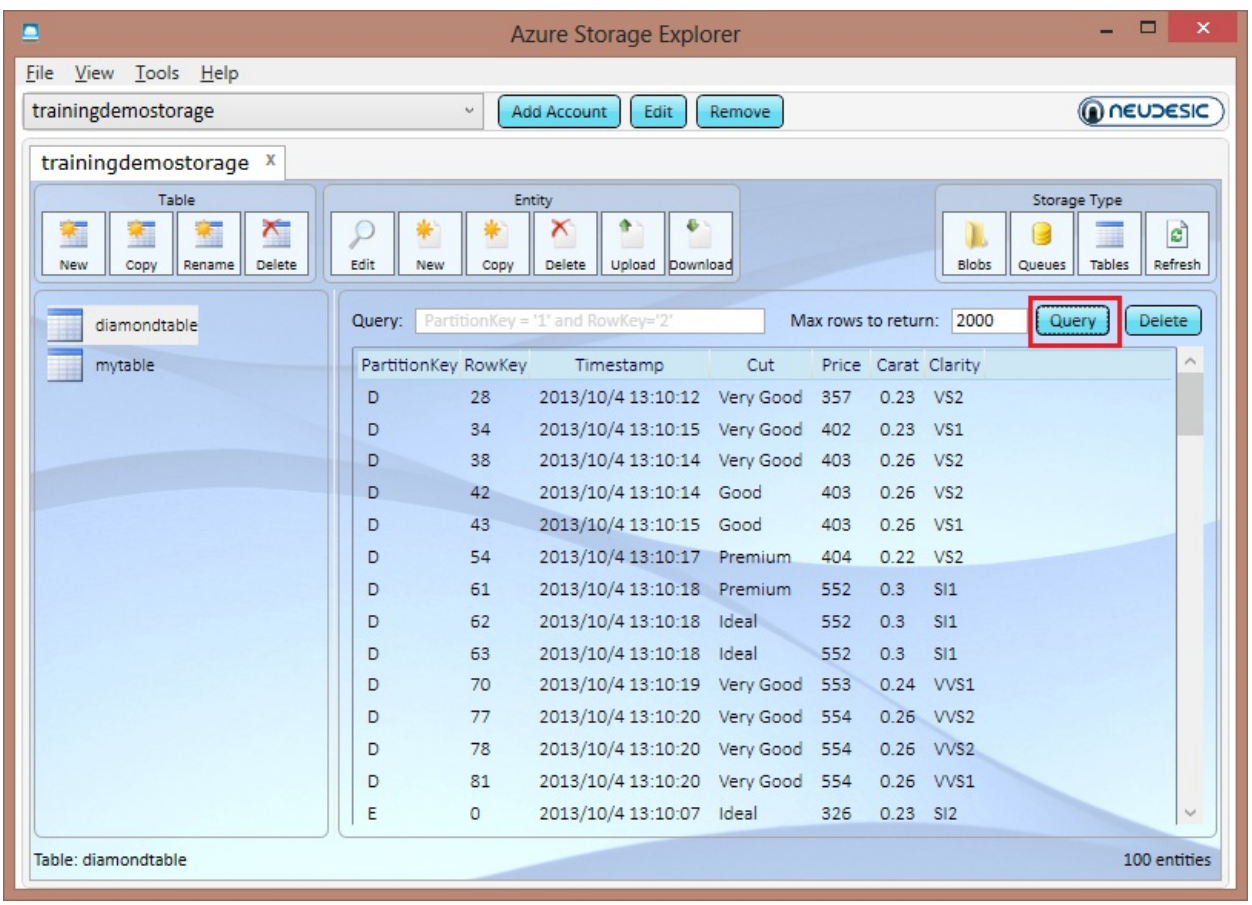
In [11]:

```
#then we insert the top 100 diamond into the table, we set PartitionKey to be each diamonds' 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.Clarity = row['Clarity']
        entity.Cut = row['Cut']
        entity.Carat = row['Carat']
        entity.Price = row['Price']
        table_service.insert_entity(table_name, entity)
        print row
        index=index+1
        if index >= 100:
            break
#we can check the azure storage explore to query all entities that we inserted.
```

```
{'': '', 'Cut': 'Ideal', 'Color': 'E', 'Price': '326', 'Carat': '0.23', 'Clarity': 'SI2'}
{'': '', 'Cut': 'Premium', 'Color': 'E', 'Price': '326', 'Carat': '0.21', 'Clarity': 'SI1'}
{'': '', 'Cut': 'Good', 'Color': 'E', 'Price': '327', 'Carat': '0.23', 'Clarity': 'VS1'}
{'': '', 'Cut': 'Premium', 'Color': 'I', 'Price': '334', 'Carat': '0.29', 'Clarity': 'VS2'}
{'': '', 'Cut': 'Good', 'Color': 'J', 'Price': '335', 'Carat': '0.31', 'Clarity': 'SI2'}
{'': '', 'Cut': 'Very Good', 'Color': 'J', 'Price': '336', 'Carat': '0.24', 'Clarity': 'VVS2'}
{'': '', 'Cut': 'Very Good', 'Color': 'I', 'Price': '336', 'Carat': '0.24', 'Clarity': 'VVS1'}
{'': '', 'Cut': 'Very Good', 'Color': 'H', 'Price': '337', 'Carat': '0.26', 'Clarity': 'SI1'}
{'': '', 'Cut': 'Fair', 'Color': 'E', 'Price': '337', 'Carat': '0.22', 'Clarity': 'VS2'}
{'': '', 'Cut': 'Very Good', 'Color': 'H', 'Price': '338', 'Carat': '0.23', 'Clarity': 'VS1'}
{'': '', 'Cut': 'Good', 'Color': 'J', 'Price': '339', 'Carat': '0.3', 'Clarity': 'SI1'}
{'': '', 'Cut': 'Ideal', 'Color': 'J', 'Price': '340', 'Carat': '0.23', 'Clarity': 'VS1'}
{'': '', 'Cut': 'Premium', 'Color': 'F', 'Price': '342', 'Carat': '0.22', 'Clarity': 'SI1'}
```

Insert Entities

In the Azure Storage Explorer, click *Query* button and you will find all entities are inserted.



Entities Are Inserted

15. We are also perform query against the table. Now we want to get all diamonds information with D color. The code is followed:

```
#we can also query all table entities with diamonds' color = 'D'
diamonds = table_service.query_entities(table_name, "PartitionKey eq 'D'")
for d in diamonds:
```

```
print(str(d.Cut),str(d.PartitionKey),str(d.Clarity),str(d.Carat),'$'+ str(d.Price))
```

In [12]:

```
#we can also query all table entities with diamonds' color = 'D'
diamonds = table_service.query_entities(table_name, "PartitionKey eq 'D'")
for d in diamonds:
    print(str(d.Cut),str(d.PartitionKey),str(d.Clarity),str(d.Carat),'$'+ str(d.Price))
```

```
('Very Good', 'D', 'VS2', '0.23', '$357')
('Very Good', 'D', 'VS1', '0.23', '$402')
('Very Good', 'D', 'VS2', '0.26', '$403')
('Good', 'D', 'VS2', '0.26', '$403')
('Good', 'D', 'VS1', '0.26', '$403')
('Premium', 'D', 'VS2', '0.22', '$404')
('Premium', 'D', 'SI1', '0.3', '$552')
('Ideal', 'D', 'SI1', '0.3', '$552')
('Ideal', 'D', 'SI1', '0.3', '$552')
('Very Good', 'D', 'VVS1', '0.24', '$553')
('Very Good', 'D', 'VVS2', '0.26', '$554')
('Very Good', 'D', 'VVS2', '0.26', '$554')
('Very Good', 'D', 'VVS1', '0.26', '$554')
```

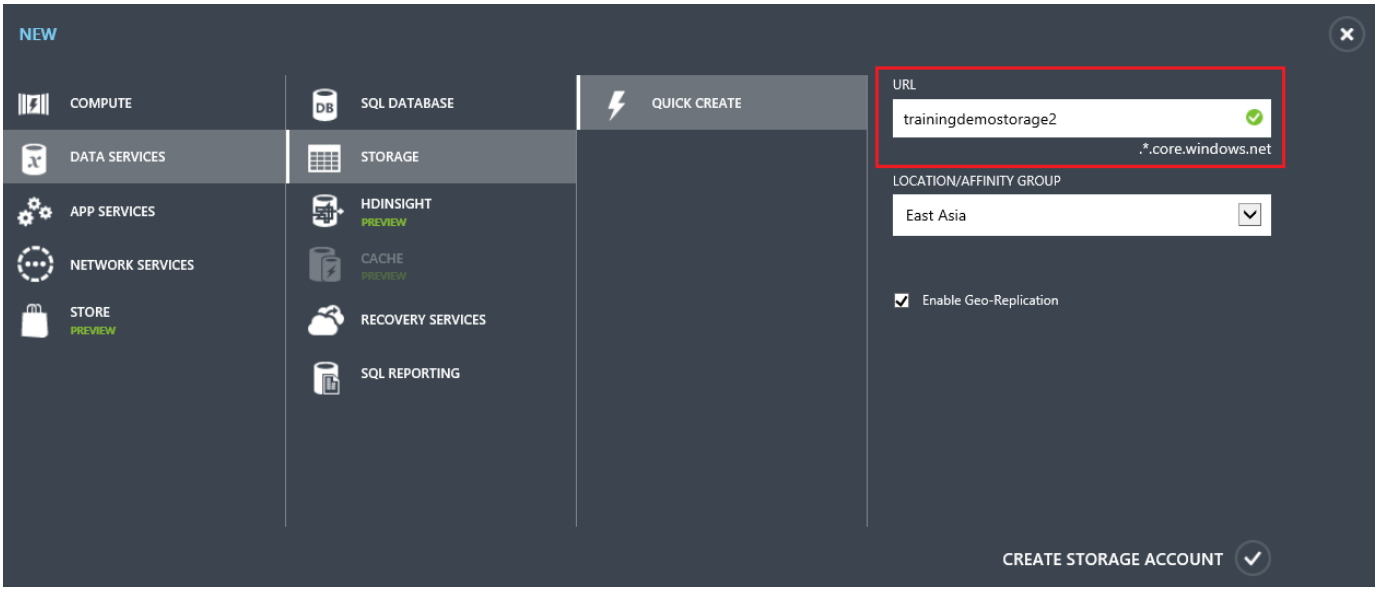
Query Table

Now we finished all operations. We can easily use IPython Notebook to manage all storage account on Windows Azure.

Excercise 3 (Optional): Use AzCopy to Copy files between different storage accounts.

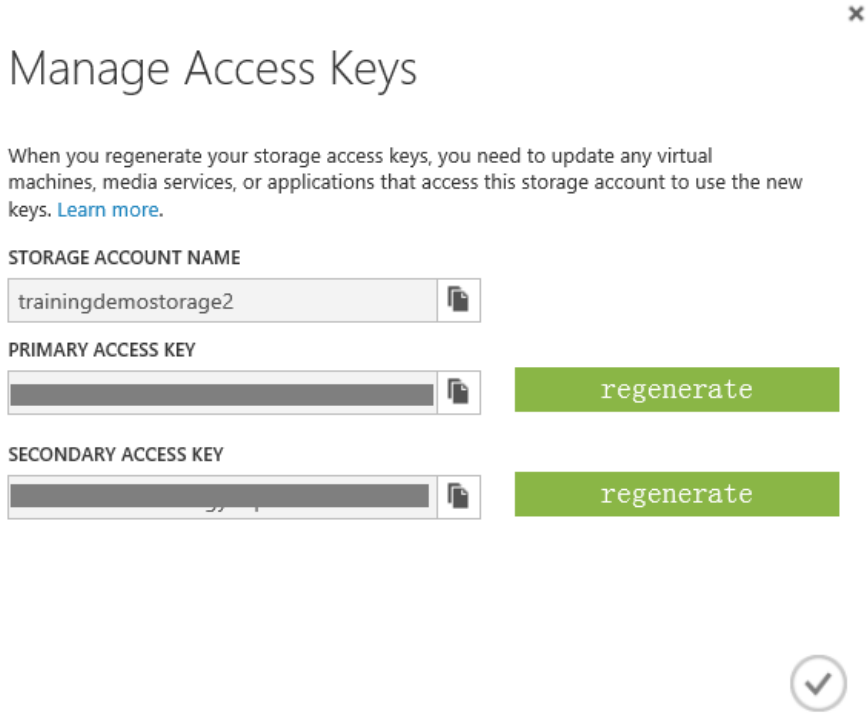
AzCopy is another tool to manage azure storage account. It can be used to copy files from local to remote storage account or even between different storage accounts. For more inforamtion, please refer to Windows Azure Storage Team Blog.

1. AzCopy.exe is distributed as .NET assemblies, we can download the CTP2 version and extract to a local folder. It is a command line tool.
2. Create a new storage account under your subscription on Windows Azure Management Portal. Set the storage account name to a different name.



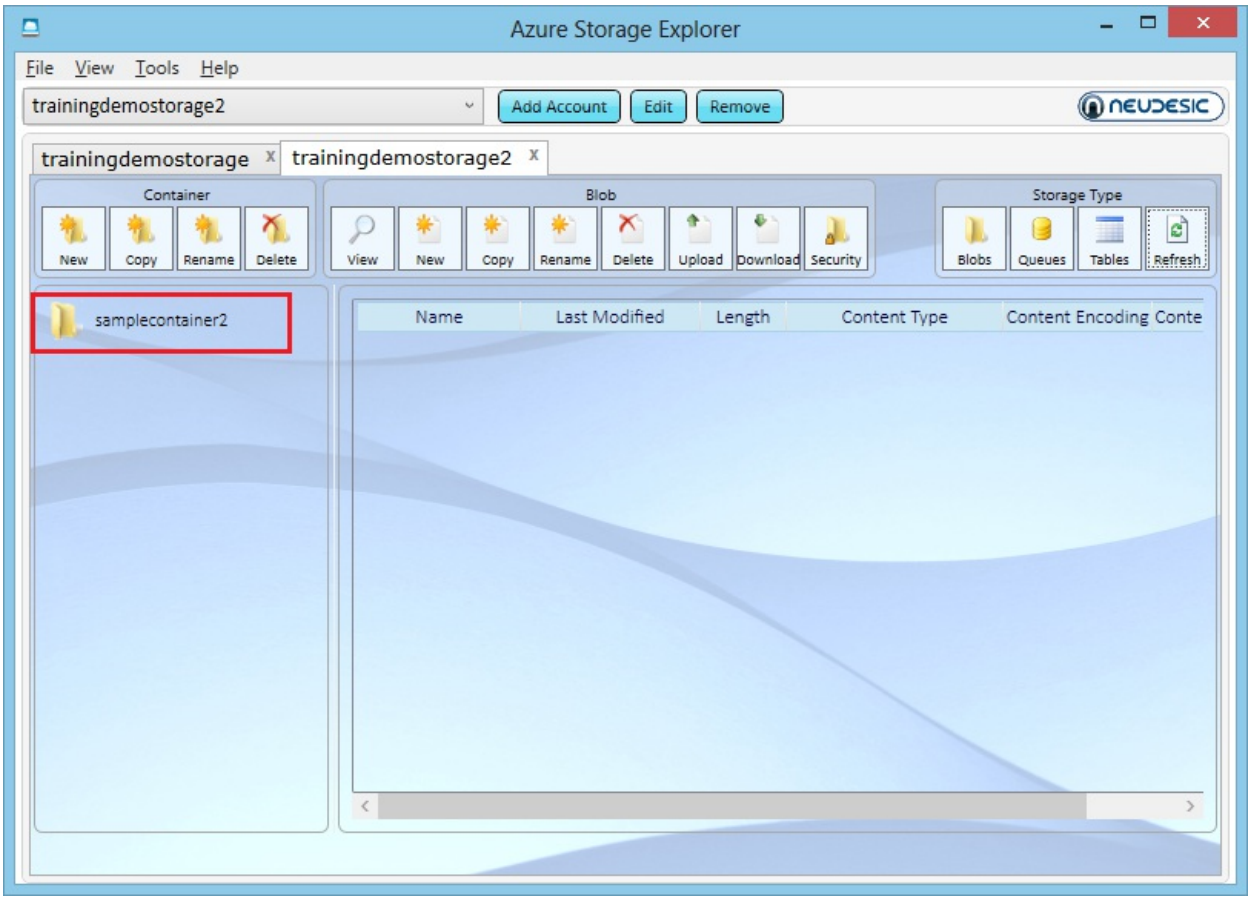
Create A New Storage Account

3. Save its access key from the portal.



Get Another Storage Account Access Key

4. Add the storage account to Azure Storage Explorer again and create a new container in the new storage account.



Create A New Container

5. Then we want to use AzCopy to copy all files from the old container to the new container. Execute the following command in commnad line:

```
AzCopy https://.blob.core.windows.net// https://.blob.core.windows.net//  /sourcekey: /destkey: /S
```

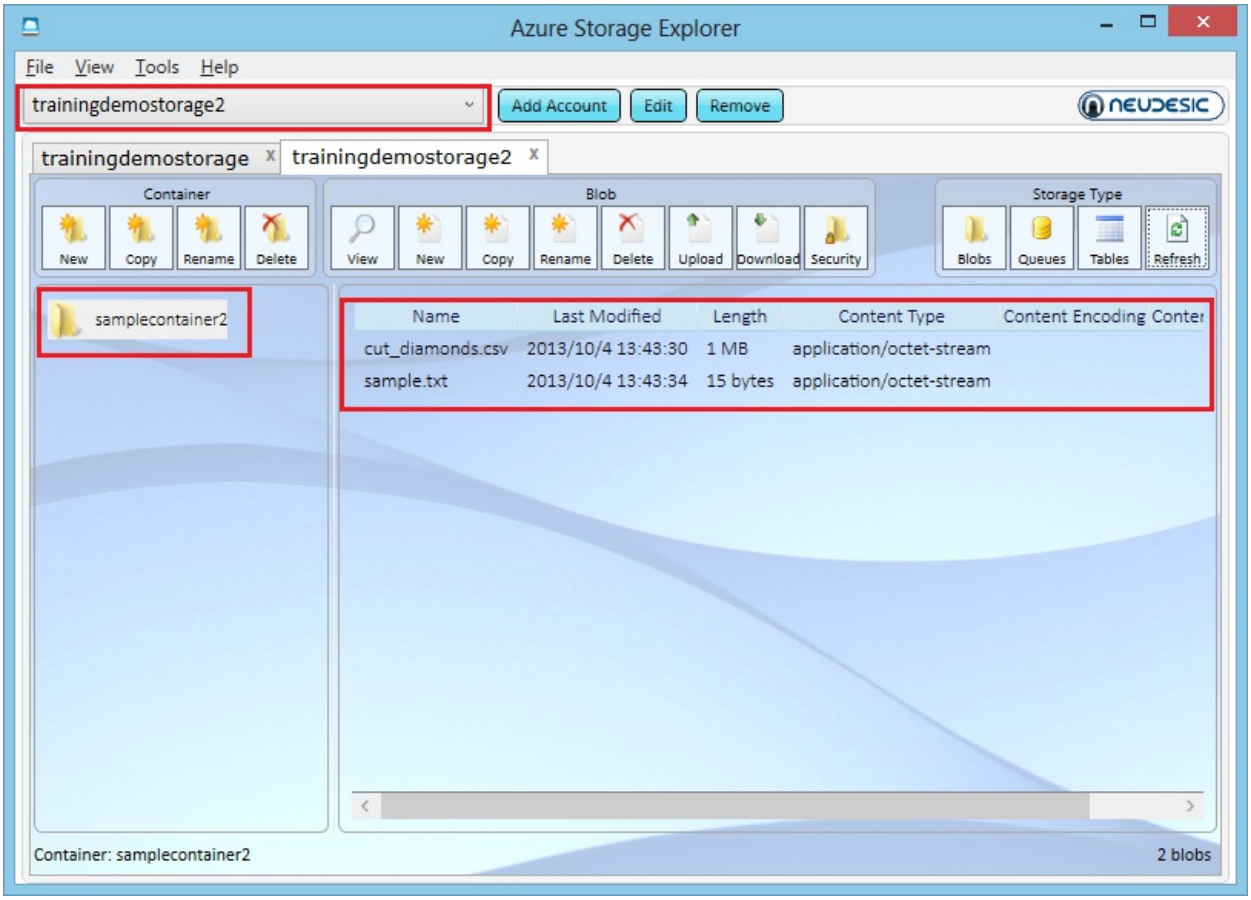
Replace all fields according to your configuration. The above command will copy all blobs from the container named "sourcecontainer" in storage account "sourceaccount" to another container named "destcontainer" in storage account "destaccount".

```
D:\AzCopy>AzCopy https://trainingdemostorage.blob.core.windows.net/samplecontain
er/ https://trainingdemostorage2.blob.core.windows.net/samplecontainer2/  /sourc
ekey:mdJnNXPYS1cC5T4pf+9yva/UJ64sU1giJQ0clXoWHszU6i3sS3CZVmQaHctL1VbE9I7HeOODPv5
HXIqVYY52nQ== /destkey:90u4XaKE5I6Nt/h6T8rCMlYBHmAK8dQirPsOohZ6sceJ2dyrTHbOGXXXO
HdPrN5ZpRf36HgsGBATwgBdi8IwdQ== /S
Transferring files |

Transfer summary:
-----------------
Total files transferred: 2
Transfer successfully:   2
Transfer failed:         0
```

AzCopy Between Storage Accounts

Let's go to the Azure Storage Explorer again and you will find all files are copied to the new container.



Check Result in Azure Storage Explorer

AzCopy also support many other features like move, snapshot and multiple network calls. For more details, please refer to the AzCopy page.

Summary

By completing this hands-on lab you learned the following:

- Use Azure Storage Explorer to manage your storage accounts.
- Use IPython notebook to run storage commands.
- Use AzCopy to Copy files between different storage accounts.