Big Data For Engineers – Exercises

Spring 2020 – Week 2 – ETH Zurich

Exercise 1: Storage devices

In this exercise, we want to understand the differences between SSD, HDD, and SDRAM in terms of capacity, speed and price. In simple terms, SDRAM (most commonly referred to as DRAM) is volatile storage (information is lost when the system is shutdown) while HDD and SSD are persistent storage (information is saved even after shutdown); SSD is an evolution of HDD where the mechanical disk was replaced by a single circuit.

All these three types of devices are used for storing data, but do that at different costs. Your task is to fill in the table below by visiting your local online hardware store. For instance, you can visit Digitec.ch to explore the prices on SSDs, HDDs, and SDRAMs. You are free to use any other website for filling the table.

You can modify the current "markdown cell" by double-clicking it and editing the table below
To return back from the editing mode to the view mode press *Ctrl+Enter*, or *Shift+Enter*

Storage Device	Maximum capacity, GB	Price, CHF/GB	Read speed, GB/s	Write speed, GB/s
HDD				
SSD				
DRAM				

Questions:

1. What type of storage devices above is the cheapest one?
2. What type of storage devices above is the fastest in terms of read speed?

Exercise 2: HTTP

HTTP is the underlying protocol used by the World Wide Web. It defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

The HTTP protocol is based on requests (usually made by a client, eg: your web browser) and responses (usually made by a server hosting a particular website or application).

Request

POST /path/index.html HTTP/1.1
Host: www .example.com User-Agent: Mozilla/4.0

BookId=3131&Author=Asimov

This fragment indicates the HTTP method used
This indicate the relative path of the resource
This is the HTTP version
These are the headers of the request (1 per line)
This is the body of the request

Response

HTTP/1.1 200 OK
Date: Tue, 25 Sep 2018 09:48:34 GMT Content-Type: text/html; charset=UTF-8 Content-Length: 138

<html> <head> <title>An Example Page</title> </head> <body> Hello World, this is a very simple HTML document. </body> </html>

This fragment indicates the status code of the response
This is the HTTP version
These are the headers of the response (1 per line)
This is the body of the response

HTTP Methods

Consider a well-designed object storage service providing a REST API implemented over HTTP.

1. Which HTTP method allows the retrieval of objects on the server? What do we mean by saying that it should be side-effect free?
2. Which HTTP methods allow the insertion and deletion of objects from the server, respectively?
3. Which other generic method allows for sending information and/or receiving results?

Exercise 3: Storage

Object Storage, Scalability

1. What are the four most important traits of Object Storage that allows scalability?
2. What are the two ways through which you can scale beyond one machine?

Azure Blob Storage vs Amazon S3

For each question give the answer for both: Azure Blob Storage and Amazon S3

1. How are objects identified?
2. What kind of objects can you create?

Exercise 4: Set up an Azure storage account

In this exercise we will execute the following steps:

1. Create a Blob storage.
2. Test Blob storage by writing to it through Python.

Step 1: Create a Locally redundant storage

1. First, you need to create a storage account. To do that, go to "Storage accounts" in the left menu. Click on "Add" at the top of the blade and fill out the following form. Choose the "exercise01" resource group or create a new one, select "Locally redundant storage (LRS)" as replication mode, and let all other values at their defaults.
2. Open the new storage account (you may need to wait a while before it has been created), go to "Access keys" (under "Settings") and copy one of its keys to the clipboard.
3. Paste the key, the account name, and some container name into the following cell and run the cell.

In [ ]:

```
# use the credentials from your server

ACCOUNT_NAME = 'YOUR_STORAGE_ACCOUNT_NAME'
ACCOUNT_KEY  = 'YOUR_KEY'
CONTAINER_NAME = 'exercise02'
```

Step 2: Install a management library for Azure storage and import it

1. Run three cells below for that

In [ ]:

```
# uninstall the old version if present
!pip uninstall azure-storage
!pip uninstall azure
```

In [ ]:

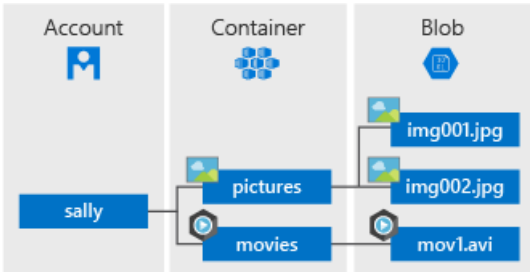
```
# install the proper packages
!pip install --upgrade requests
!pip install --upgrade cryptography
!pip install azure
!pip install azure-storage-blob
```

In [ ]:

```
from azure.storage.blob import BlockBlobService
from azure.storage.blob import PageBlobService
from azure.storage.blob import AppendBlobService
from azure.storage.blob import PublicAccess
from azure.storage.blob import ContentSettings
from azure.storage.blob import BlockListType
from azure.storage.blob import BlobBlock
```

Step 3: Explore Concepts of Azure Blob Storage

1. A container organizes a set of blobs. All blobs must be in a container. An account can contain an unlimited number of containers. A container can store an unlimited number of blobs. Note that the container name must be lowercase.



Block blobs let you upload large files to Azure Blob Storage. The maximum size of a block blob is about 4.78 TB.

In [ ]:

```
blob_service = BlockBlobService(account_name=ACCOUNT_NAME, account_key=ACCOUNT_KEY)
```

You can learn more about Python package for Azure Blob storage (git_repo, tutorial)

Step 4: Test your first container

1. Now we need to create a new container under the specified account. If the container with the same name already exists, the operation fails and returns False

In [ ]:

```
status = blob_service.create_container(CONTAINER_NAME)
#status = blob_service.create_container(CONTAINER_NAME,
#        public_access=PublicAccess.Container) #create the server with public access
if status==True:
    print("We created a new container")
else:
    print("Container already exists")
```

2. Download a file to your Jupyter's virtual machine.

In [ ]:

```
!wget https://www.vetbabble.com/wp-content/uploads/2016/11/hiding-cat.jpg -O cat.jpg # wget downloads files through HTTP. The -O
```

option indicates the name of the downloaded file.

3. Finally, upload the file to the blob storage

Note 1: The name of the file on local machine can differ from the name of the blob

Note 2: After uploading the file, check that you can access the file through Azure portal (Storage Accounts -> select your storage account -> Containers -> 'exercise02').

In [ ]:

```
local_file = "cat.jpg"

blob_name = "picture.jpg"
try:
    blob_service.create_blob_from_path(
        CONTAINER_NAME,
        blob_name,
        local_file,
        content_settings=ContentSettings(content_type='image/jpg')
    )

    print(blob_service.make_blob_url(CONTAINER_NAME,blob_name))
except:
    print ("something wrong happened when uploading the data %s"%blob_name)
```

Let's also update a text file for later

In [ ]:

```
%%bash
echo "Please don't use java" > test.txt
cat test.txt
```

In [ ]:

```
local_file = "test.txt"
blob_name = "test_file.txt"
try:
    blob_service.create_blob_from_path(
        CONTAINER_NAME,
        blob_name,
        local_file,
        content_settings=ContentSettings(content_type='text/plain;charset=UTF-8')
    )

    print(blob_service.make_blob_url(CONTAINER_NAME,blob_name))
except:
    print ("something wrong happened when uploading the data %s"%blob_name)
```

4. Try to open the link above

By default, the new container is private, so you must specify your storage access key (as you did earlier) to download blobs from this container. If you want to make the blobs within the container available to everyone, you can create the container and pass the public access level using the following code.

In [ ]:

```
blob_service.set_container_acl(CONTAINER_NAME, public_access=PublicAccess.Container)
```

5. After this change, anyone on the Internet can see blobs in a public container, but only you can modify or delete them. Now, try to open the link again (It takes few seconds to change access permisions)

6. List all blobs in the container

To list the blobs in a container, use the list_blobs method. This method returns a generator. The following code outputs name, type, size and url of each blob in a container

In [ ]:

```
# List all blobs in the container
blobs = blob_service.list_blobs(CONTAINER_NAME)
for blob in blobs:
    try:
        print('Name: {} \t\t Type: {} \t\t Size: {}'.format(blob.name,blob.properties.blob_type, blob.properties.content_length))
        print(blob_service.make_blob_url(CONTAINER_NAME,blob.name)) #We can also print the url of a blob
    except:
        print("something went wrong")
```

7. List all containers in your storage

Similarly, you can list all containers in your account

In [ ]:

```
# List all containers in your storage
containers = blob_service.list_containers()
for container in containers:
    try:
        print(container.name)
    except:
        print("something went wrong")
```

8. Download blobs

To download data from a blob, use get_blob_to_path, get_blob_to_stream, get_blob_to_bytes, or get_blob_to_text. They are high-level methods that perform the necessary chunking when the size of the data exceeds 64 MB.

Note: The Name of a file after downloading can differ from the name of a blob

The following example demonstrates using get_blob_to_path to download the content of your container and store it with names file_i where i is a sequential index.

In [ ]:

```
LOCAL_DIRECT = "./"

blobs = blob_service.list_blobs(CONTAINER_NAME)
i=0
for blob in blobs:
    local_file = LOCAL_DIRECT + 'file_{}'.format(str(i))
    i=i+1
    try:
        blob_service.get_blob_to_path(CONTAINER_NAME, blob.name, local_file)
        print("success for %s"%local_file)
    except:
        print("something went wrong with the blob: %s"%blob.name)
```

In [ ]:

```
!ls
```

9. Delete blobs

Finally, to delete a blob, we can use the method **delete_blob**.

In [ ]:

```
#Delete all blobs in the container
blobs = blob_service.list_blobs(CONTAINER_NAME)
for blob in blobs:
    try:
        blob_service.delete_blob(CONTAINER_NAME, blob.name)
        print("deleted  correctly : %s"%blob.name)
    except:
        print("something went wrong with the blob: %s"%blob.name)
```

To check whether the container or the blob exists, the method `exists` can be used.

In [ ]:

```
if (blob_service.exists(CONTAINER_NAME,blob.name)==True):
    print("Blob %s exists"%blob.name)
if(blob_service.exists(CONTAINER_NAME)==True):
   print("Container %s exists"%CONTAINER_NAME)
```

Own exploration

Exercise 5: Explore the REST API

REpresentational State Transfer (REST), or RESTful, web services provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of *stateless* operations.

The most popular operations in REST are GET, POST, PUT, DELETE. A response may be in XML, HTML, JSON, or some other format.

In this exercise we will use REST online client ReqBin.

URLs tells the server which resources the client wants to interact with.

The **method** request tells the server what kind of action the client wants the server to take in.

Headers provide meta-information about a request.

The request **body** contains the data the client wants to send the server.

Task 1.

1. Use ReqBin to list all blobs in your container. For this use the following REST request. What does it do? To avoid setting up an authentication you can make your container public.
   A. Access your container from all resources > your storage account > "Blobs" in the overview tab > your container name
   B. On the top menu of the tab "Change access level" and set for public access level "Container"
2. What is the response format of the request?
3. Try to perform the following REST requests Get Blob, Get Blob Properties, Get Blob Metadata using ReqBin.