Capstone Project Cloud Consideration (Step 06)

1. Cluster

1) Using aztk to spin up a low cost spark cluster is an ideal solution. However, followed the instructions I was not able to set it up
   a) The account set up script, 'account_setup.sh &&chmod 755 account_setup.sh &&/bin/bash account_setup.sh', didn't work. So I manually set up the service principle (AAD, APP registration, storage account, batch account)

   b) When creating cluster, got following error

```
root@DESKTOP-NV1A4OE:/home/chu/aztk# aztk spark cluster create \
>   --id test01 \
>   --vm-size standard_ds2_v2 \
>   --size 3 \
>   --username demouser01 \
>   --password P@ssw0rd
----------------------------------------
cluster id:              test01
cluster toolkit:         spark 2.3.0
cluster size:            3
>        dedicated:      3
>     low priority:      0
cluster vm size:         standard_ds2_v2
subnet ID:               None
file shares:             0
gpu enabled:             False
docker repo name:        aztk/spark:v0.1.0-spark2.3.0-base
wait for cluster:        False
username:                demouser01
Password: ********
Plugins:
    None Configured
```

2) HDinsight
   I tried Azure HDInsight, here are the results
   a) I was able to spin up a cluster and SSH to it.
   b) A quick testing shows, after adding configuration logic, the program can access S3 or azure blob.
   c) After carefully choosing cluster configuration parameters, the cost could be acceptable

   So HDinsight is a good solution, which we will use for now. If I later find some other solutions (such as EMR) are better, we can change

2. Cloud storage

I tried Azure blob and AWS S3 both.

For S3, it is easier to set up – I was able to set it up on my local machine. Two issues found so far

1. One program (in Integration process)crashed when calculating on a big volume of data (:error message 'ConnectionClosedException: Premature end of Content-Length delimited message body'). I did some research. But still I can not determine if it was related to collecting big amount of data across the network, or because I was using an older version of software (hadoop under Spark is 2.7), or something else.

2. Access to S3 is not stable. The same code could work in one environment, for example Windows on my machine but not working in another environment, such as Ubuntu. From the error message it looks like an permission issue. I need to do more testing to figure it out.

For Azure blob, it was hard to set up – I didn't succeed on config spark up on my machine to access azure blob (so it is hard to do testing). But it worked when using HDinsight cluster and Databricks.

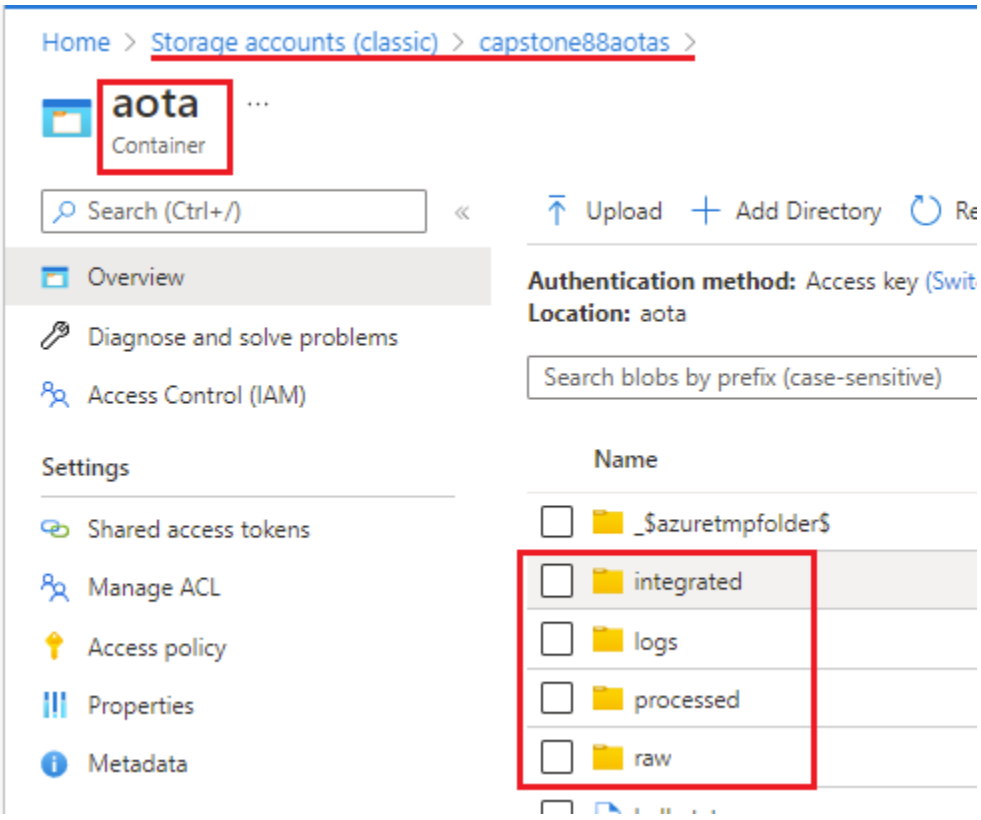
On Databricks, we can mount the azure blob. I also tested the major ETL and Integration processes with Databricks. The processes went through smoothly – the Integration process didn't crash like on AWS S3. However it was slow. ETL took 30 minutes and Integration took over 1 hour, while both processes took about 10 minutes on my local machine, and ETL took about 10 minutes with AWS S3

So now I still can not decide which storage to choose. In theory, both are possible solutions. I will do more experiments and then decide.

Here are the details of the storage I set up:

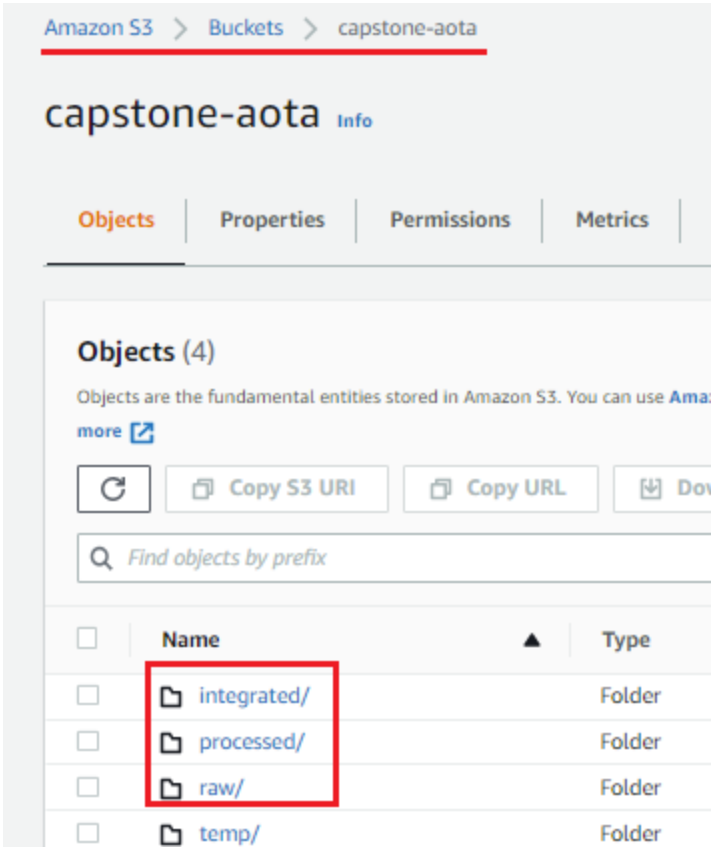
Azure Blob

	Storage account: capstone88aotas

	Container: aota
	

AWS S3 bucket

	S3 bucket: capstone-aota



3. Code changes for cloud

I have used some common utility modules to store the storage location or obtain a spark session. By making some changes on them, the code can work for cloud.

Class	Changes needed
ut_store	Will need modify the variables that point to location of the 'raw', 'processed', 'integrated' layer. folder_base ='s3a://capstone-aota2' **folder_raw** = '/data/aota2/raw/' **folder_cln** = folder_base+ '/processed/' **folder_intgr** = folder_base+ '/integrated/'

Ut_spark changes for AWS S3	.Method ‘creat_spark’
	
	This method is a common utility for every process to get a spark session. By adding cloud storage related configuration logic to this method, all the programs calling this method is able to access cloud storage.
	Here is an example when data is stored on S3::
	
	`@classmethod`
	`def creat_spark(cls):`
	`    '''Create and return a spark session'''`
	
	`    conf = SparkConf()`
	
	`    conf.set('spark.hadoop.fs.s3a.impl', 'org.apache.hadoop.fs.s3a.S3AFileSystem')`
	`    conf.set('spark.hadoop.fs.s3a.aws.credentials.provider',`
	`             'org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider')`
	`    conf.set('spark.hadoop.fs.s3a.access.key', 'A[illegible]KIM')`
	`    conf.set('spark.hadoop.fs.s3a.secret.key', 'WF5[illegible]')`
	
	`    try:`
	`        spark = SparkSession. \`
	`            builder. \`
	`            appName("Aggregation"). \`
	`            master("local[4]"). \`
	`            config(conf=conf).\`
	`            getOrCreate()`
	
	`        spark.conf.set("spark.sql.shuffle.partitions", cls.c_spark_sql_shuffle_partitions)`
	
	`        return spark`

4. Other things noticed

Basically, here are something I found:

1) Download process won't work with either S3 or Azure blob.

   The problem is, when the downloader opens a target file and tries to write to it, the Python File write () Method won't recognize the cloud storage path and error out with a message, 'The file doesn't exist'.

   For alternative solutions. I am thinking of two ways:

Method 1: Download the source files to a local machine(where the ETL pipeline is running). Then using AWS cli or Azure CLI to copy the files to the 'raw' location on cloud.

Method 2: Download the source files to the local machine. Run a pyspark script to read from the source files (csv, bz2), and write to parquet files to the 'raw' location on cloud. A good thing about this method is that parquet files have more advantages over csv files so it is good to store the 'raw' files as parquet.

2) Log files can not be put on the cloud. It looks like logging.logger won't recognize the cloud file path. So we will put the log files on the cluster.