Data Ingestion

Ingests and parses data from Blob container and writes to Databricks cluster

In [ ]:
```
from datetime import datetime
from pyspark.sql.types import StructType, StructField, DateType, StringType, TimestampType, IntegerType, FloatType
import json
import os
```

In [ ]:
```
# Azure credentials
storageAccountName = 'exchangedata1'
storageAccountAccessKey = '<your-access-key>'
ContainerName = 'source-container'
```

In [ ]:
```
# mount azure blob container to make files accessible to PySpark
if not any(mount.mountPoint == '/mnt/blobsource/' for mount in dbutils.fs.mounts()):
  try:
    dbutils.fs.mount(
    source = "wasbs://{}@{}.blob.core.windows.net".format(ContainerName, storageAccountName),
    mount_point = "/mnt/blobsource",
    extra_configs = {'fs.azure.account.key.' + storageAccountName + '.blob.core.windows.net': storageAccountAccessKey}
  )
  except Exception as e:
    print("already mounted. Try to unmount first")

display(dbutils.fs.ls('mnt/blobsource'))
```

```
already mounted. Try to unmount first
```

path	name	size
dbfs:/mnt/blobsource/data/	data/	0

Parse CSV and JSON files

In [ ]:
```
#parse CSV and create common schema for data types and structure
#create partition columns
def parse_csv(value):
    field=value.split(',')
    try:
        if field[2]=='Q':
            trade_dt=datetime.strptime(field[0], '%Y-%m-%d')
            rec_type=field[2]
            symbol=field[3]
            exchange=field[6]
            event_tm=datetime.strptime(field[4], '%Y-%m-%d %H:%M:%S.%f')
            event_seq_nb=int(field[5])
            arrival_tm=datetime.strptime(field[1], '%Y-%m-%d %H:%M:%S.%f')
            trade_pr=None
            bid_pr=float(field[7])
            bid_size=int(field[8])
            ask_pr=float(field[9])
            ask_size=int(field[10])
            partition='Q'
            return (trade_dt,rec_type,symbol,exchange,event_tm,event_seq_nb,arrival_tm,trade_pr,bid_pr,bid_size,ask_pr,ask_size,partition)
        elif field[2]=='T':
            trade_dt=datetime.strptime(field[0], '%Y-%m-%d')
            rec_type=field[2]
            symbol=field[3]
            exchange=field[6]
            event_tm=datetime.strptime(field[4], '%Y-%m-%d %H:%M:%S.%f')
            event_seq_nb=int(field[5])
            arrival_tm=datetime.strptime(field[1], '%Y-%m-%d %H:%M:%S.%f')
            trade_pr=float(field[7])
            bid_pr=None
            bid_size=None
            ask_pr=None
            ask_size=None
            partition='T'
            return (trade_dt,rec_type,symbol,exchange,event_tm,event_seq_nb,arrival_tm,trade_pr,bid_pr,bid_size,ask_pr,ask_size,partition)
    except Exception as e:
            trade_dt=None
            rec_type=None
            symbol=None
            exchange=None
            event_tm=None
            event_seq_nb=None
            arrival_tm=None
            trade_pr=None
            bid_pr=None
            bid_size=None
            ask_pr=None
            ask_size=None
```

```
            partition='B'
            return (trade_dt,rec_type,symbol,exchange,event_tm,event_seq_nb,arrival_tm,trade_pr,bid_pr,bid_size,ask_pr,ask_size,partition)
```

In [ ]:

```
#parse json and create common schema for data types and structure
#create partition columns
def parse_json(value):
    field=json.loads(value)
    try:
        if field['event_type']=='Q':
            trade_dt=datetime.strptime(field['trade_dt'], '%Y-%m-%d')
            rec_type=field['event_type']
            symbol=field['symbol']
            exchange=field['exchange']
            event_tm=datetime.strptime(field['event_tm'], '%Y-%m-%d %H:%M:%S.%f')
            event_seq_nb=int(field['event_seq_nb'])
            arrival_tm=datetime.strptime(field['file_tm'], '%Y-%m-%d %H:%M:%S.%f')
            trade_pr=None
            bid_pr=float(field['bid_pr'])
            bid_size=int(field['bid_size'])
            ask_pr=float(field['ask_pr'])
            ask_size=int(field['ask_size'])
            partition='Q'
            return (trade_dt,rec_type,symbol,exchange,event_tm,event_seq_nb,arrival_tm,trade_pr,bid_pr,bid_size,ask_pr,ask_size,partition)
        elif field['event_type']=='T':
            trade_dt=datetime.strptime(field['trade_dt'], '%Y-%m-%d')
            rec_type=field['event_type']
            symbol=field['symbol']
            exchange=field['exchange']
            event_tm=datetime.strptime(field['event_tm'], '%Y-%m-%d %H:%M:%S.%f')
            event_seq_nb=int(field['event_seq_nb'])
            arrival_tm=datetime.strptime(field['file_tm'], '%Y-%m-%d %H:%M:%S.%f')
            trade_pr=float(field['price'])
            bid_pr=None
            bid_size=None
            ask_pr=None
            ask_size=None
            partition='T'
            return (trade_dt,rec_type,symbol,exchange,event_tm,event_seq_nb,arrival_tm,trade_pr,bid_pr,bid_size,ask_pr,ask_size,partition)
    except Exception as e:
            trade_dt=None
            rec_type=None
            symbol=None
            exchange=None
            event_tm=None
            event_seq_nb=None
            arrival_tm=None
            trade_pr=None
            bid_pr=None
            bid_size=None
            ask_pr=None
            ask_size=None
            partition='B'
            return (trade_dt,rec_type,symbol,exchange,event_tm,event_seq_nb,arrival_tm,trade_pr,bid_pr,bid_size,ask_pr,ask_size,partition)
```

Read data to RDD and create pair RDD using map with parsers functions

In [ ]:

```
def create_pair_rdd(exchange,date):

    if exchange=='NASDAQ':
        dir_path='/mnt/blobsource/data/json/{}/{}/'.format(date,exchange)
        dir_list=os.listdir('/dbfs'+dir_path)
        for filename in dir_list:
            if filename.endswith('.txt'):
                raw_json=spark.sparkContext.textFile("{}/{}".format(dir_path,filename))
                parsed_rdd=raw_json.map(parse_json)


    elif exchange=='NYSE':
        dir_path='/mnt/blobsource/data/csv/{}/{}/'.format(date,exchange)
        dir_list=os.listdir('/dbfs'+dir_path)
        for filename in dir_list:
            if filename.endswith('.txt'):
                raw_csv=spark.sparkContext.textFile("{}/{}".format(dir_path,filename))
                parsed_rdd=raw_csv.map(parse_csv)

    return parsed_rdd
```

In [ ]:

```
csv_rdd_one=create_pair_rdd(exchange='NYSE', date='2020-08-05')
csv_rdd_two=create_pair_rdd(exchange='NYSE', date='2020-08-06')
json_rdd_one=create_pair_rdd(exchange='NASDAQ', date='2020-08-05')
json_rdd_two=create_pair_rdd(exchange='NASDAQ', date='2020-08-06')
```

In [ ]:

```
#join the RDD for NYSE and NASDAQ together
def join_rdd(rdd_one,rdd_two):
```

```
    joined_rdd=rdd_one.union(rdd_two)
    return joined_rdd
```

In [ ]:
```
joined_rdd_one=join_rdd(csv_rdd_one,json_rdd_one)
joined_rdd_two=join_rdd(csv_rdd_two,json_rdd_two)
```

Transform RDD to Dataframe and write to parquet

In [ ]:
```
# set the schema and transform RDD to dataframe
# write dataframe to parquet partitioned by Quote, Trade, or Bad Records
def dataframe_parition(rdd,date):
    # dataframe schema to impose
    schema = StructType([
    StructField('trade_dt', DateType(), True),
    StructField('rec_type', StringType(),True),
    StructField('symbol', StringType(),True),
    StructField('exchange', StringType(),True),
    StructField('event_tm', TimestampType(),True),
    StructField('event_seq_nb', IntegerType(),True),
    StructField('arrival_tm', TimestampType(),True),
    StructField('trade_pr', FloatType(),True),
    StructField('bid_pr', FloatType(),True),
    StructField('bid_size', IntegerType(),True),
    StructField('ask_pr', FloatType(),True),
    StructField('ask_size', IntegerType(),True),
    StructField('partition', StringType(),True)])

    df=spark.createDataFrame(rdd,schema=schema)
    df.write.partitionBy('partition').mode('overwrite').parquet('output/parsed_data/{}/'.format(date))
    return
```

In [ ]:
```
dataframe_parition(joined_rdd_one,'2020-08-05')
dataframe_parition(joined_rdd_two,'2020-08-06')
```

```
+----------+--------+------+--------+--------------------+-----------+------------------+--------+---------+--------+--------+--------+---------+
  trade_dt|rec_type|symbol|exchange|            event_tm|event_seq_nb|        arrival_tm|trade_pr|   bid_pr|bid_size|  ask_pr|ask_size|partition|
+----------+--------+------+--------+--------------------+-----------+------------------+--------+---------+--------+--------+--------+---------+
2020-08-05|       Q|  SYMA|    NYSE|2020-08-05 09:34:...|          1|2020-08-05 09:30:00|    null| 75.30255|     100|75.35917|     100|        Q|
2020-08-05|       Q|  SYMA|    NYSE|2020-08-05 09:40:...|          2|2020-08-05 09:30:00|    null| 77.20875|     100|78.90918|     100|        Q|
2020-08-05|       Q|  SYMA|    NYSE|2020-08-05 09:50:...|          3|2020-08-05 09:30:00|    null| 77.15973|     100|77.33205|     100|        Q|
2020-08-05|       Q|  SYMA|    NYSE|2020-08-05 09:57:...|          4|2020-08-05 09:30:00|    null|79.299774|     100|80.08399|     100|        Q|
2020-08-05|       Q|  SYMA|    NYSE|2020-08-05 10:06:...|          5|2020-08-05 09:30:00|    null|77.863495|     100|78.30821|     100|        Q|
+----------+--------+------+--------+--------------------+-----------+------------------+--------+---------+--------+--------+--------+---------+
only showing top 5 rows

+----------+--------+------+--------+--------------------+-----------+------------------+--------+---------+--------+---------+--------+---------+
  trade_dt|rec_type|symbol|exchange|            event_tm|event_seq_nb|        arrival_tm|trade_pr|   bid_pr|bid_size|   ask_pr|ask_size|partition|
+----------+--------+------+--------+--------------------+-----------+------------------+--------+---------+--------+---------+--------+---------+
2020-08-06|       Q|  SYMA|    NYSE|2020-08-06 09:39:...|          1|2020-08-06 09:30:00|    null| 77.67913|     100|78.437355|     100|        Q|
2020-08-06|       Q|  SYMA|    NYSE|2020-08-06 09:47:...|          2|2020-08-06 09:30:00|    null| 76.53373|     100| 76.94425|     100|        Q|
2020-08-06|       Q|  SYMA|    NYSE|2020-08-06 09:56:...|          3|2020-08-06 09:30:00|    null|75.120605|     100| 75.39408|     100|        Q|
2020-08-06|       Q|  SYMA|    NYSE|2020-08-06 10:03:...|          4|2020-08-06 09:30:00|    null| 74.86369|     100| 75.76861|     100|        Q|
2020-08-06|       Q|  SYMA|    NYSE|2020-08-06 10:09:...|          5|2020-08-06 09:30:00|    null|  77.7765|     100|78.801094|     100|        Q|
+----------+--------+------+--------+--------------------+-----------+------------------+--------+---------+--------+---------+--------+---------+
only showing top 5 rows
```

In [ ]:
```
# shows directories for each date created under /output/parsed_data/
dbutils.fs.ls('/output/parsed_data/')
```

```
Out[14]: [FileInfo(path='dbfs:/output/parsed_data/2020-08-05/', name='2020-08-05/', size=0),
 FileInfo(path='dbfs:/output/parsed_data/2020-08-06/', name='2020-08-06/', size=0)]
```

In [ ]:
```
# shows current contents under each date directory, partitioned by Trade and Quote
print(os.listdir('/dbfs/output/parsed_data/2020-08-05'))
print(os.listdir('/dbfs/output/parsed_data/2020-08-06'))
```

```
['_SUCCESS', '_committed_787419416886541593', 'partition=Q', 'partition=T']
['_SUCCESS', '_committed_4428932175041918723', 'partition=Q', 'partition=T']
```

In [ ]:
```
dbutils.notebook.exit('SUCCESS')
```