Final Project

By Fengxue Zhang

1. Functionality

User type in the stock name on the webpage, the application return the historical ***Sharpe Ratio*** (reward-to-variability ratio), one of the most popular measure for the performance of mutual funds proposed in (Sharpe 1966). Recently the stock market has been gaining popularity for individual investors. I hope the application could provide the ***query service for single stock*** first, and develop the extensive service in the future work.

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

where:

- R_p is the exptected return on the asset or portfolio.
- R_f is the risk-free rate of return.
- σ_p is the risk (the standard deviation of returns) of the asset or portfolio.

In practice, I use the corresponding index as the risk-free baseline, and the historical return as a substituition of the expected return.

2. Data Acquirement

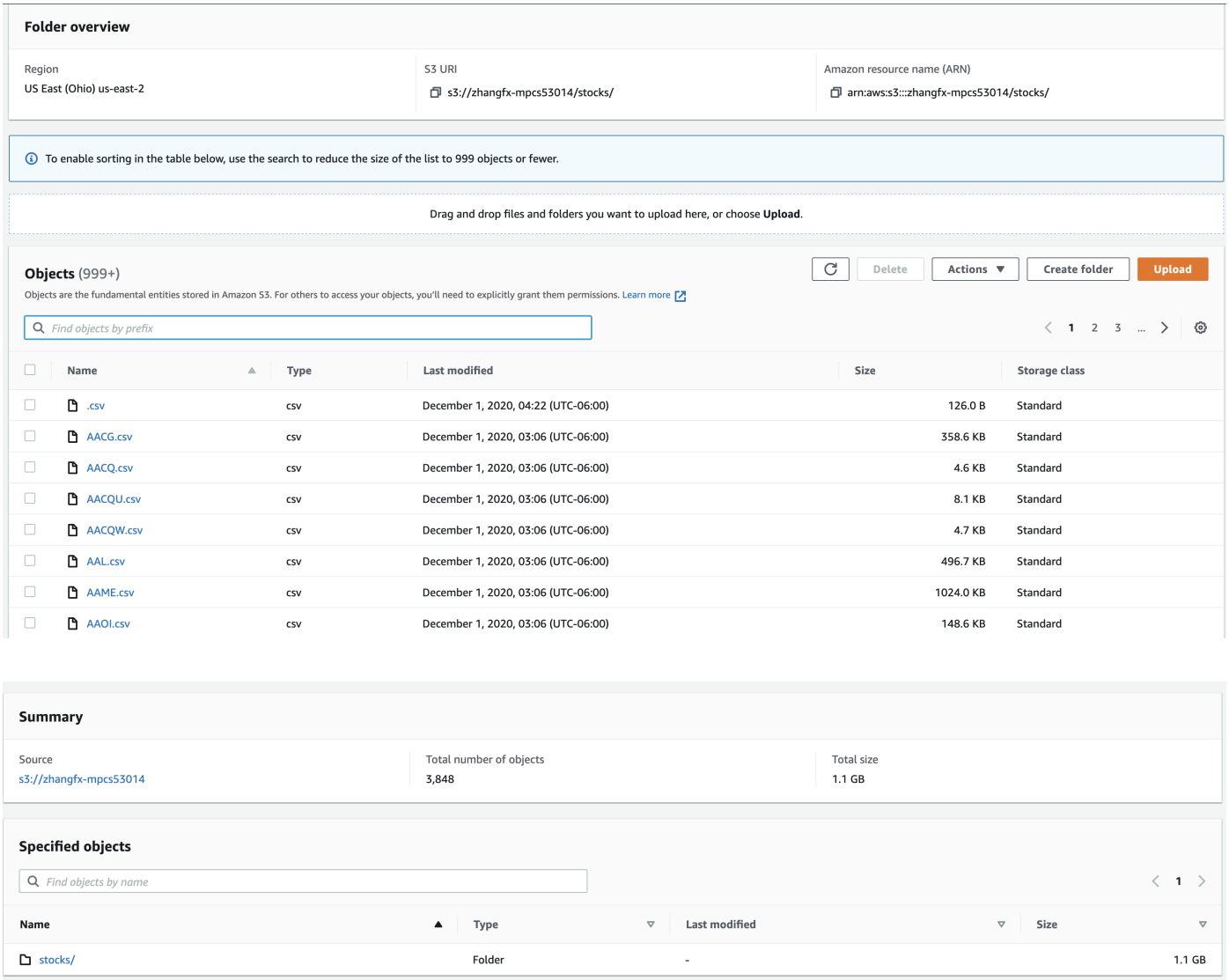
- I downloaded the list of all stocks from NASDAQ official FTP Directory:

  `ftp://ftp.nasdaqtrader.com/symboldirectory.`

  where two files *nasdaqlisted.txt* and *otherlisted.txt* contain the entire list of tradeable symbols. The list is updated on a daily basis.
- I used the Quandl Student Access to obtain all the needed end-of-day (EOD) price for each stock. On the name node, I conducted the following commands to establish the necessary environment and acquire the dataset.

```
# enter personal folder
cd /home/hadoop/zhangfx
# create virtual env
python –m venv ./
# activate python vritual environment
source ./bin/activate
# install quandl
pip3 install quandl
# fetch stock list from nasdaq ftp server to name node
wget ftp://ftp.nasdaqtrader.com/symboldirectory/nasdaqlisted.txt
# execute the python script to load latest data from quandle (private token required
python3 data_ingestion.py –s3 ––token <quandl token>
```

The total size for all nasdaq EOD historical data is about 1.1GB. Here shows a screen shot of the data stored on S3.



- I used the following hive command to create a hive table to manage all csv files stored in S3 ***zhangfx_final*** and ***zhangfx_final_index***:

```
DROP TABLE IF EXISTS zhangfx_final;
CREATE EXTERNAL TABLE zhangfx_final (trade_day DATE,
    open float, high float, low float,
    close float, volume float, dividend float, split float,
    adj_Open float, adj_High float, adj_Low float, adj_Close float, adj_Volume float
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
LOCATION 's3://zhangfx-mpcs53014/stocks/'
TBLPROPERTIES ("skip.header.line.count"="1");

DROP TABLE IF EXISTS zhangfx_final_index;
CREATE EXTERNAL TABLE zhangfx_final_index (trade_day Date,
    Index_Value float, High float, Low float,
    Total_Market_Value float, Dividend_Market_Value float)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
LOCATION 's3://zhangfx-mpcs53014/indices/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

2. Batch Layer & Serving Layer

First, create a new table in hbase.

```
# one table for both index and stocks
create 'zhangfx_final_summary', 'result'
```

Second, use hive to extract all EOD of stocks and index value of index. Use the input_file_name + date as key.

```
-- create external table
create  table zhangfx_final_summary_test (
    stock_name      string,
    num_days        bigint,
    value_avg       float,
    value_std       float,
    start_day_index float,
    end_day_index   float,
    start_day_stock float,
    end_day_stock   float
    );

create external table zhangfx_final_summary (
    stock_name      string,
    num_days        bigint,
    value_avg       double,
    value_std       double,
    start_day_index double,
    end_day_index   double,
    start_day_stock double,
    end_day_stock   double
    ) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,
            result:num_days,
            result:value_avg,
            result:value_std,
            result:start_day_index,
            result:end_day_index,
            result:start_day_stock,
            result:end_day_stock
            ')
TBLPROPERTIES ('hbase.table.name' = 'zhangfx_final_summary');

-- create intermediate views
create table zhangfx_final_view (
    stock_name      string,
    trade_day       Date,
    value_of_day    float,
    index_of_day    float
);

create table zhangfx_final_view2(
    stock_name      string,
    num_days        bigint,
    start_day       Date,
    end_day         date,
    value_avg       double,
    value_std       double
);

-- insert data from stocks
```

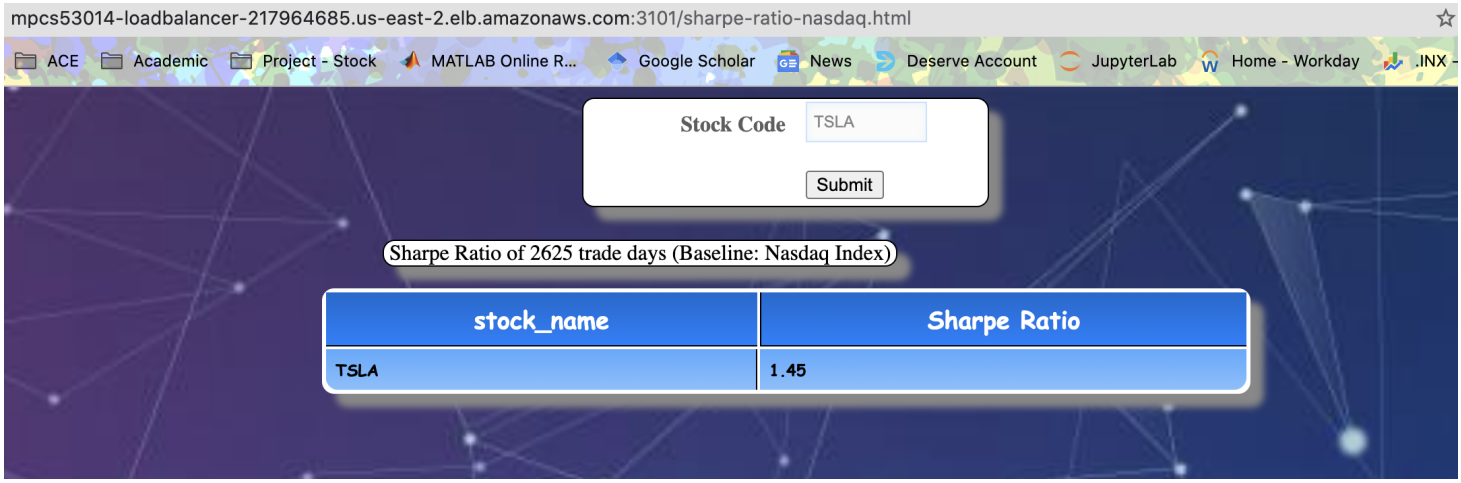
```
insert overwrite table zhangfx_final_view
    select split(split( zhangfx_final.INPUT__FILE__NAME, '/')[4],'[.]')[0] as stock_name
        zhangfx_final.trade_day as trade_day,
        zhangfx_final.adj_close as value_of_day,
        zhangfx_final_index.Index_Value as index_of_day
    from zhangfx_final join zhangfx_final_index on zhangfx_final.trade_day = zhangfx_fi
    where zhangfx_final.adj_Close != '' and zhangfx_final_index.Index_Value != '';

insert overwrite table zhangfx_final_view2
   select stock_name, count(trade_day) as num_days,
        min(trade_day) as start_day, max(trade_day) as end_day,
        avg(value_of_day) as value_avg, std(value_of_day) as value_std
        from zhangfx_final_view group by stock_name;

-- insert batch view into hbase
insert into table zhangfx_final_summary
    select a.stock_name as stock_name, c.num_days as num_days,
    c.value_avg as value_avg, c.value_std as value_std,
    a.index_of_day as start_day_index, b.index_of_day as end_day_index,
    a.value_of_day as start_day_stock, b.value_of_day as end_day_stock

    from zhangfx_final_view as a, zhangfx_final_view as b, zhangfx_final_view2 as c
    where (a.trade_day = c.start_day and b.trade_day = c.end_day and a.stock_name = c.s

select count(distinct stock_name) from zhangfx_final_summary;
```

3. Web Application

I used port 3101 to deploy this new application. The application is successfully deployed on the loadbalancer.

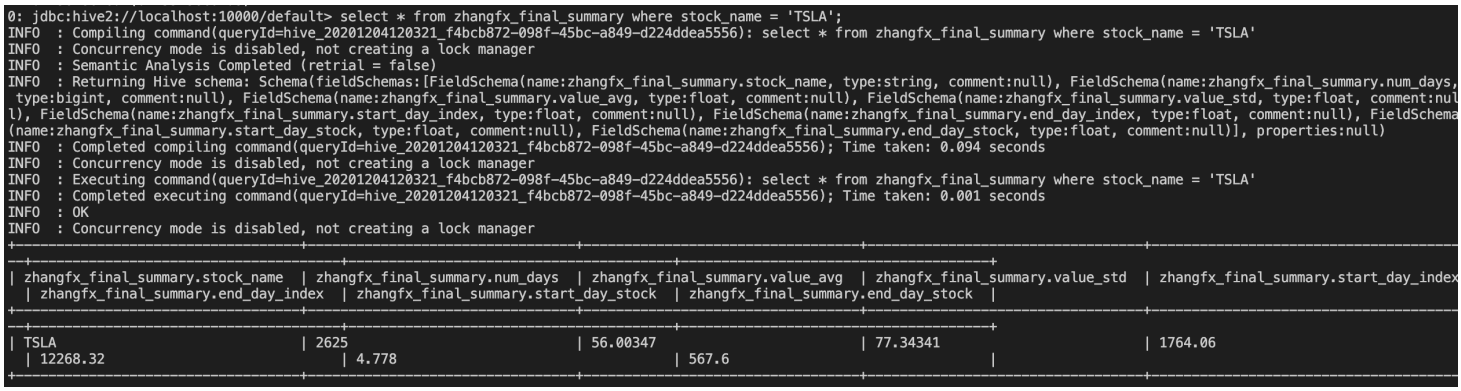
Here is the link to the application: loadbalancer.

The deployment success is shown below:



Users are allowed to enter the nasdaq stock code to query the sharpe ratio. All 3832 stocks in Nasdaq are included in the database.

Here also attached the screenshot of querying TSLA in hive:



```
0: jdbc:hive2://localhost:10000/default> select * from zhangfx_final_summary where stock_name = 'TSLA';
INFO  : Compiling command(queryId=hive_20201204120321_f4bcb872-098f-45bc-a849-d224ddea5556): select * from zhangfx_final_summary where stock_name = 'TSLA'
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:zhangfx_final_summary.stock_name, type:string, comment:null), FieldSchema(name:zhangfx_final_summary.num_days,
 type:bigint, comment:null), FieldSchema(name:zhangfx_final_summary.value_avg, type:float, comment:null), FieldSchema(name:zhangfx_final_summary.value_std, type:float, comment:nul
l), FieldSchema(name:zhangfx_final_summary.start_day_index, type:float, comment:null), FieldSchema(name:zhangfx_final_summary.end_day_index, type:float, comment:null), FieldSchema
(name:zhangfx_final_summary.start_day_stock, type:float, comment:null), FieldSchema(name:zhangfx_final_summary.end_day_stock, type:float, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hive_20201204120321_f4bcb872-098f-45bc-a849-d224ddea5556); Time taken: 0.094 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=hive_20201204120321_f4bcb872-098f-45bc-a849-d224ddea5556): select * from zhangfx_final_summary where stock_name = 'TSLA'
INFO  : Completed executing command(queryId=hive_20201204120321_f4bcb872-098f-45bc-a849-d224ddea5556); Time taken: 0.001 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------
--+---------------------------------+---------------------------------+---------------------------------+
| zhangfx_final_summary.stock_name  | zhangfx_final_summary.num_days  | zhangfx_final_summary.value_avg  | zhangfx_final_summary.value_std  | zhangfx_final_summary.start_day_index
  | zhangfx_final_summary.end_day_index  | zhangfx_final_summary.start_day_stock  | zhangfx_final_summary.end_day_stock  |
+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------
--+---------------------------------+---------------------------------+---------------------------------+
| TSLA                            | 2625                            | 56.00347                        | 77.34341                        | 1764.06
  | 12268.32                        | 4.778                           | 567.6                           |
+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------
--+---------------------------------+---------------------------------+---------------------------------+
```

The application files are included in the ./sharpeRatio/src directory.

- ***result.mustache*** provides the template for return results.
- ***app.js*** includes main functionalities. Read data from HBase and calculate sharpe ratio.
- ***public*** folder include all resources.
- ***package.json*** includes the dependencies.

Also the deployment zip file uploaded to s3 is ***sharpeRatio.zip***.

4. Speed Layer

1) Kafka

I've created zhangfx_mpcs53014 topic in Kafka for the speed layer.

```
# create topic with replication and no partition
/home/hadoop/kafka_2.12-2.2.1/bin/kafka-topics.sh --create --zookeeper z-2.mpcs53014-ka

# check topic
/home/hadoop/kafka_2.12-2.2.1/bin/kafka-topics.sh --list --zookeeper z-2.mpcs53014-kafk

# Install kafka dependencies for python (in the virtual env zhangfx)
pip3 install kafka-python

# Install Hbase for python
pip3 install hbase-python
```

2) Producer

Using the ***stockStream.py*** script, I fetch the stock data and push then into the kafka queue on a daily basis.

```
# enter personal directory
cd /home/hadoop/zhangfx
# venv activate
source ./bin/activate
# fetch and push to kafka
python3 stockStream.py --token <quandl_token>
```

The following shows a sample result.

```
(zhangfx) [hadoop@ip-172-31-11-144 zhangfx]$ python3 stockStream.py --num 2 --token [redacted]
Accept token  R2BtPJDmkwx6Tsc_ox_S Num: 2
Start connecting!
Finish Reading 3 stocks
Message 'AACG: 1.17' published successfully in topic zhangfx_mpcs53014.
Message 'AACQ: 9.9' published successfully in topic zhangfx_mpcs53014.
Message 'NDX: 12467.13' published successfully in topic zhangfx_mpcs53014.
```

3) Consumer

The consumer side automatically obtain stream from kafka and update the hbase (view) accordingly.

```
# run the consumer side
spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home
```

5. Architecture Reasoning

- I've used ***S3 object storage service*** because the data size is ***huge*** (above 1GB) while it is not frequently accessed. Since the application only uses the end-of-day prices, it only needs to update the batch layer ***on a daily basis***.
- I've used ***HBase*** for the serving layer. All the attributes used in the table in HBase minimize the space occupation while allowing an approximate update in the speed layer. The first and last days' prices enable the speed layer to update the historical return every day. The average price, the number of days counted, and the historical data's standard deviation allows the speed layer to update the standard deviation without maintaining the complete history at the cost of slight accuracy loss.