THE UNIVERSITY of NEW ORLEANS

# CSCI 4452/4452 CLOUD COMPUTING

Fall 2021

Assigned: Nov 1

Credits: 20 course points

**Due: NOV 15 @ 11:59PM**

# *Lab-1*

## Objectives & Educational Outcomes

The goal of this assignment is to produce a working data processing pipeline that employs multiple AWS services – S3, SNS and Lambda. See next page for a detailed description of the problem.

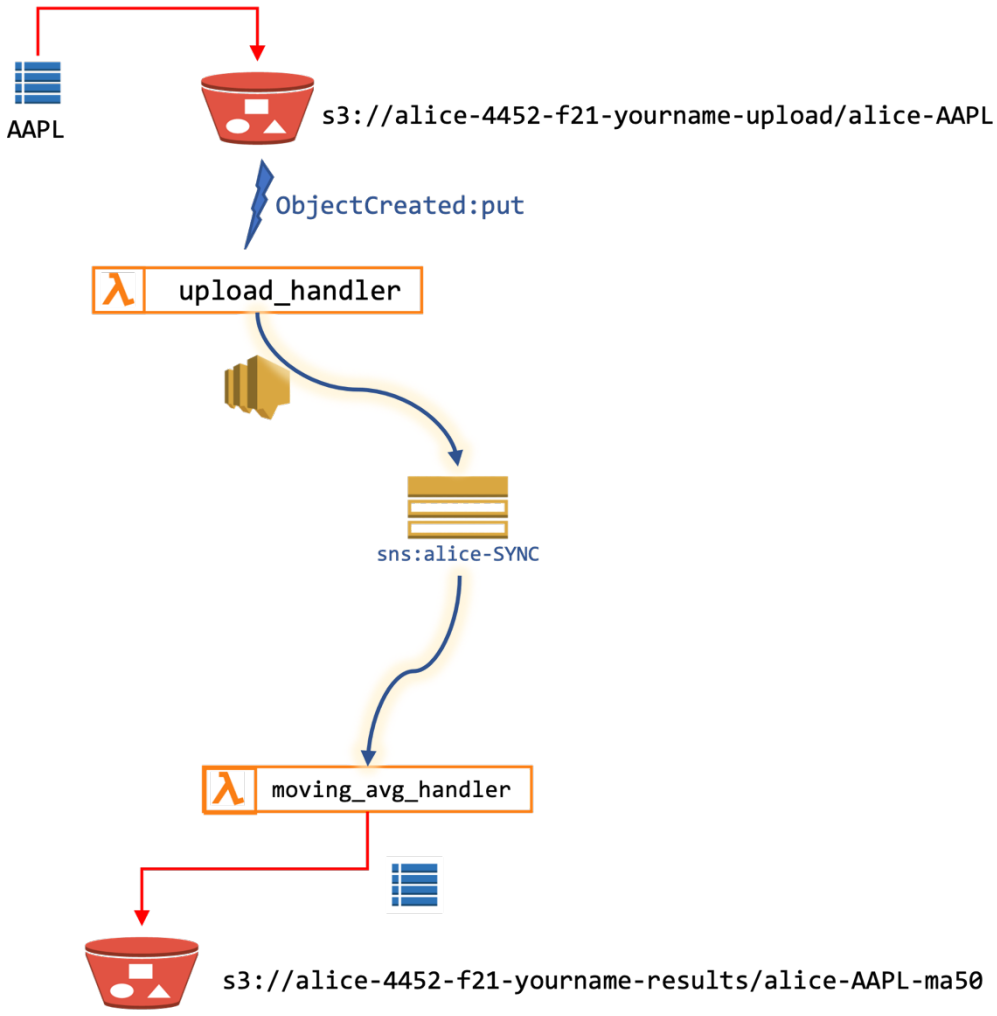
## Notes

- *Permissions*: every data exchange requires appropriate permissions, e.g., *upload_handler* lambda function must be authorized to access bucket and send SNS messages, *moving_avg_handler* must be authorized to receive message and write to bucket, etc.
- *Logs:* from the console, use *AWS CloudWatch>>Logs>>Log Groups* to monitor execution errors. You may need to enable/authorize logging – in SNS, you need to set up *Delivery Status Logging* for each topic from the console.
- *Input Files:* sample input files are available on Moodle.

## Operational Description

The overall architecture of the required system is given on the figure below.

Note that all S3/SNS/SQS resource names have a common prefix (e.g., `alice-`); this is needed to ensure that instantiations are not confused during testing. It is also convenient to be able to create a new instance of the architecture since some of the resources cannot be used immediately after deallocation. In the rest of the document, we will assume this prefix to be `alice-`. However, **your script should be able to take the desired prefix as an input and set up the entire system accordingly.**




The data input consists of historical prices of individual stocks (in csv format); the corresponding output is one, or more, data files with computed moving/rolling averages. On the diagram, the original file is named after the stock ticker (e.g., AAPL), which is uploaded to an S3 bucket (`alice-4452-f21-yourname-upload`).

The input data file has the following format:

```
<TICKER>,<PER>,<DATE>,<TIME>,<OPEN>,<HIGH>,<LOW>,<CLOSE>,<VOL>,<OPENINT>

AAPL,5,20200812,153000,110.5,111.17,110.4,110.64,5684400,0
AAPL,5,20200812,153500,110.65,110.91,110.48,110.51,2604600,0
AAPL,5,20200812,154000,110.56,110.84,110.3,110.33,1538536,0
...
```

The meaning of each field is as follows: 1) stock ticker (unique identifier); 2) period (in min); 3) date; 4) time (GMT), 5) open price (at the beginning of the period); 6) high(est) price for the period; 7) low(est) price; 8) closing price (price at the end of the period); 9) volume (number of shares trade for the period); 10) open interest (N/A, ignore it).

Once a file is uploaded to the designated S3 bucket, an *ObjectCreated:Put* event is triggered, which invokes a lambda function (handler) with a description of the newly created object.

The handler downloads the object (file), parses it and, for each line of data (except the last line), generates and sends a SNS message on a topic determined by the application instantiation prefix e.g., `alice-sns`. In turn, the message is delivered to another lambda function, `moving_avg_handler`, is a subscriber. In other words, if there are 100 stock quotes in the uploaded file, then 100 messages will be delivered via SNS.

After all lines (except the last line) of the CSV file have been processed, the `upload_handler` parses the last line and sends a special message to `alice-sns`. The last line encodes the requested simple moving average (SMA) computation: *ticker*, *period*, *quote* (e.g. `<AAPL, 10, close>`). The *price_quote* parameter specifies over which price value the average is to be computed—open, high, low, close—with the latter being the common case.

After all these lines are sent to `alice-sns`, the `moving_avg_handler` will receive and process the messages. Note that the `moving_avg_handler` can cache things in memory in Lambda provided that you are limiting the concurrency to 1. This handler should finally output a file to a separate results bucket named `alice-4452-f21-yourname-upload.`

The output format should be as follows:

```
<TICKER>,<PER>,<AVG_TYPE>,<QUOTE>,<DATE>,<TIME>,<AVG>,<VOL>

AAPL, 5, SMA10, CLOSE, 20200812, 162000, 110.57, 5124417
AAPL, 5, SMA10, CLOSE, 20200812, 162500, 110.65, 2204400
AAPL, 5, SMA10, CLOSE, 20200812, 163000, 110.45, 1839472
…
```

Note that the AVG_TYPE value above is SMA10 as the *period* was given as 10 i.e. this value was obtained by averaging the 10 past values.

In general, a simple moving/rolling average with a period of *n* is the average of the last *n* values. Clearly, the output will have *n*-1 fewer lines than the original input.

The <DATE>/<TIME> reflect the last data point (the timestamp of the $n^{th}$ sample). The main SMA average for the price should be in the <AVG> field. The <VOL> value is the volume average over the same time period.

## Implementation & Submission

- You are free to choose any language/combination of languages that you feel comfortable with.
- Make your entire deployment fully automated; in other words, the entire application can be fully setup and run from the command line. This may involve executing several components/scripts, but the complete solution should require *no* interaction with the AWS Console.

- Your submission should include sufficient documentation for your code. Please add a read-me file that briefly explains the structure of your code. Also, try to add comments to your code describing the important steps.
- **<u>IMPORTANT - Zoom recording requirement</u>:**

  Your submission should also include a Zoom-based screen recording of the system that you have setup. You can this by starting your own meeting, sharing your screen and recording to the cloud. Please include an HTTP link to this recording in your submission write-up mentioned above. The Zoom recording should cover the following tasks at least to demonstrate that you have set up the system correctly:

1. You should start a fresh instance of the AWS Educate Sandbox.
2. You should now show the dashboard of your AWS S3 which lists all the buckets.
3. Next, you should update your AWS Educate credentials on your machine in preparation for your code invocation.
4. You should invoke your code from the command line supplying a never-before-used *project-prefix* as per your choice as an argument.
5. You should show that two AWS buckets with the chosen *project-prefix* are created (one for upload and one for results).
6. You should now upload two sample ticker files into your bucket for **Tesla** and **Apple** stocks. These are available on Moodle with the assignment.
7. You should next go to CloudWatch logs and show the logs being produced by the two lambda functions.
8. After your system process both the input ticker files, go to the dedicated AWS results bucket that has been created by your setup script and show the output files there..

- **Deliverables**: Please make sure to include all **code**, **documentation** (i.e. the above-mentioned read-me file which should include a link to the Zoom video), **all logs produced by the two Lambda functions** in your recordings and **also both the output files (for AAPL and TSLA)** that were uploaded to the results directory by your system in the Zoom recording.
- **Submission**: Create a **private repository** named `csci4452-f21` on `gitlab.cs.uno.edu` (use the UNO Active Directory Login). Please add me as a collaborator with at least **Developer** access. My e-mail address for gitlab is **kkvadrev@uno.edu**
    - Make sure that the repository is named **exactly** as specified (this includes 5452 students)
    - Create a folder in your repo called `lab-1` and place all your deliverables there.
- **The submission deadline is a hard one. Any late submissions will be penalized at 5% per day.**
- This is an individual take-home assignment. You may passively consult any available sources on the Internet—that is, you can read anything but you cannot solicit advise in any form. **All work must by yours only.**