Streaming Finance Data with AWS Lambda

For this project, you are tasked with provisioning a few Lambda functions to generate near real time finance data records for downstream processing and interactive querying.

[**Requirements**](#_nbl730cyehkj) **2**

[**Assignment**](#_9wait567xq9y) **2**

[Infrastructure](#_isf8j84tjymn) 2

[Data Collection](#_yq31cayzy5fw) 3

[**IMPORTANT CLARIFICATION**](#_l65jmio1o8um) **4**

[Does this suck? Yes.](#_p51kmid9mjnd) 5

[Data Analysis](#_280z5q5tbur) 6

[**Artifacts**](#_gcvq0fmil4i) **6**

[Lambda Source Code For DataCollector](#_kt313cs57j61) 6

[S3 Content From DataTransformer](#_3uz57vj1h1ay) 7

[CSV Output From Query in DataAnalyzer](#_w0y8r7beg7ji) 7

[README](#_v5u4xui65tlj) 7

[**Extra Credit**](#_hlu6cuqexk45) **7**

[**Submission**](#_t90ju1mwhvz1) **7**

[**Rubric**](#_a8h16kdfssyq) **7**

[**Appendix A**](#_xcezytrm8149) **8**

[**Appendix B**](#_9bxy79fpdqks) **9**

# 

# 

# 

# 

# Requirements

This project leads you through the process of consuming “real time” data, processing the data and then dumping it in a manner that facilitates querying and further analysis, either in real time or near real time capacity.

In doing so, you will familiarize yourself with a process that you can leverage in your professional or personal endeavors that require consumption of data that is “always on” and changing very quickly, in sub hour (and typically) sub minute intervals.

This project is due **MONDAY MAY 25TH, MIDNIGHT**. **NO EXCEPTIONS.**

**PLEASE READ THE FOLLOWING INSTRUCTIONS CAREFULLY**. I will likely be automating some or most of the grading process and therefore your output artifact must match the spec I define below.

# Assignment

This assignment is broken into three parts:

* Infrastructure
* Data collection
* Data analysis

## Infrastructure

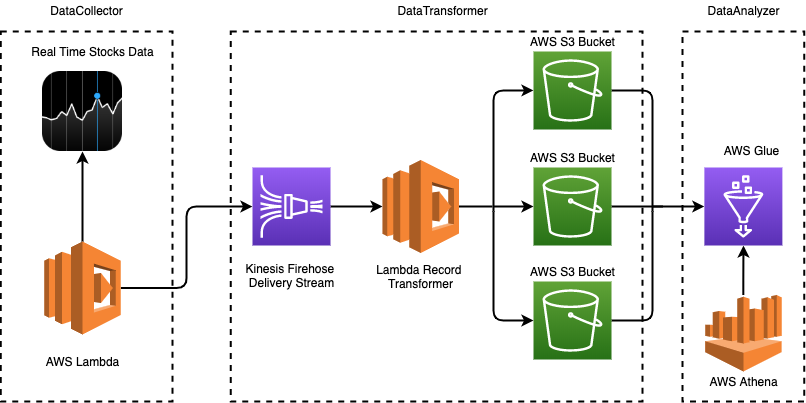
This project consists of three major infrastructure elements that work in tandem:

1. A lambda function that collects our data (**DataCollector**)
2. A lambda function that transforms and places data into S3 (**DataTransformer**)
3. A serverless process that allows us to query our s3 data (**DataAnalyzer**)

First, you will create a **Kinesis Firehose Delivery Stream** like we did in Lecture 12. This stream should also have a lambda function that transforms your record and streams it into an S3 bucket. This is what I am calling the **DataTransformer**.

Then, you will write another **Lambda function** that is triggered from a simple URL call, just like we configured in Lecture 11. On trigger, it will grab stock price data and place it into the delivery defined in the **DataTransformer**. This is what I am calling the **DataCollector**.

Finally, configure **AWS Glue**, pointing it to the **S3 Bucket** you created in your **DataTransformer.** This will allow us to now interactively query the **S3** files generated by the **DataTransformer** using **AWS** Athena to gain insight into our streamed data. This is what I am calling the **DataAnalyzer**.



As you can see from this diagram, on the left side we have a Lambda function that can programmatically pull data in from some internet source (more on that in the next section). On the right side we have AWS Athena, which allows us to write ad-hoc, interactive queries against the data we have accumulated to gain various types of insights.

If our left hand, DataCollector lambda were to run every hour, we would have a system that manages and stores realtime data and allows querying of that data in near real time.

## Data Collection

In our collector lambda, using the [yfinance](https://pypi.org/project/yfinance/) module ([documentation](https://github.com/ranaroussi/yfinance) here), you will grab pricing information for each of the following stocks:

* Facebook (FB)
* Shopify (SHOP)
* Beyond Meat (BYND)
* Netflix (NFLX)
* Pinterest (PINS)
* Square (SQ)
* The Trade Desk (TTD)
* Okta (OKTA)
* Snap (SNAP)
* Datadog (DDOG)

You are to collect **one full day’s worth of stock HIGH and LOW prices** for each company listed above **on Thursday, May 14th 2020**, at an **one minute interval**. Note that by “full day” we mean one day of stock trading, which **is not 24 hours**.

You have two options - you can use the `history` function and get a previous day’s data OR you can attempt to collect this data “in real time” (of course this would require usage of the **Cloudwatch Events** trigger on Lambda).

Using the `history` function is definitely the easier approach and I would recommend starting there. (You just create a Lambda as seen in Lecture 12 and fire it with a simple browser call)

For each datapoint, generate a JSON object that looks like so:

|  |
| --- |
| {  "high": 67.5,  "low": 64.61,  "ts": "2020-05-13 09:30:00-04:00",  "name": "DDOG"  } |

This is an example of a single “record” that you would place into the firehose stream defined in DataTransformer.

# IMPORTANT CLARIFICATION

So, it is worth noting that this lambda will require an additional dependency requirement, the y**finance** module. Now, the UI provided by AWS allows you to write the simplest lambas possible, meaning NO dependencies are supported. In order to write a lambda function with pip requirements as we are used to, you **must** create and upload what is called a **deployment artifact**. Creating such an artifact requires working in the **terminal**, which is fine but for the sake of simplicity I wanted to avoid.

For this reason I will be providing a work-around to ensure that your lambda can be written **without** having to create and upload a deployment package via the terminal. In order to make your project work with just the AWS lambda homepage provided code editor, please copy and paste the snippet of code below into your **DataCollector** lambda function. (**NOT the DataTransformer Lambda).**

|  |
| --- |
| **import boto3 import os import subprocess import sys  subprocess.check\_call([sys.executable, "-m", "pip", "install", "--target", "/tmp", 'yfinance']) sys.path.append('/tmp') import yfinance   def lambda\_handler(event, context):  pass** |

Do you see the **`pass`** line? Replace that with your actual code for downloading **yfinance** historical data and producing to your Kinesis stream.

**Another major caveat**: this will **NOT** work with the API Gateway invocation. Why not? Because downloading this dependency takes some time, too long for API Gateway to stay “on”.

Instead, do the following:

* Increase your Lambda timeout in the **Basic Settings** section of your Lambda homepage to **15minutes**.
* Instead of loading your lambda by the URL provided, click on the **Test** button on the top right corner of the page and run your lambda that way (input doesn’t matter because this function doesn’t care about inputs).
* Use **Cloudwatch Logs** and your **S3** file (from Kinesis) to validate that your lambda works.

### **Does this suck? Yes.**

How can this be improved? Many ways, in fact. The first and foremost is via the deployment package, where you *don't* have to install the **yfinance** module via subprocess. You can actually create your dependency package via docker (which is a big reason why docker is so useful, you can use it for anything!). [**Here is a repo**](https://github.com/mottaquikarim/STA9760_simple_deployment_package) I put together demonstrating this, I won’t talk further about it (in a formal lecture) but feel free to ask questions / clarifications.

The next improvement is to acknowledge that running this lambda via API Gateway is kind of stupid. It works, but it is not really the way we want to run this code. Instead, it would be far better in this case to set up a Cloudwatch Event trigger that runs this lambda once/minute or 10 mins. At each invocation, the lambda should get stock data for the last minute/10mins (depending on the CloudWatch Event frequency you configured) and push that to the Kinesis stream. This way, you have a **true** real time pipeline that always takes stock data and pushes it to a place where it can be easily analyzed or otherwise used for a plethora of other usecases.

**YOU DO NOT HAVE TO ENCODE THESE APPROACHES TO YOUR PROJECT**.

I felt compelled to list them for the sake of your education as we lack the time to cover these in class, unfortunately. After the completion of this class, I would recommend playing around with Lambda technologies, they are quite powerful and more importantly **useful** for a myriad of use cases.

## Data Analysis

We want to prep this data gathered for analysis! To do so, set up a Glue crawler so that you can run AWS Athena queries against your data. Then, in Athena, write and run a query that gives us the highest hourly stock “high” per company from the list above.

# Artifacts

You are to create a Github Repo with a well formatted title (not something like “Project II”). The Github repo must contain the following:

## Lambda Source Code For DataCollector

I’d like the source code for your lambda function that collects the data from yfinance and puts into the firehose stream.

You **must** name your lambda function file **data\_collector.py**. Please place this in your project root folder.

Also, in your **README**, please provide a screenshot of your AWS Lambda configuration page (see Appendix A for example).

**PLEASE DO NOT COMMIT YOUR AWS ACCESS KEYS OR SECRET KEYS ANYWHERE. IF I SEE THEM IN YOUR GITHUB REPO, I WILL GIVE YOU AN “F”. 🚨🚨**

Yes, this is a really big deal. I’d like for you to manage your keys in environment variables as per Lecture 11.

## S3 Content From DataTransformer

Please create a folder called **finance\_data** in your github repo. This folder should contain all the S3 data that your system collected and stored as part of the **DataTransformer**. (You will have to go into the subfolders and download the file itself).

## CSV Output From Query in DataAnalyzer

Please upload a file called **results.csv** that contains our query output (highest hourly stock “high” per company from the list provided).

Please also include a file called **query.sql** that contains the actual query that you ran to generate your **results.csv** file.

## README

The **README**, in markdown, should contain a brief blurb describing the project and the technology leveraged to conduct your analysis. This ought to be brief and informational, in case folks in the future want to recreate your results.

# Extra Credit

For extra credit, I will accept a **Jupyter Notebook** file that takes your **results.csv** file and generates a few visualizations on the data you accumulated. Of course, feel free to write any additional queries you want to generate interesting graphs and other visualizations. This file must be called **Analysis.ipynb**.

**Upon successful completion of this extra credit assignment, I will boost your lowest project grade up by one letter.** So for example, if you get a “B” in Project 1 and that was your lowest grade, I will boost your Project 1 grade to an “A”.

# Submission

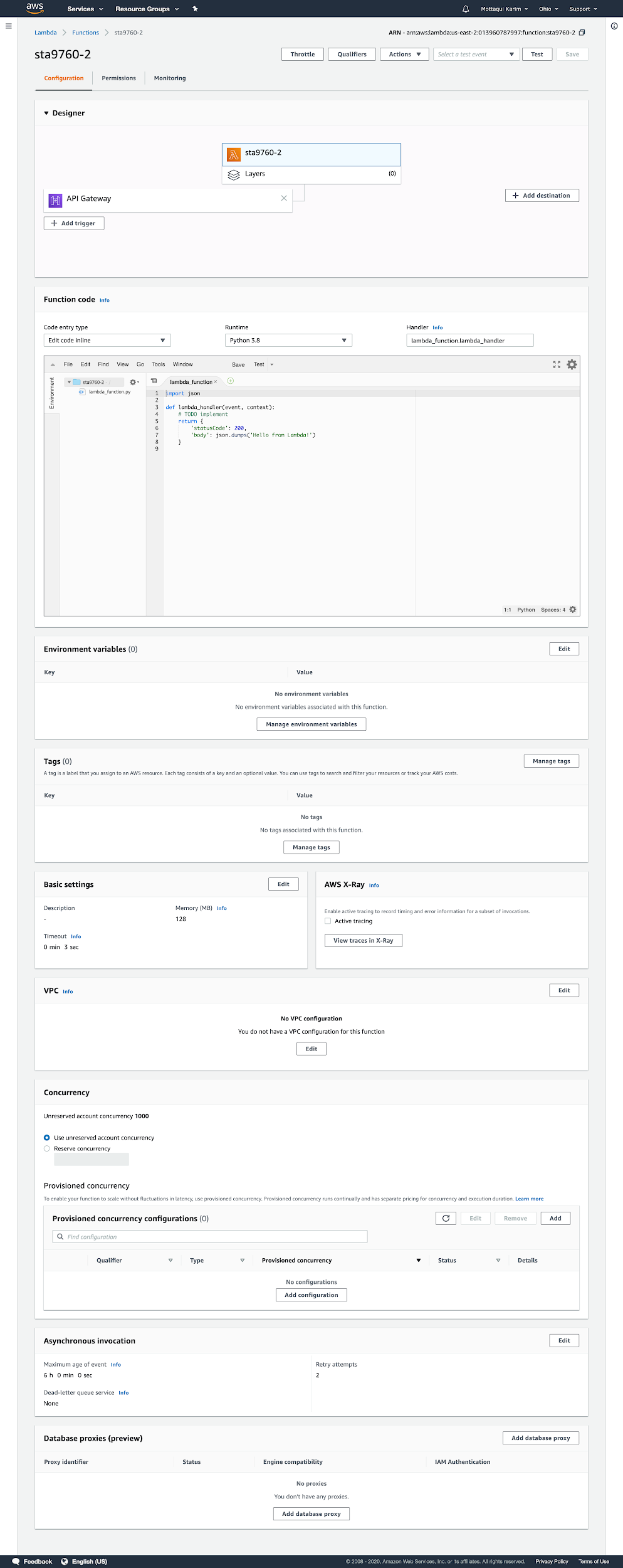
[Please submit your Github Repo URL here.](https://airtable.com/shrP7CiSfR9cTSMAN)

# Rubric

|  |  |
| --- | --- |
| **RUBRIC** | |
| **DataCollector** | **10** |
| Lambda function URL is available in the README document and functional (if I open that link in the browser, your lambda runs). | 1 |
| Lambda function source code imports **yfinance** module and correctly retrieves stock data from companies listed on the day of **May 14th 2020** | 3 |
| Lambda function source code correctly transforms the dataframe from **yfinance** module into JSON in the correct format (specified in this document). | 3 |
| Lambdafunction calls **boto3.client(‘firehose’)** to put records successfully into the firehose delivery stream. | 3 |
| **DataTransformer** | **3** |
| **finance\_data** subfolder is available in Github repo root. | 1 |
| **finance\_data** subfolder contains files holding data collected by DataCollector lambda function | 1 |
| **README** has a screenshot of the kinesis firehose delivery stream “Monitoring” page, showcasing graphs that prove firehose was used (see Appendix B) | 1 |
| **DataAnalyzer** | **7** |
| **results.csv** file is available in Github repo root | 1 |
| **query.sql** file is available in Github repo root | 1 |
| **results**.csv contains data that correctly identifies hourly “high” stock price per company. Expected columns: company name, high stock price, datetime of when high price occurred, hour of da. | 3 |
| **query.sql** file contains a query that has the correct syntax and can generate the output in results.csv if run | 2 |

# Appendix A

DataCollector Lambda configuration page



# 

# 

# 

# 

# 

# 

# Appendix B

Kinesis Data Firehose Delivery Stream Monitoring

