

Setting Up a Database for Financial Data to Predict Stock Prices

INTRODUCTION

There are substantial challenges that the financial industry often encounters in processing and analyzing large volumes of stock data daily. These challenges include limitations in scalability, efficiency, and integration of existing infrastructures. As a result, decision-making processes are often delayed, and opportunities are missed. Our team aims to address these issues by setting up a robust solution capable of managing the volume, variety, and velocity of stock data efficiently in real-time. The dynamic nature of financial markets necessitates the development of powerful and scalable real-time data analysis solutions, so we want to develop a scalable platform to store large volumes of stock data and provide real-time analytics. This project is centered around harnessing the capabilities of the AWS (Amazon Web Services) ecosystem to create a platform that can efficiently manage large volumes of stock data. Through integration with Alpha Vantage to access extensive data sources, the platform is designed to furnish financial institutions, traders, and investors with immediate insights and analytics. This report will discuss our outcomes in detail, including data used, methods, results, visualizations, and our conclusion about stock data.

DATA

The API we chose is Alpha Vantage. Alpha Vantage is an API provider for accessing a wide range of financial data. It is widely utilized by investors, financial developers, and researchers for real-time and historical stock data analysis. Alpha Vantage supports a range of data types including stock prices, forex rates, cryptocurrencies, and over 50 technical indicators. Data from Alpha Vantage can be accessed at multiple frequencies, including real-time, intraday, daily, weekly, and monthly, to meet different analysis needs. This service is especially valued for its simplicity, ease of integration, and scalability.

METHODOLOGY

For this project, we used AWS to handle large volumes of data and did feature engineering, then we used Jupyter to generate visualization and conduct analysis. We chose AWS service because it provides scalability, efficiency, integration, and cost effectiveness. AWS services like Lambda and S3 scale automatically handle large data volumes without manual intervention. The serverless computing and managed services reduce the need to manage infrastructure, allowing focus on business logic and data analysis. Moreover, the seamless integration among AWS services enhances data flow and simplifies architectures. Finally, the pay-as-you-go pricing models of AWS services ensure cost efficiency, particularly important in projects involving large datasets.

Our main architecture consists of AWS Lambda, S3, Glue, Eventbridge, and Sagemaker. S3 acts as our central data lake, stores stock data with high durability and accessibility. Lambda powers our serverless data processing, allowing us to run code in response to triggers without provisioning or managing servers. Glue automates the preparation and loading of data for analysis, serving as a fully managed extract, transform, and load (ETL) service to move data between data stores. EventBridge schedules events, facilitating regular data updates and ensuring our models operate on the latest information. SageMaker streamlines the machine learning workflow, enabling quick model building, training, and development with fully managed infrastructure. Finally, Jupyter provides business intelligence and visualization, training our data into actionable insights through intuitive dashboards and reports. We will provide key steps in detail in the following report.

STEPS TAKEN

1. Data Collection
 - a. S3
 - b. Lambda
 - c. Event Bridge
 - d. AWS Lambda is used to execute a Python script that fetches stock data via an API.
 - e. Amazon S3 serves as the storage destination for the fetched data in CSV format.
 - f. AWS EventBridge schedules the data fetching task to run daily at 12:00 AM, ensuring that the data is consistently updated.
2. Data Cleaning
 - a. AWS Glue
 - b. AWS EventBridge
3. Data Modeling
 - a. AWS SageMaker
 - b. AWS EventBridge

DATA COLLECTION

AWS Lambda, S3 and EventBridge are used for data collection. Below is the script which generates data from API

```
import csv
import requests
import boto3
from io import StringIO
def lambda_handler(event, context):
    # Define the list of stock symbols to process
    symbols = ['AAPL', 'AMZN', 'META', 'NFLX', 'GOOGL', 'NVDA']

    # Define S3 parameters
    bucket_name = 'stockdatacsvfiles'
```

```
# Get the S3 client
s3 = boto3.client('s3')

results = []

# Loop through each symbol to process
for symbol in symbols:

    # Define the URL for the CSV download for each symbol
    url =
f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&outputsize=full&apikey=AL95JY63PIVWK60Q&datatype=csv'

    # Make the HTTP request to download the CSV data
    try:
        response = requests.get(url)
        response.raise_for_status() # Ensure the request succeeded

        # Handle different content types appropriately
        if 'application/x-download' in response.headers.get('Content-Type', '') or 'text/csv' in response.headers.get('Content-Type', ''):

            # File path in S3 for each stock symbol, each in its own subfolder
            file_name = f'path/{symbol.lower()}/{symbol.lower()}.csv'

            # Write the CSV directly to S3 without saving locally
            s3.put_object(Bucket=bucket_name, Key=file_name, Body=response.content,
Content-Type='text/csv')

            results.append(f'CSV file for {symbol} successfully saved to S3 in its own folder.')
        else:
            print(f'Content-Type {response.headers.get("Content-Type", "")} not supported')
    except requests.exceptions.RequestException as e:
        print(f'Error: {e}')
```

```
raise ValueError(f"Unexpected content type for {symbol}:\n{response.headers['Content-Type']}")
```

```
except Exception as e:
```

```
    results.append(f"Error processing {symbol}: {str(e)}")
```

```
return {
```

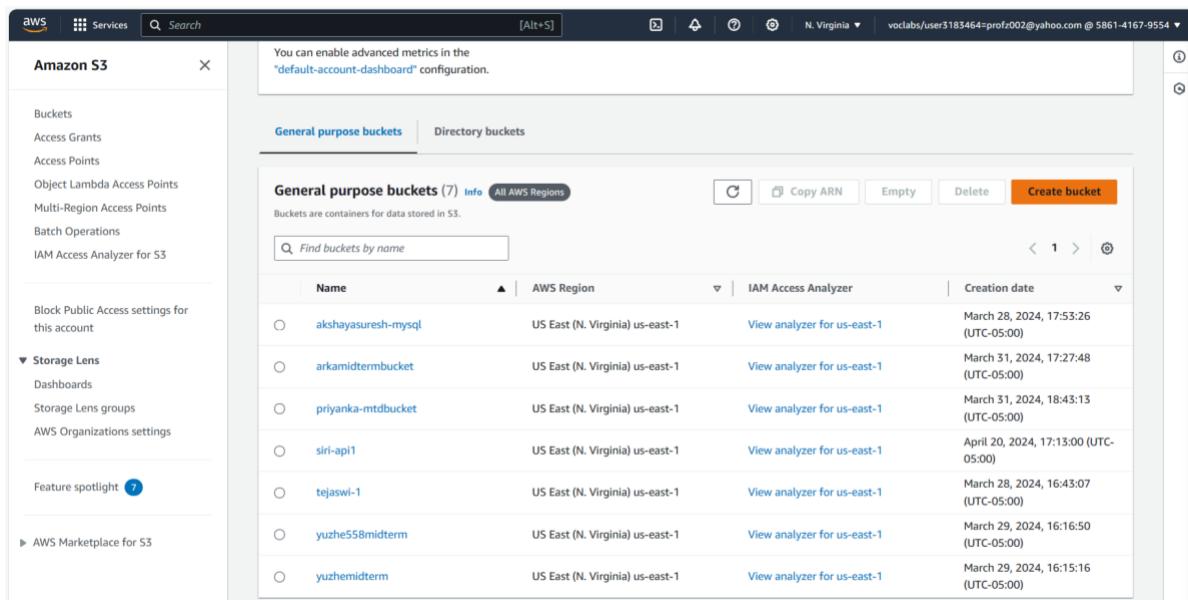
```
    'statusCode': 200,
```

```
    'body': results
```

```
}
```

A. S3

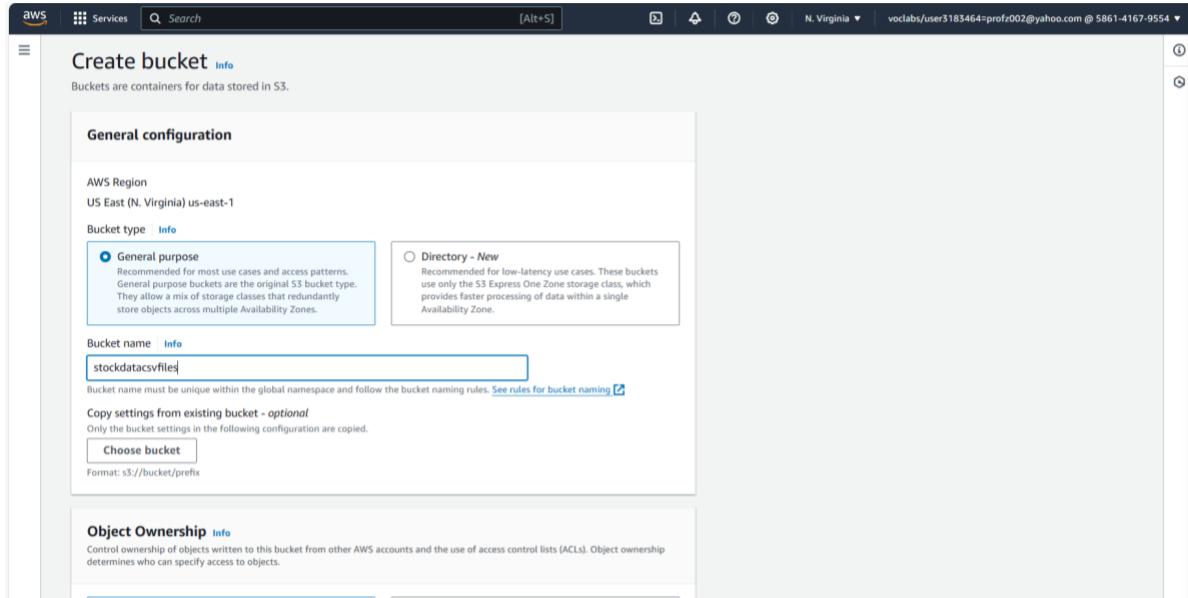
1. Open S3 and create a bucket



The screenshot shows the AWS S3 console interface. On the left, there is a navigation sidebar with links for Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, and IAM Access Analyzer for S3. Below that is a section for Block Public Access settings for this account, followed by Storage Lens (Dashboards, Storage Lens groups, AWS Organizations settings), Feature spotlight, and a link to AWS Marketplace for S3. The main content area is titled "General purpose buckets (7)" and includes a search bar, filter buttons for "All AWS Regions", and a "Create bucket" button. A table lists seven buckets: "akshayasuresh-mysql", "arkamidtermbucket", "priyanka-mtdbucket", "siri-api1", "tejaswi-1", "yuzhe558midterm", and "yuzhemidterm". Each row shows the bucket name, AWS Region (US East (N. Virginia) us-east-1), IAM Access Analyzer (with a "View analyzer" link), and Creation date.

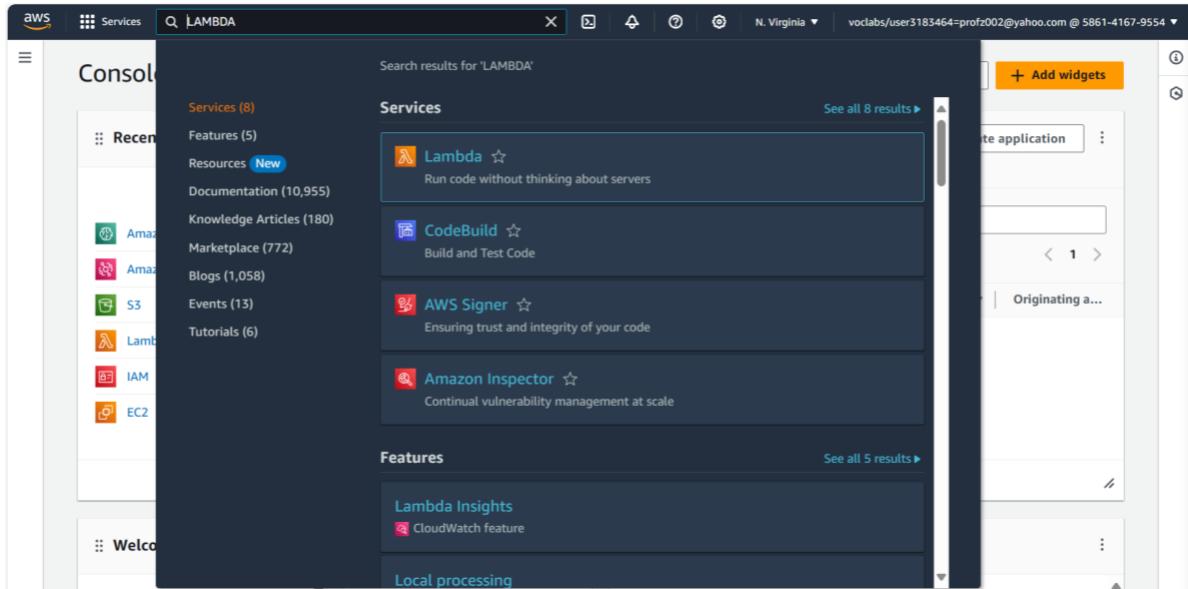
Name	AWS Region	IAM Access Analyzer	Creation date
akshayasuresh-mysql	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 28, 2024, 17:53:26 (UTC-05:00)
arkamidtermbucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 31, 2024, 17:27:48 (UTC-05:00)
priyanka-mtdbucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 31, 2024, 18:43:13 (UTC-05:00)
siri-api1	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 20, 2024, 17:13:00 (UTC-05:00)
tejaswi-1	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 28, 2024, 16:43:07 (UTC-05:00)
yuzhe558midterm	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 29, 2024, 16:16:50 (UTC-05:00)
yuzhemidterm	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 29, 2024, 16:15:16 (UTC-05:00)

2. Give a bucket name



B. LAMBDA

1. Open AWS Lambda



2. Click on create function

The screenshot shows the AWS Lambda Functions page. On the left, there's a sidebar with 'AWS Lambda' at the top, followed by 'Dashboard', 'Applications', 'Functions' (which is selected and highlighted in blue), 'Additional resources', and 'Related AWS resources'. The main area is titled 'Functions (6)' and shows a list of existing functions: 'stockprice', 'RedshiftOverwatch', 'RoleCreationFunction', and 'ModLabRole'. A red box highlights the 'Create function' button at the top right of the list. To the right of the list, there's a sidebar titled 'Tutorials' with a section for 'Create a simple web app'.

3. Give a function name, Select run time (Python 3.10 here) and Select an IAM Role and click on create function

The screenshot shows the 'Basic information' step in the Lambda function creation wizard. It has several sections: 'Function name' (with 'stock' entered), 'Runtime' (set to 'Python 3.10'), 'Architecture' (set to 'x86_64'), 'Permissions' (with 'LabRole' selected as the existing role), and 'Existing role' (also set to 'LabRole'). Red boxes highlight the 'Function name', 'Runtime' dropdown, and the 'Existing role' dropdown. To the right, there's a sidebar titled 'Create a simple web app'.

4. Enter the code in the ‘Code’ section of the function

```

1 import csv
2 import requests
3 import boto3
4 from io import StringIO
5 def lambda_handler(event, context):
6     # Define the list of stock symbols to process
7     symbols = ['AAPL', 'AMZN', 'META', 'NFLX', 'GOOGL', 'NVDA']
8
9     # Define S3 parameters
10    bucket_name = 'stockdatacsvfiles'
11
12    # Get the S3 client
13    s3 = boto3.client('s3')
14
15    results = []
16
17    # Loop through each symbol to process
18    for symbol in symbols:
19        # Define the URL for the CSV download for each symbol
20        url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&outputsize=full&apikey=AL95JY63PIWK60Q&datatype=csv'
21
22        # Make the HTTP request to download the CSV data
23        try:
24            response = requests.get(url)
25            response.raise_for_status() # Ensure the request succeeded
26
27            # Handle different content types appropriately
28            if 'application/x-download' in response.headers.get('Content-Type', '') or 'text/csv' in response.headers.get('Content-Type', ''):
29                # File path in S3 for each stock symbol, each in its own subfolder
30                file_name = f'path/{symbol.lower()}/{symbol.lower()}.csv'
31
32                # Write the CSV directly to S3 without saving locally
33                s3.put_object(Bucket=bucket_name, Key=file_name, Body=response.content, ContentType='text/csv')
34                results.append(f'CSV file for {symbol} successfully saved to S3 in its own folder.')
35            else:
36                raise ValueError(f'Unexpected content type for {symbol}: {response.headers["Content-Type"]}')
37        except Exception as e:
38            results.append(f'Error processing {symbol}: {str(e)}')
39
40    return {
41        'statusCode': 200,
42        'body': results
43    }
44

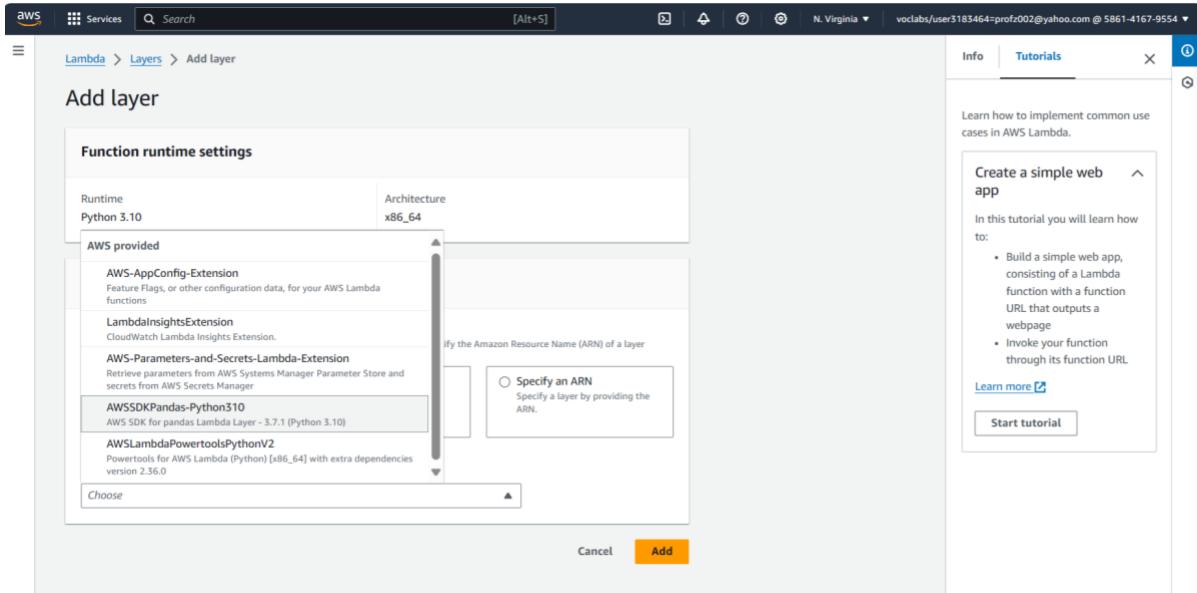
```

5. Click on ‘add a layer’ in layers section

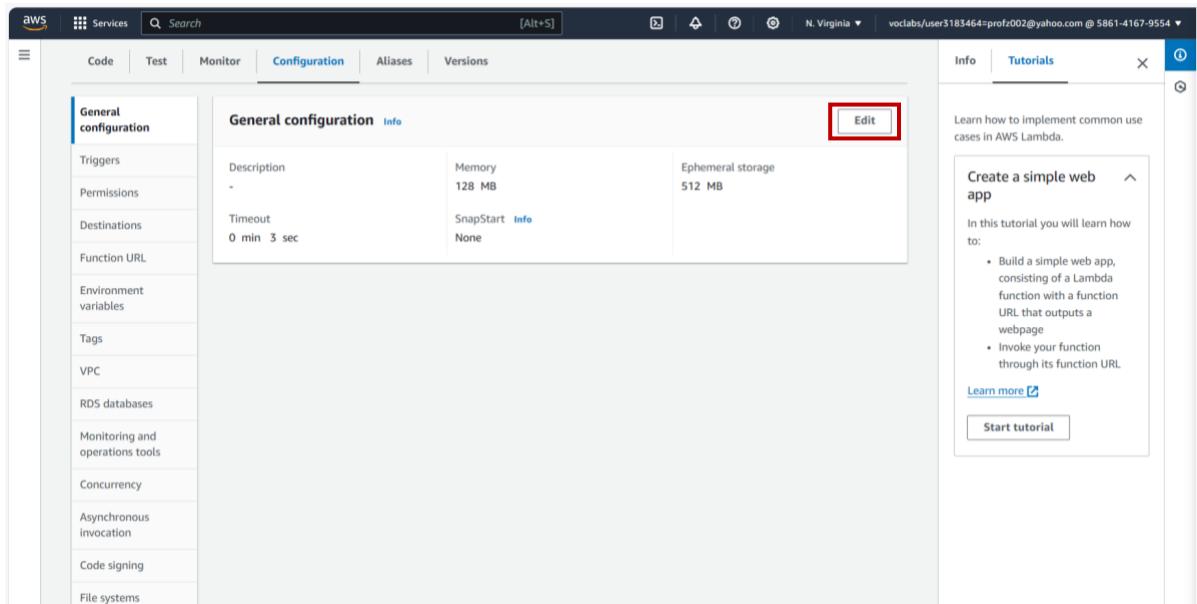
The screenshot shows the AWS Lambda function configuration interface. The top navigation bar includes 'Services', 'Search', and 'N. Virginia'. The main area displays the function's code properties, runtime settings, and layers.

- Code properties:** Shows package size (299.0 byte), SHA256 hash (SHA256 hash: HAPq9EReJVEC5gLavtc/gyd5vZtd9eiUGF932t0JBxY=), and last modified (April 20, 2024 at 05:57 PM CDT).
- Runtime settings:** Shows runtime (Python 3.10), handler (lambda_function.lambda_handler), and architecture (x86_64).
- Layers:** This section is highlighted with a red box around the 'Add a layer' button. It shows a table with columns: Merge order, Name, Layer version, Compatible runtimes, Compatible architectures, and Version ARN. A message indicates 'There is no data to display.'
- Tutorials:** A sidebar titled 'Create a simple web app' provides a brief overview and links to 'Learn more' and 'Start tutorial'.

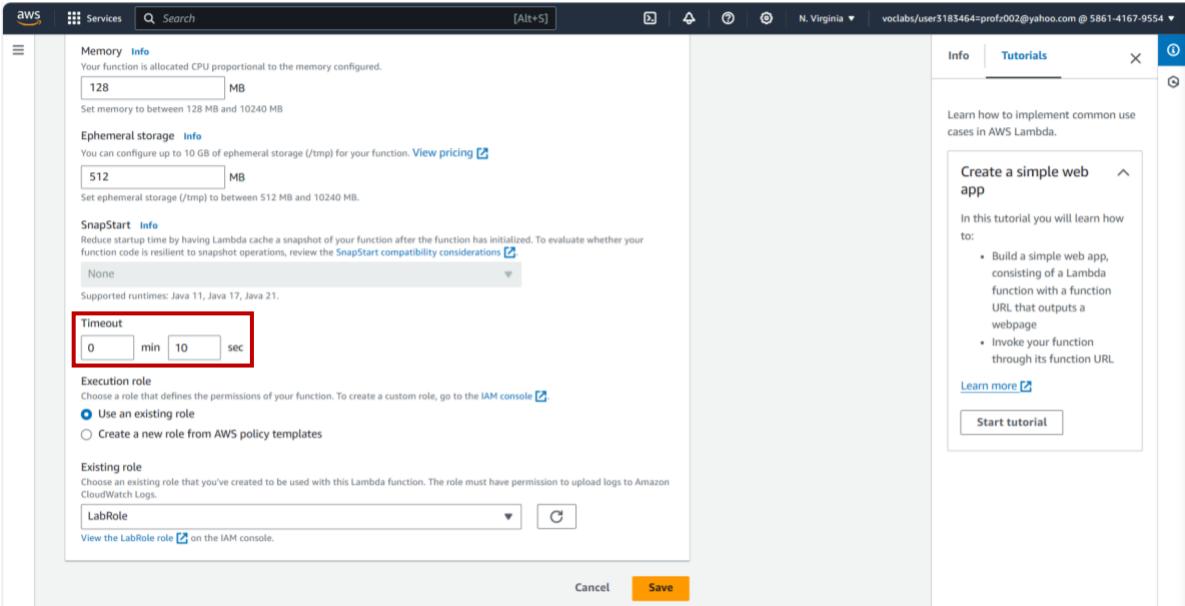
6. Select AWS layer with pandas and other related packages installed and click on ‘add’



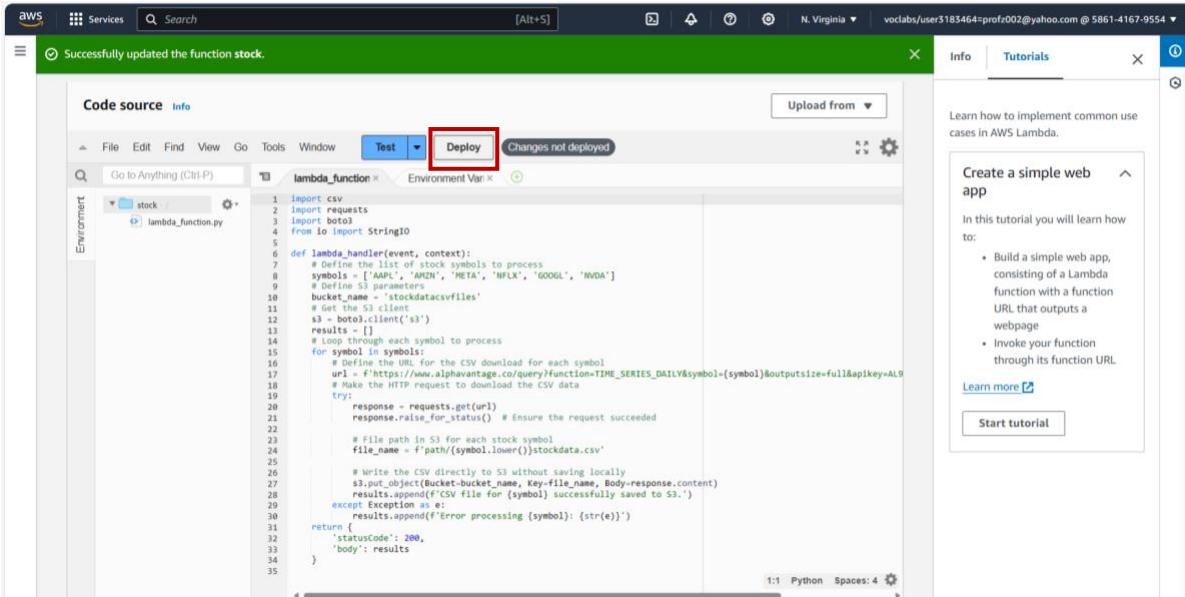
7. Go to ‘Configuration’ and click on Edit



8. Increase the timeout to 10 sec, as the python script takes longer than the default 3 sec.



9. Click on 'Deploy'



DATA CLEANING

A. AWS Glue

Setting up Glue Crawler

1. Open AWS Glue from the main menu.

2. Click on Crawler tab from the left-hand menu on the AWS Glue page.
3. Click create crawler.
4. Give your crawler a name: Here we are Stock_data_Crawler as the name for the crawler.

AWS Glue > Crawlers > Add crawler

Step 1 Set crawler properties

Step 2 Choose data sources and classifiers

Step 3 Configure security settings

Step 4 Set output and scheduling

Step 5 Review and create

Set crawler properties

Crawler details Info

Name
Stock_data_Crawler
Name can be up to 255 characters long. Some character set including control characters are prohibited.

Description - optional
Enter a description
Descriptions can be up to 2048 characters long.

Tags - optional
Use tags to organize and identify your resources.

Cancel **Next**

5. Select the data source by clicking add data source. Here we have to add the S3 bucket path from which the crawler will read the files.

AWS Glue > Crawlers > Add crawler

Step 1 Set crawler properties

Step 2 Choose data sources and classifiers

Step 3 Configure security settings

Step 4 Set output and scheduling

Step 5 Review and create

Choose data sources and classifiers

Data source configuration

Is your data already mapped to Glue tables?

Not yet
Select one or more data sources to be crawled.

Yes
Select existing tables from your Glue Data Catalog.

Data sources (0) Info

The list of data sources to be scanned by the crawler.

Type	Data source	Parameters
You don't have any data sources.		
Add a data source		

Custom classifiers - optional
A classifier checks whether a given file is in a format the crawler can handle. If it is, the classifier creates a schema in the form of a StructType object that matches that data format.

Cancel **Previous** **Next**

6. Here we browsed and selected the bucket from which we need to take the data from. We need to specify the folder which contains all the subfolders containing the csv files. In our case it is “stockdatacsvfiles/path/”

7. Further we select the option for crawling all sub-folders so that it reads the files from all the subfolders within the folder path we have specified.

Add data source



Data source

Choose the source of data to be crawled.

S3



Network connection - *optional*

Optionally include a Network connection to use with this S3 target. Note that each crawler is limited to one Network connection so any other S3 targets will also use the same connection (or none, if left blank).

▼C

[Clear selection](#)

[Add new connection](#)

Location of S3 data

- In this account
- In a different account

S3 path

Browse for or enter an existing S3 path.

s3://stockdatacsvfiles



[View](#)

[Browse S3](#)

All folders and files contained in the S3 path are crawled. For example, type s3://MyBucket/MyFolder/ to crawl all objects in MyFolder within MyBucket.

Subsequent crawler runs

This field is a global field that affects all S3 data sources.

- Crawl all sub-folders**
Crawl all folders again with every subsequent crawl.
 - Crawl new sub-folders only**
Only Amazon S3 folders that were added since the last crawl will be crawled. If the schemas are compatible, new partitions will be added to existing tables.
 - Crawl based on events**
Rely on Amazon S3 events to control what folders to crawl.
-
- Sample only a subset of files**
 - Exclude files matching pattern**

[Cancel](#)

[Add an S3 data source](#)

8. Next, we select the IAM role which is LabRole in our case.

The screenshot shows the 'Configure security settings' step of a crawler setup. On the left, a sidebar lists steps: Step 1 (Set crawler properties), Step 2 (Choose data sources and classifiers), Step 3 (Configure security settings), Step 4 (Set output and scheduling), and Step 5 (Review and create). The main area is titled 'Configure security settings'. It contains an 'IAM role' section with a dropdown set to 'LabRole' and buttons for 'Create new IAM role' and 'View'. Below this is a note: 'Only IAM roles created by the AWS Glue console and have the prefix "AWSGlueServiceRole-" can be updated.' Under 'Lake Formation configuration - optional', there's a checkbox for 'Use Lake Formation credentials for crawling S3 data source' with a note about account registration. A 'Security configuration - optional' section is also present. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

9. Select the database for storing the table data. We created a database which can be done by selecting the databases option below the data catalog tab in AWS Glue.

The screenshot shows the 'Set output and scheduling' step of a crawler setup. The sidebar is identical to the previous step. The main area is titled 'Set output and scheduling'. It contains an 'Output configuration' section with a 'Target database' dropdown set to 'stock_data' and buttons for 'Clear selection' and 'Add database'. Below it is a 'Table name prefix - optional' field and a 'Maximum table threshold - optional' field with a note about generating tables. An 'Advanced options' section is shown. Under 'Crawler schedule', there's a note about defining cron-like schedules and a 'Frequency' dropdown set to 'On demand'. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

10. Review the configuration settings and click create crawler.

AWS Glue > Crawlers > Add crawler

Step 1 Set crawler properties

Step 2 Choose data sources and classifiers

Step 3 Configure security settings

Step 4 Set output and scheduling

Step 5 Review and create

Review and create

Step 1: Set crawler properties

Name	Description	Tags
Stock_data_Crawler	-	-

Step 2: Choose data sources and classifiers

Type	Data source	Parameters
S3	s3://stockdatacsvfiles	Recrawl all

Step 3: Configure security settings

IAM role	Security configuration	Lake Formation configuration
LabRole	-	-

Step 4: Set output and scheduling

Database	Table prefix - optional	Maximum table threshold - optional	Schedule
stock_data	-	-	On demand

Cancel Previous Create crawler

11. Once created you would be able to select the crawler from the list and run it.

AWS Glue > Crawlers

Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawlers (3) Info

Last updated (UTC)
May 2, 2024 at 22:55:41

Action Run Create crawler

Q Filter crawlers

Table changes from las...

Name	State	Schedule	Last run	Last run timestamp	Log
Stock_data_Crawler	Ready	At 10:00 PM	Succeeded	May 2, 2024 at 22:00:56	View log
mtd_crawler	Ready		Succeeded	March 31, 2024 at 01:41...	View log
nba_crawler_bdi	Ready		Succeeded	February 15, 2024 at 23...	View log

12. Below is the screenshot of all the run by the crawler on a daily basis. The crawler is scheduled to run daily with the help of Eventbridge.

Crawler properties

Name	IAM role	Database	State
Stock_data_Crawler	LabRole	stock_data	READY
Description	Security configuration	Lake Formation configuration	Table prefix
-	-	-	-
Maximum table threshold	-	-	-

Crawler runs (25)

Start time (UTC)	End time (UTC)	Current/last duration	Status	DPU hours	Table changes
May 2, 2024 at 22:00:56	May 2, 2024 at 22:02:08	01 min 12 s	Completed	0.037	-
May 2, 2024 at 05:01:02	May 2, 2024 at 05:02:34	01 min 32 s	Completed	0.040	1 table change, 6 partition changes
May 1, 2024 at 22:00:58	May 1, 2024 at 22:02:06	01 min 08 s	Completed	0.037	-
May 1, 2024 at 05:01:02	May 1, 2024 at 05:02:45	01 min 45 s	Completed	0.041	1 table change, 6 partition changes
April 30, 2024 at 22:00:55	April 30, 2024 at 22:02:05	01 min 09 s	Completed	0.036	-
April 30, 2024 at 05:01:02	April 30, 2024 at 05:02:50	01 min 48 s	Completed	0.049	1 table change, 6 partition changes
April 29, 2024 at 22:02:05	April 29, 2024 at 22:03:21	01 min 15 s	Completed	0.037	-
April 29, 2024 at 05:01:02	April 29, 2024 at 05:02:40	01 min 38 s	Completed	0.039	-
April 28, 2024 at 22:00:50	April 28, 2024 at 22:01:37	47 s	Completed	0.036	-

13. These are the partitions that are created within the main table. Each partition containing data for one stock.

Table overview

Name	Description	Database	Classification
path	-	stock_data	CSV
Location	Connection	Deprecated	Last updated
s3://stockdatascvfiles/path/	-	-	May 2, 2024 at 05:02:33
Input format	Output format	Serde serialization lib	
org.apache.hadoop.mapred.TextInputFormat	org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	

Partitions

partition_0	Files	Properties
nvda	View files	View Properties
amzn	View files	View Properties
googl	View files	View Properties
meta	View files	View Properties
aapl	View files	View Properties
nflx	View files	View Properties

14. Below is a screenshot where we used Athena to show how the data looks.

Query 6 : `1 SELECT * FROM "AwsDataCatalog"."stock_data"."path" limit 10;`

SQL Ln 2, Col 1

Run again Explain Cancel Clear Create

Reuse query results up to 60 minutes ago

Completed Time in queue: 66 ms Run time: 1.187 sec Data scanned: 456.25 KB

Results (10)

#	timestamp	open	high	low	close	volume	partition_0
1	2024-05-01	547.84	560.39	544.25	551.71	3473233	nflx
2	2024-04-30	560.0	560.0	549.375	550.64	3361492	nflx
3	2024-04-29	559.18	559.64	554.24	559.49	2508882	nflx
4	2024-04-26	558.21	562.92	553.19	561.23	4332593	nflx
5	2024-04-25	549.46	566.54	545.705	564.8	3807101	nflx
6	2024-04-24	574.31	576.907	551.3	555.12	5355312	nflx

AWS GLUE ETL JOBS

Create job Info

Author in a visual interface focused on data flow. **Visual ETL**

Author using an interactive code notebook. **Notebook**

Author code with a script editor. **Script editor**

Example jobs Info Create example job

Your jobs (1) Info

Job name	Type	Last modified	AWS Glue version
ETL job	Glue ETL	20/04/2024, 20:18:53	4.0

ETL script

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrame
```

```
# Initialize the Glue context
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext.getOrCreate()
```

```
glueContext = GlueContext(sc)
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Read data into a DynamicFrame
data_source = glueContext.create_dynamic_frame.from_catalog(
    database = "stock_data",
    table_name = "path",
    transformation_ctx = "AWSGlueDataCatalog_node1713657425405"
)

# Convert to DataFrame for more options and better performance on complex operations
df = data_source.toDF()

# Clean the data (example: drop rows with missing values)
df_clean = df.dropna()

# Convert back to a DynamicFrame
dynamic_frame_clean = DynamicFrame.fromDF(df_clean, glueContext,
    "dynamic_frame_clean")

# Write out the cleaned data to S3
glueContext.write_dynamic_frame.from_options(
    frame = dynamic_frame_clean,
    connection_type = "s3",
    connection_options = {"path": "s3://stockdatacsvfiles/output_folder/cleaned_data/"},
    format = "csv"
)

job.commit()
```

ETL job

Script Job details Runs Data quality - updated Schedules Version Control

Script Info

```
3  from awsglue.utils import getResolvedOptions
4  from pyspark.context import SparkContext
5  from awsglue.context import GlueContext
6  from awsglue.job import Job
7  from awsglue.dynamicframe import DynamicFrame
8
9  # Initialize the Glue context
10 args = getResolvedOptions(sys.argv, ['JOB_NAME'])
11 sc = SparkContext.getOrCreate()
12 glueContext = GlueContext(sc)
13 job = Job(glueContext)
14 job.init(args['JOB_NAME'], args)
15
16 # Read data into a DynamicFrame
17 data_source = glueContext.create_dynamic_frame.from_catalog(
18     database = "stock_data",
19     table_name = "path",
20     transformation_ctx = "AWSGlueDataCatalog_node1713657425405"
21 )
22
23 # Convert to DataFrame for more options and better performance on complex operations
24 df = data_source.toDF()
25
26 # Clean the data (example: drop rows with missing values)
27 df_clean = df.dropna()
28
29 # Convert back to a DynamicFrame
30 dynamic_frame_clean = DynamicFrame.fromDF(df_clean, glueContext, "dynamic_frame_clean")
31
32 # Write out the cleaned data to S3
33 glueContext.write_dynamic_frame.from_options(
34     frame = dynamic_frame_clean,
35     connection_type = "s3",
36     connection_options = {"path": "s3://stockdatacsvfiles/output_folder/cleaned_data/"},
37     format = "csv"
38 )
39
40 job.commit()
41
```

Objects (72) Info						
Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload						
<input type="text"/> Find objects by prefix						
Name	Type	Last modified	Size	Storage class		
run-1714626394112-part-r-00002	-	May 2, 2024, 00:06:42 (UTC-05:00)	260.2 KB	Standard		
run-1714626394112-part-r-00000	-	May 2, 2024, 00:06:41 (UTC-05:00)	295.8 KB	Standard		
run-1714626394112-part-r-00001	-	May 2, 2024, 00:06:41 (UTC-05:00)	305.0 KB	Standard		
run-1714626394112-part-r-00005	-	May 2, 2024, 00:06:41 (UTC-05:00)	268.4 KB	Standard		
run-1714626394112-part-r-00003	-	May 2, 2024, 00:06:38 (UTC-05:00)	151.5 KB	Standard		
run-1714626394112-part-r-00004	-	May 2, 2024, 00:06:38 (UTC-05:00)	307.3 KB	Standard		
run-1714539992138-part-r-00002	-	May 1, 2024, 00:06:41 (UTC-05:00)	260.1 KB	Standard		
run-1714539992138-part-r-00000	-	May 1, 2024, 00:06:40 (UTC-05:00)	295.7 KB	Standard		
run-1714539992138-part-r-00001	-	May 1, 2024, 00:06:39 (UTC-05:00)	305.0 KB	Standard		
run-1714539992138-part-r-00003	-	May 1, 2024, 00:06:39 (UTC-05:00)	151.4 KB	Standard		
run-1714539992138-part-r-00004	-	May 1, 2024, 00:06:38 (UTC-05:00)	307.2 KB	Standard		
run-1714539992138-part-r-00005	-	May 1, 2024, 00:06:37 (UTC-05:00)	268.4 KB	Standard		

DATA MODELING

A. SAGEMAKER NOTEBOOK

We utilized the Sagemaker notebook instance for development and testing purposes. Initial plan was to utilize Sagemaker studio for developing predictive models but due to access issues we pivoted to use notebook instance instead. During the time of deployment, we struggled in scheduling this from Eventbridge so we used the notebook instance for development and testing purpose only and decided to store the entire code in form of python script in the S3 bucket.

We created a lambda function to Create and trigger the Sagemaker training job (code for the same will be mentioned below in the document) to execute the code in our python script. We faced a lot of challenges in writing the lambda function for this since we didn't really know how to setup the parameters for it. Took some reference from AWS resources and ChatGPT to understand and debug the errors faced during this entire process.

The Python Script does the following:

- Reads the cleaned data from the S3 bucket (Output of the Glue ETL jobs)
- Perform feature engineering to create indicators which are then used for training the model.
- Split the main data frame into different data frames for each of the 6 stocks.
- Fit a Lasso Regression Model to the each of the data frame.
- Using the model to predict the next price for each stock.

- Storing the output of each stock's predicted price in form of a csv file in the S3 bucket.

Steps to create a notebook instance:

1. Open Amazon Sagemaker service by searching in the search bar.
2. Click on the notebook instance tab on the left-hand menu. It will a page like the one shown in the screenshot below:

Name	Instance	Creation time	Status	Actions
Stockproject	ml.t3.medium	4/20/2024, 8:32:30 PM	Stopped	Start

3. Click on Create notebook instance and configure the notebook instance as per your requirement. For our project we chose to utilize the ml.t3. medium instance type as it comes in free tier, and we are using it for development and testing. Additionally, we stuck with default configuration settings for Volume size and minimum IMDS version.

Notebook instance settings

Notebook instance name: Stock_project

Notebook instance type: ml.t3.medium

Platform identifier: Amazon Linux 2, Jupyter Lab 3

Additional configuration

Lifecycle configuration - optional

Volume size in GB - optional

Minimum IMDS Version - optional

Permissions and encryption

IAM role: LabRole

Create role using the role creation wizard

Root access - optional

4. Click on create notebook instance button once you finish with the configuration settings. Once the notebook instance is created it will look like in the screenshot below. Now you can click on Open JupyterLab to open the notebook for writing the code.

Amazon SageMaker > Notebook instances

Notebook instances Info

Name	Instance	Creation time	Status	Actions
Stockproject	ml.t3.medium	4/20/2024, 8:32:30 PM	InService	Open Jupyter Open JupyterLab

File Edit View Run Kernel Git Tabs Settings Help

Stockdata_project.ipynb

```
[1]: !pip install s3fs
Requirement already satisfied: s3fs in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (0.4.2)
Requirement already satisfied: botocore>=1.12.91 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from s3fs) (1.34.85)
Requirement already satisfied: fsspec==0.6.0 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from s3fs<2024.2.0)
Requirement already satisfied: jmespath>=2.0.0,>=0.7.1 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from botocore>=1.12.91->s3fs) (1.0.1)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from botocore>=1.12.91->s3fs) (2.8.2)
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from botocore>=1.12.91->s3fs) (2.0.7)
Requirement already satisfied: six>=1.5 in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from python-datetimeutil<3.0.0,>=2.1->botocore>=1.12.91->s3fs) (1.16.0)
```

Load Data from Bucket

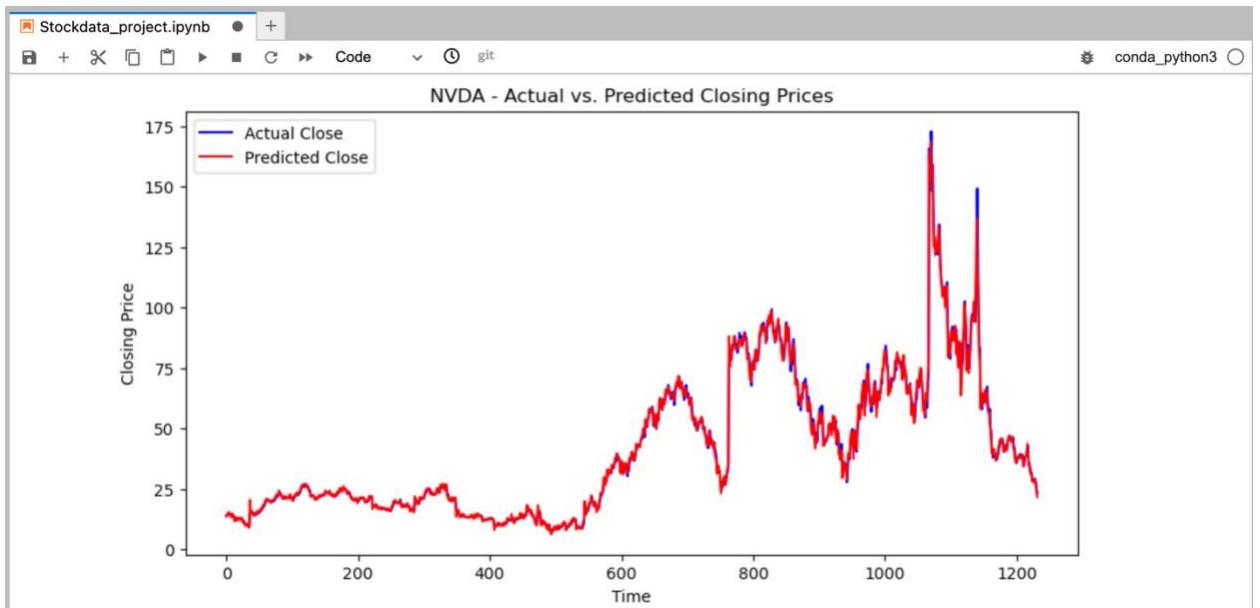
```
[9]: import os
import pandas as pd
import boto3
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import joblib
import argparse

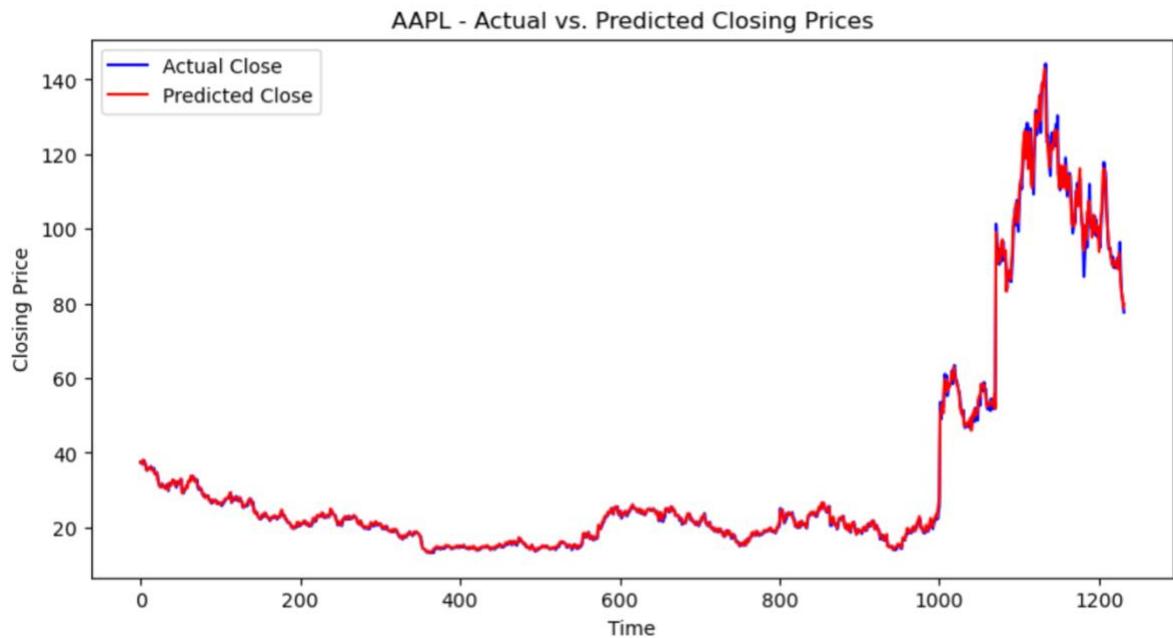
# Define your S3 bucket and path
bucket_name = 'stockdatacsvfiles'
folder_path = 'output_folder/cleaned_data/'

# Create a boto3 S3 client
s3_client = boto3.client('s3')

# Get the list of files from the bucket
response = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=folder_path)
```

5. Here is a screenshot of the notebook we created within Jupyterlab to write our code.





SAGEMAKER JOB

Lambda Script

```

import boto3
import json
import datetime

# Define the Lambda Handler
def lambda_handler(event, context):
    # Set up SageMaker client for us-east-1 region
    sagemaker = boto3.client('sagemaker', region_name='us-east-1')

    # Define job name uniquely
    job_name = f'Daily-Lasso-Model-Training-{datetime.datetime.now():%Y-%m-%d-%H-%M-%S}'

    # Specify the location of your training script and the data in S3

```

```

    training_script_uri = "s3://stockdatacsvfiles/code/sagemaker_training_package.tar.gz"
    # Ensure your bucket is in us-east-1
    output_data_uri = "s3://stockdatacsvfiles/data/output/" # Ensure your bucket is in us-
    east-1

    # Specify the Docker image for the environment in us-east-1 region (this should match
    your Python and library version requirements)
    # Update this URI to the correct image available in us-east-1
    image_uri = '683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-scikit-
    learn:0.23-1-cpu-py3'

    # Define the SageMaker training job
    response = sagemaker.create_training_job(
        TrainingJobName=job_name,
        HyperParameters={
            'sagemaker_program': 'sagemaker_training_script.py',
            'sagemaker_submit_directory': training_script_uri,
            'sagemaker_container_log_level': '20', # Log level INFO
            'sagemaker_job_invocation_id': job_name
        },
        AlgorithmSpecification={
            'TrainingImage': image_uri,
            'TrainingInputMode': 'File'
        },
        RoleArn='arn:aws:iam::891377110529:role/LabRole', # Make sure the role is
        present in us-east-1 and has the right permissions
        OutputDataConfig={
            'S3OutputPath': output_data_uri
        },
        ResourceConfig={
            'InstanceType': 'ml.m5.large',
            'InstanceCount': 1,
            'VolumeSizeInGB': 5
        },
        StoppingCondition={
            'MaxRuntimeInSeconds': 3600
        }
    )

    return {

```

```

'statusCode': 200,
'body': json.dumps(f"Training Job '{job_name}' started successfully.")
}

```

The screenshot shows the AWS Lambda function editor with the code for creating a SageMaker training job. The code uses the boto3 library to interact with SageMaker, specifying the training script, data source, Docker image, and other parameters. It returns a JSON response indicating success.

```

1 import boto3
2 import json
3 import datetime
4
5 # Define the Lambda Handler
6 def lambda_handler(event, context):
7     # Set up SageMaker client for us-east-1 region
8     sagemaker = boto3.client('sagemaker', region_name='us-east-1')
9
10    # Define job name uniquely
11    job_name = f'Daily-Lasso-Model-Training-{datetime.now():%Y-%m-%d-%H-%M-%S}'
12
13    # Specify the location of your training script and the data in S3
14    training_script_url = "s3://stockdatascv/files/code/sagemaker_training_package.tar.gz" # Ensure your bucket is in us-east-1
15    output_data_url = "s3://stockdatascvfiles/data/output/" # Ensure your bucket is in us-east-1
16
17    # Specify the Docker image for the environment in us-east-1 region (this should match your Python and library version requirements)
18    # Update this URI to the correct image available in us-east-1
19    image_uri = '68331688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-sklearn:0.23-1-cpu-py3'
20
21    # Define the SageMaker training job
22    response = sagemaker.create_training_job(
23        TrainingJobName=job_name,
24        HyperParameters={
25            'sagemaker_train_script': 'sagemaker_training_script.py',
26            'sagemaker_submit_directory': training_script_url,
27            'sagemaker_container_log_level': '20', # Log level INFO
28            'sagemaker_job_invocation_id': job_name
29        },
30        AlgorithmSpecification={
31            'TrainingImage': image_uri,
32            'TrainingInputMode': 'File'
33        },
34        RoleArn='arn:aws:iam::891377118529:role/LabRole', # Make sure the role is present in us-east-1 and has the right permissions
35        OutputDataConfig={
36            'S3OutputPath': output_data_url
37        },
38        ResourceConfig={
39            'InstanceType': 'ml.m5.large',
40            'InstanceCount': 1,
41            'VolumeSizeInGB': 5
42        },
43        StoppingCondition={
44            'MaxRuntimeInSeconds': 3600
45        }
46    )
47
48    return {
49        'statusCode': 200,
50        'body': json.dumps(f"Training Job '{job_name}' started successfully.")
51    }
}

```

Below is a list of SageMaker job which were triggered by the eventbridge scheduler to execute the python script for predicting the stock prices daily.

The screenshot shows the Amazon SageMaker console under the 'Training jobs' section. A table lists ten completed training jobs, each with a unique name starting with 'Daily-Lasso-Model-Training-' followed by a timestamp. The table includes columns for Name, Creation time, Duration, Job status, Warm pool status, and Time left.

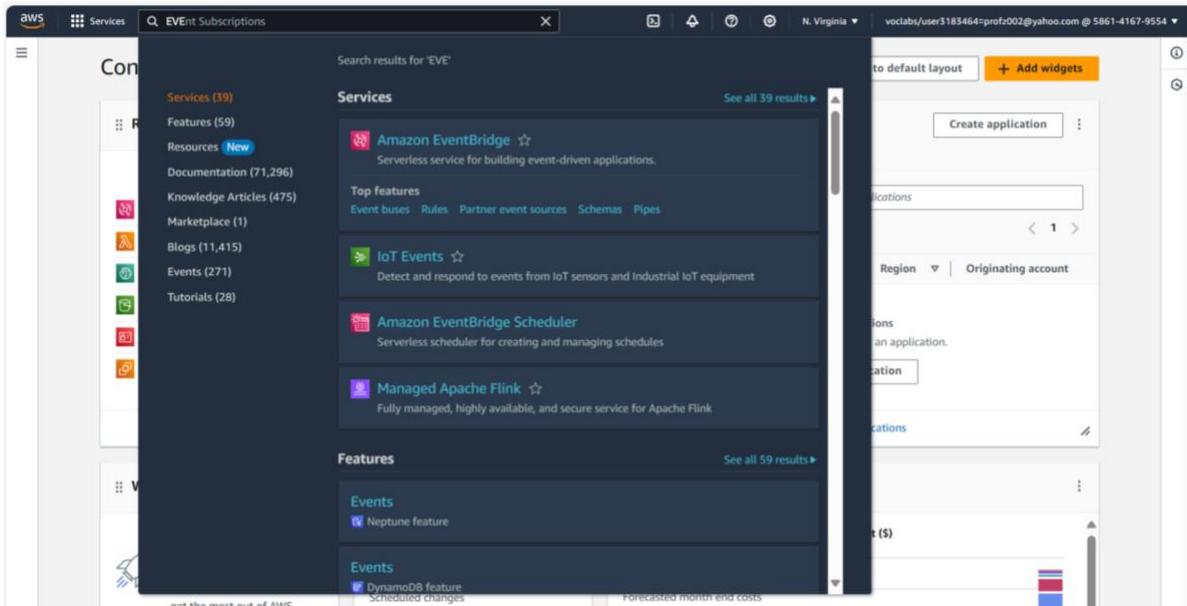
Name	Creation time	Duration	Job status	Warm pool status	Time left
Daily-Lasso-Model-Training-2024-05-02-05-10-03	5/2/2024, 12:10:04 AM	2 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-05-01-05-10-03	5/1/2024, 12:10:04 AM	3 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-30-05-10-03	4/30/2024, 12:10:04 AM	2 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-29-05-10-03	4/29/2024, 12:10:03 AM	2 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-28-05-10-03	4/28/2024, 12:10:03 AM	2 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-27-05-10-03	4/27/2024, 12:10:04 AM	2 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-26-05-10-03	4/26/2024, 12:10:04 AM	2 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-25-05-10-03	4/25/2024, 12:10:04 AM	3 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-24-18-18-43	4/24/2024, 1:18:44 PM	3 minutes	Completed	-	-
Daily-Lasso-Model-Training-2024-04-24-05-10-03	4/24/2024, 12:10:04 AM	2 minutes	Completed	-	-

At Every stage, amazon event bridge is used to schedule other AWS services

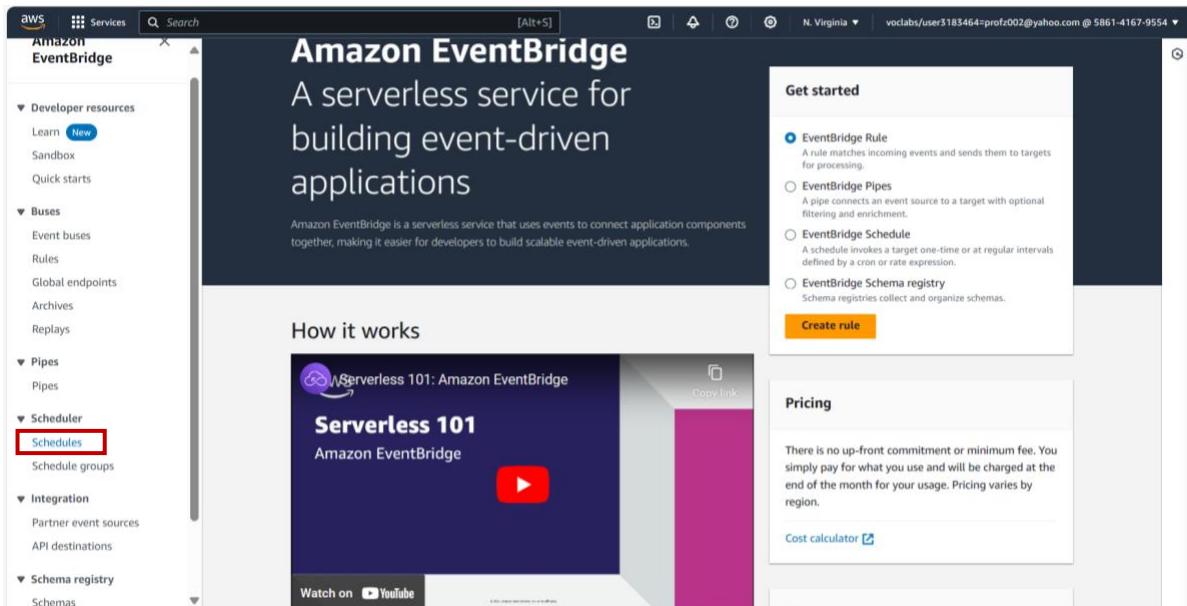
B. AMAZON EVENTBRIDGE

Scheduling data extraction

1. Open Amazon EventBridge



2. Click on Schedules in the left panel



3. Click on ‘Create Schedule’

The screenshot shows the Amazon EventBridge Scheduler interface. On the left, there's a navigation sidebar with sections like 'Developer resources', 'Buses', 'Pipes', 'Scheduler', 'Integration', and 'Schedule editor'. Under 'Scheduler', 'Schedules' is selected. The main area is titled 'Amazon EventBridge Scheduler' and describes it as a serverless scheduler. It features four icons: 'Templated targets', 'Universal targets', 'Flexible time windows', and 'Retries'. Below these are sections for 'Schedules (1)' and a table showing one schedule named 'siri-stockdata' with details like 'Status: Disabled', 'Target: stockprice', and 'Last modified: Apr 20, 2024, 22:23:25 (UTC+00:00)'. A red box highlights the 'Create schedule' button at the top right of the schedule list.

4. Give name, select ‘recurring schedule’ and ‘rate-based schedule’ and give rate as 1 per day. Make flexible time window to ‘off’

The screenshot shows the 'Create schedule' configuration page. At the top, there's a search bar with 'default'. The main area is titled 'Schedule pattern' and includes sections for 'Occurrence', 'Time zone', 'Schedule type', 'Rate expression', and 'Flexible time window'. Under 'Occurrence', 'Recurring schedule' is selected. Under 'Schedule type', 'Rate-based schedule' is selected. Under 'Rate expression', 'rate (1 days)' is set. Under 'Flexible time window', 'Off' is selected. A red box highlights the 'Recurring schedule' option in the occurrence section.

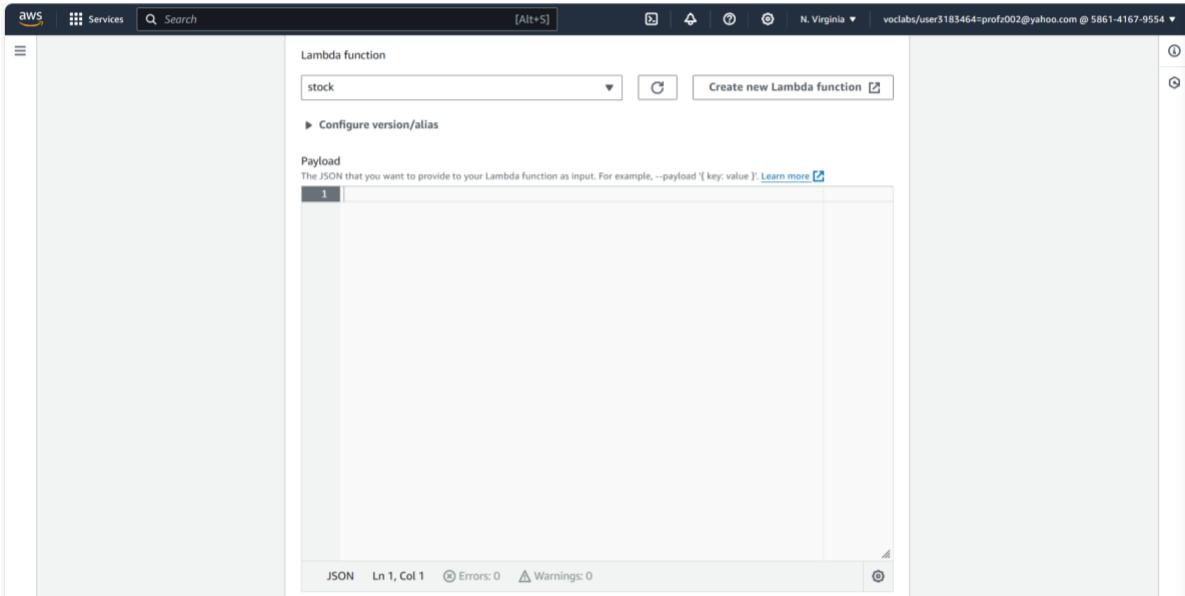
5. Give a time when we want to reload the data everyday. Here it is 12:00 AM

The screenshot shows the 'Timeframe' configuration section of the AWS Lambda Scheduler. It includes fields for 'Start date and time - optional' (set to 2024/04/21 12:00) and 'End date and time - optional' (set to YYYY/MM/DD hh:mm). A note about Daylight saving time is displayed, stating that Amazon EventBridge Scheduler automatically adjusts for daylight saving time. Buttons for 'Cancel', 'Skip to Review and save schedule', and 'Next' are at the bottom.

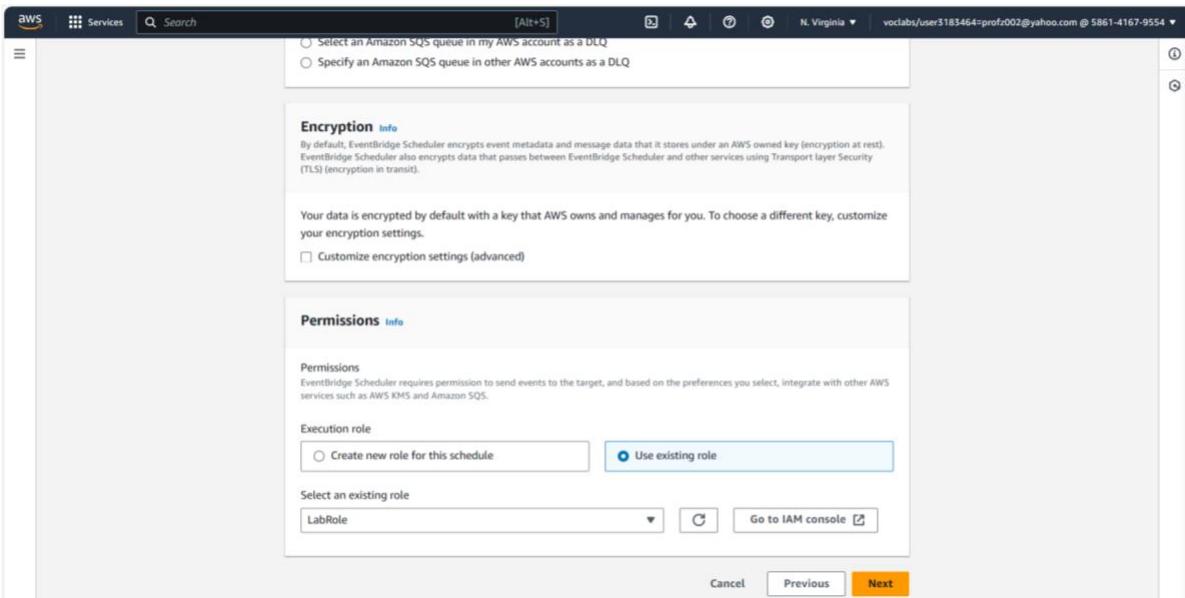
6. After Clicking ‘next’, Select ‘AWS Lambda’

The screenshot shows the 'Select target' step of the AWS Lambda Scheduler configuration. Under 'Target detail', the 'Target API' dropdown is set to 'Info'. The 'Templated targets' tab is selected, showing options like AWS Lambda (Invoke), which is highlighted with a blue border. Other options include CodeBuild (StartBuild), CodePipeline (StartPipelineExecute...), Amazon ECS (RunTask), Amazon EventBridge (PutEvents), Kinesis Data Firehose (PutRecord), Amazon Inspector V1 (StartAssessmentRun), Kinesis Data Streams (PutRecord), SageMaker (StartPipelineExecute...), AWS Step Functions (StartExecution), Amazon SNS (Publish), and Amazon SQS (SendMessage).

7. Select the lambda function created in the previous step and click next



8. Select IAM role, click next and create schedule



9. The new data will be stored everyday at 12:00AM

The screenshot shows the AWS S3 console interface. The path is set to 'Amazon S3 > Buckets > siri-api1 > path/'. The 'Objects' tab is selected, displaying 6 items. The table lists the following details:

Name	Type	Last modified	Size	Storage class
aaplstockdata.csv	CSV	April 20, 2024, 17:21:27 (UTC-05:00)	333.7 KB	Standard
amznstockdata.csv	CSV	April 20, 2024, 17:21:28 (UTC-05:00)	332.1 KB	Standard
googlstockdata.csv	CSV	April 20, 2024, 17:21:29 (UTC-05:00)	276.7 KB	Standard
metastockdata.csv	CSV	April 20, 2024, 17:21:28 (UTC-05:00)	163.5 KB	Standard
nflstockdata.csv	CSV	April 20, 2024, 17:21:29 (UTC-05:00)	291.9 KB	Standard
nvdstockdata.csv	CSV	April 20, 2024, 17:21:30 (UTC-05:00)	322.1 KB	Standard

Scheduling GLUE Crawler for data cleaning

1. Create a new schedule in Eventbridge for scheduling the GLUE crawler. Give the name of the Schedule as GLUE_crawler

The screenshot shows the AWS EventBridge console with the 'Create schedule' wizard open. The steps are as follows:

- Step 1: Specify schedule detail**
 - Schedule name and description**: Schedule name is set to "GLUE_Crawler". Description is optional.
- Step 2: Select target**
- Step 3: Settings**
- Step 4: Review and create schedule**

In the 'Schedule pattern' section, the 'Occurrence' dropdown is set to "Recurring schedule". In the 'Schedule type' section, the "Rate-based schedule" option is selected.

2. Next, we select a recurring schedule as we want to execute the script daily once. So, I select rate expression as 1 and the unit as days to run the crawler 1 time a day.

Rate expression [Info](#)

Enter a value and the unit of time to run the schedule.

rate (days)
Value Unit

Flexible time window

If you choose a flexible time window, Scheduler invokes your schedule within the time window you specify. For example, if you choose 15 minutes, your schedule runs within 15 minutes after the schedule start time.

Off

Timeframe



Daylight saving time

Amazon EventBridge Scheduler automatically adjusts your schedule for daylight saving time. When time shifts forward in the Spring, if a cron expression falls on a non-existent date, your schedule invocation is skipped. When time shifts backwards in the Fall, your schedule runs only once and does not repeat its invocation. The following invocations occur normally at the specified date and time.

Start date and time - *optional*

The start date and time of the schedule.

2024/04/22



00:00

YYYY/MM/DD

Use 24-hour format timestamp (hh:mm)

End date and time - *optional*

The end date and time of the schedule.

YYYY/MM/DD



hh:mm

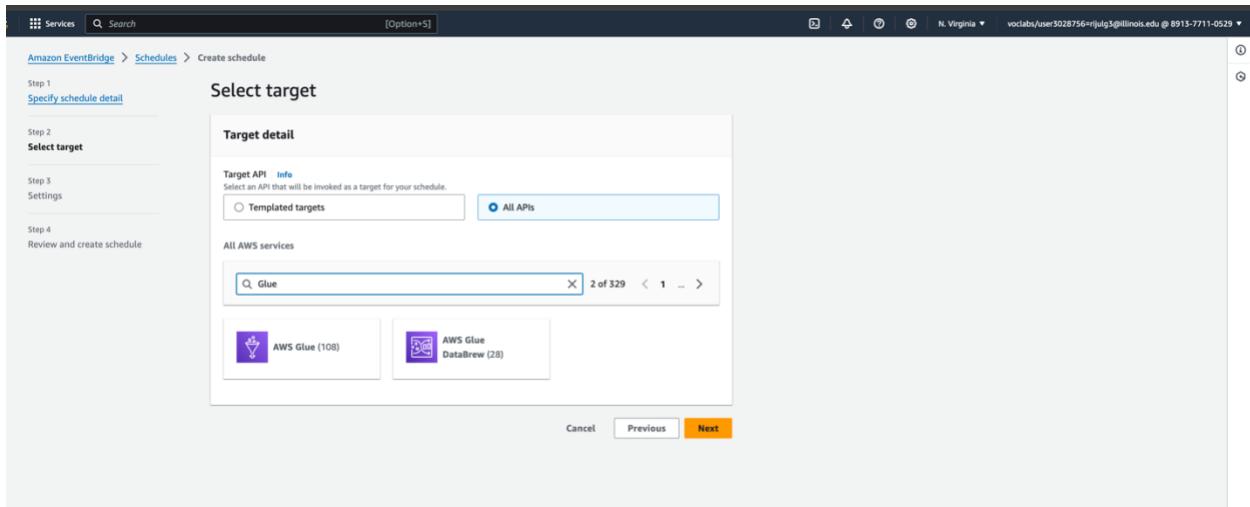
YYYY/MM/DD

Use 24-hour format timestamp (hh:mm)

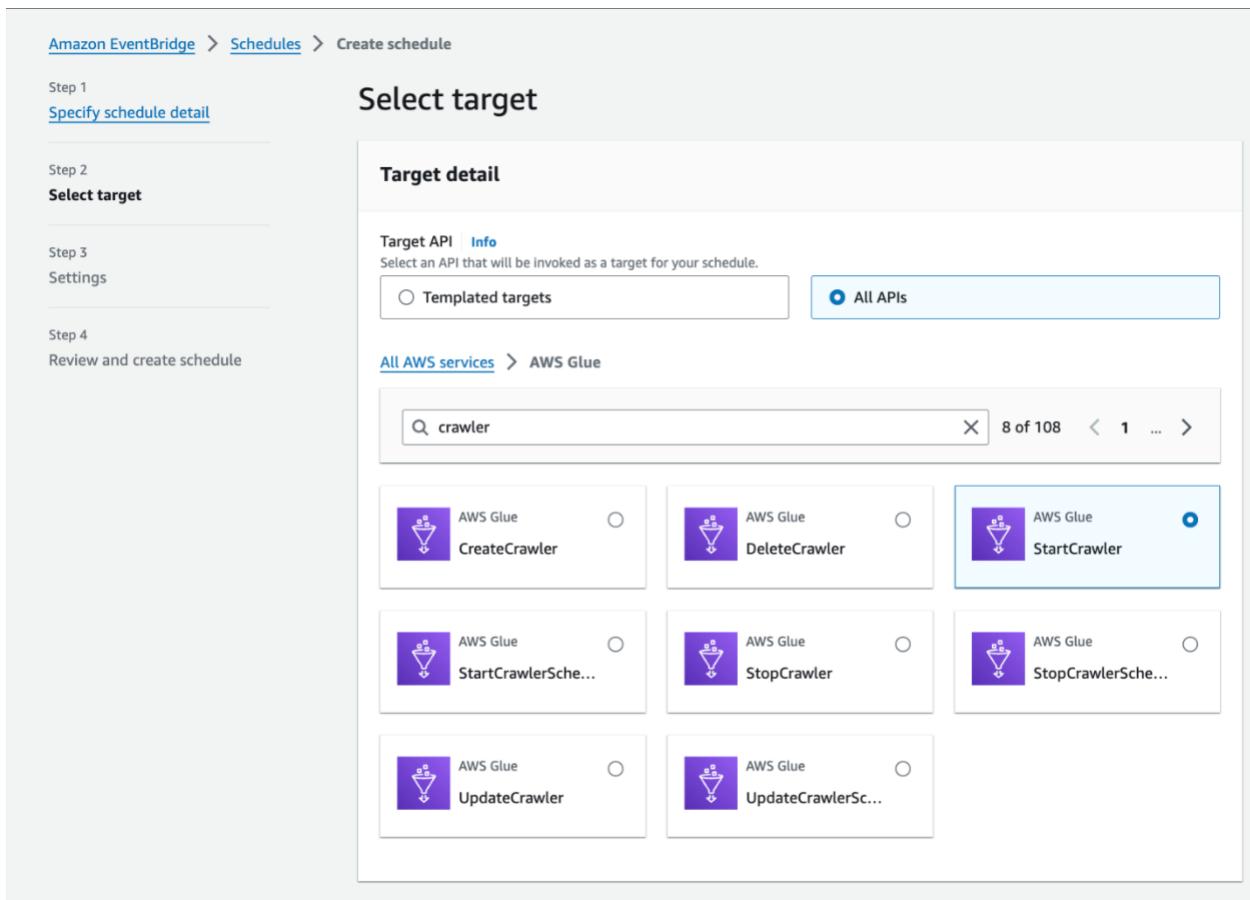
Cancel

Next

3. Next, we will define the start date and time for the schedule. We chose to not fill the end date but that can all be set for a different use-case.
4. Now we need to select the appropriate API's for scheduling the crawler. For this select All API's and enter crawler in the search bar. Select AWS Glue.



5. Glue has a lot of APIs which serve different purpose. For running the crawler, we need to use Start Crawler API from this list. For this enter crawler in the search bar and select start crawler API.



6. Give the name of the crawler to schedule and click on Next.

StartCrawler

AWS Glue

Input

JSON object containing the parameters to pass into the API [\[?\]](#). Contains sample values. Update the JSON with your own values. Note: parameter names must be in PascalCase. [Learn more \[?\]](#)

```
1 ▼ {  
2   "Name": "Stock_Data_Crawler"  
3 }
```

JSON Ln 2, Col 30 ✖ Errors: 0 ⚠ Warnings: 0

Cancel Previous Next

7. Select the IAM role as LabRole. All other settings can be let to be default.

The maximum amount of time to keep unprocessed events. The maximum and default value is 24 hours.

hour(s) minute(s)

Retry attempts - optional

The maximum number of times to retry when a target returns an error. The maximum value is 185 times.

times

Dead-letter queue (DLQ)

Standard Amazon SQS queues that EventBridge Scheduler uses to store events that couldn't be delivered successfully to a target.

None

Select an Amazon SQS queue in my AWS account as a DLQ

Specify an Amazon SQS queue in other AWS accounts as a DLQ

Encryption Info

By default, EventBridge Scheduler encrypts event metadata and message data that it stores under an AWS owned key (encryption at rest). EventBridge Scheduler also encrypts data that passes between EventBridge Scheduler and other services using Transport layer Security (TLS) (encryption in transit).

Your data is encrypted by default with a key that AWS owns and manages for you. To choose a different key, customize your encryption settings.

Customize encryption settings (advanced)

Permissions Info

Permissions

EventBridge Scheduler requires permission to send events to the target, and based on the preferences you select, integrate with other AWS services such as AWS KMS and Amazon SQS.

Execution role

Create new role for this schedule Use existing role

Select an existing role

LabRole ▼ Go to IAM console

Cancel Previous Next

8. Finally review all settings for the schedule and click Create schedule.

Amazon EventBridge > [Schedules](#) > Create schedule

Step 1 [Specify schedule detail](#)

Step 2 - optional [Select target](#)

Step 3 - optional [Settings](#)

Step 4 [Review and create schedule](#)

Review and create schedule

Step 1: Schedule detail

Schedule detail

Schedule name GLUE_Crawler	Description -	Schedule group default
Time zone (UTC-05:00) America/Chicago	Occurrence Recurring	Start date and time 2024-04-22 00:00
End date and time -	Flexible time window Off	
Rate expression		
rate (1 days)		

Step 2: Target

Target detail

Target AWS Glue Payload { "Name": "Stock_Data_Crawler" }	Target ARN -
---	-----------------

Step 3: Settings

Step 3: Settings Edit

Schedule state and permissions

Schedule state Enabled	Execution role LabRole
Action after schedule completion -	

Retry policy and dead-letter queue (DLQ)

Retry policy Max age of event: 24 hours 0 minutes	Retry policy Maximum retries: 185
Dead-letter queue ARN None	

Encryption

Customer master key (CMK) aws/scheduler	Description Default master key that protects my Amazon EventBridge Scheduler data when no other key is defined
Key ARN -	

Cancel
Previous
Create schedule

Scheduling Glue ETL Job

1. For scheduling ETL job we follow the same steps as for scheduling the crawler above. Only difference is the API we use for this purpose. In case of ETL jobs we need to look for StartJobRun API from the API list for Glue.

Amazon EventBridge > Schedules > Create schedule

Step 1
[Specify schedule detail](#)

Step 2
Select target

Step 3
Settings

Step 4
Review and create schedule

Select target

Target detail

Target API [Info](#)
Select an API that will be invoked as a target for your schedule.

Templated targets All APIs

All AWS services > AWS Glue

Q job X 8 of 108 < 1 ... >

 AWS Glue BatchStopJobRun	 AWS Glue CreateJob	 AWS Glue DeleteJob
 AWS Glue ResetJobBookmark	 AWS Glue StartJobRun	 AWS Glue UpdateJob
 AWS Glue UpdateJobFromS...	 AWS Glue UpdateSourceCon...	

2. Now give the name of the ETL job to be scheduled.

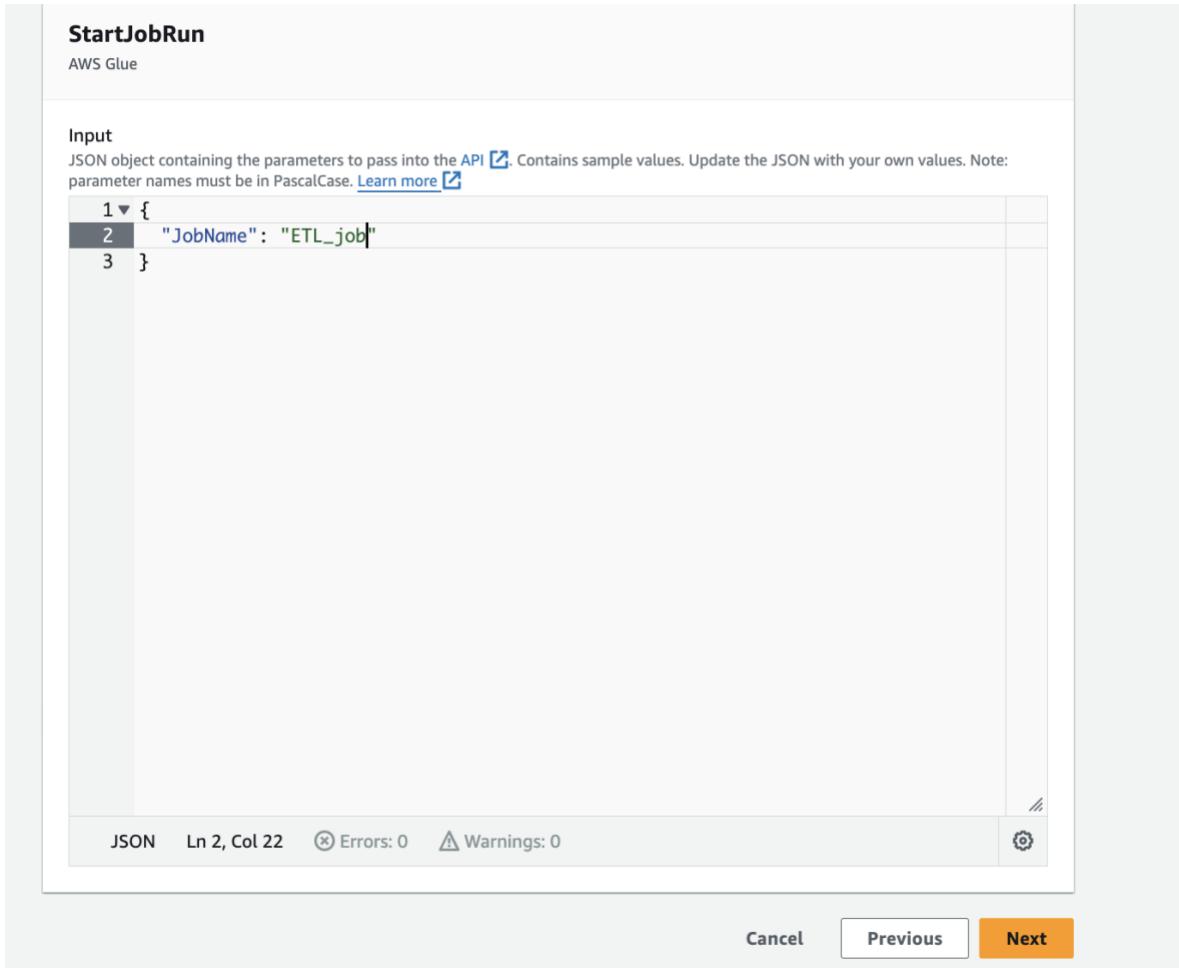
StartJobRun
AWS Glue

Input
JSON object containing the parameters to pass into the API. Contains sample values. Update the JSON with your own values. Note: parameter names must be in PascalCase. [Learn more](#)

```
1 {  
2   "JobName": "ETL_job"  
3 }
```

JSON Ln 2, Col 22 Errors: 0 Warnings: 0

Cancel Previous **Next**



Scheduling Sagemaker Job Trigger.

1. Just like we scheduled the job function for data ingestion we will follow the same steps to trigger the Sagemaker Job.

Step 1
[Specify schedule detail](#)

Step 2
Select target

Step 3 - optional
[Settings](#)

Step 4
[Review and create schedule](#)

Select target

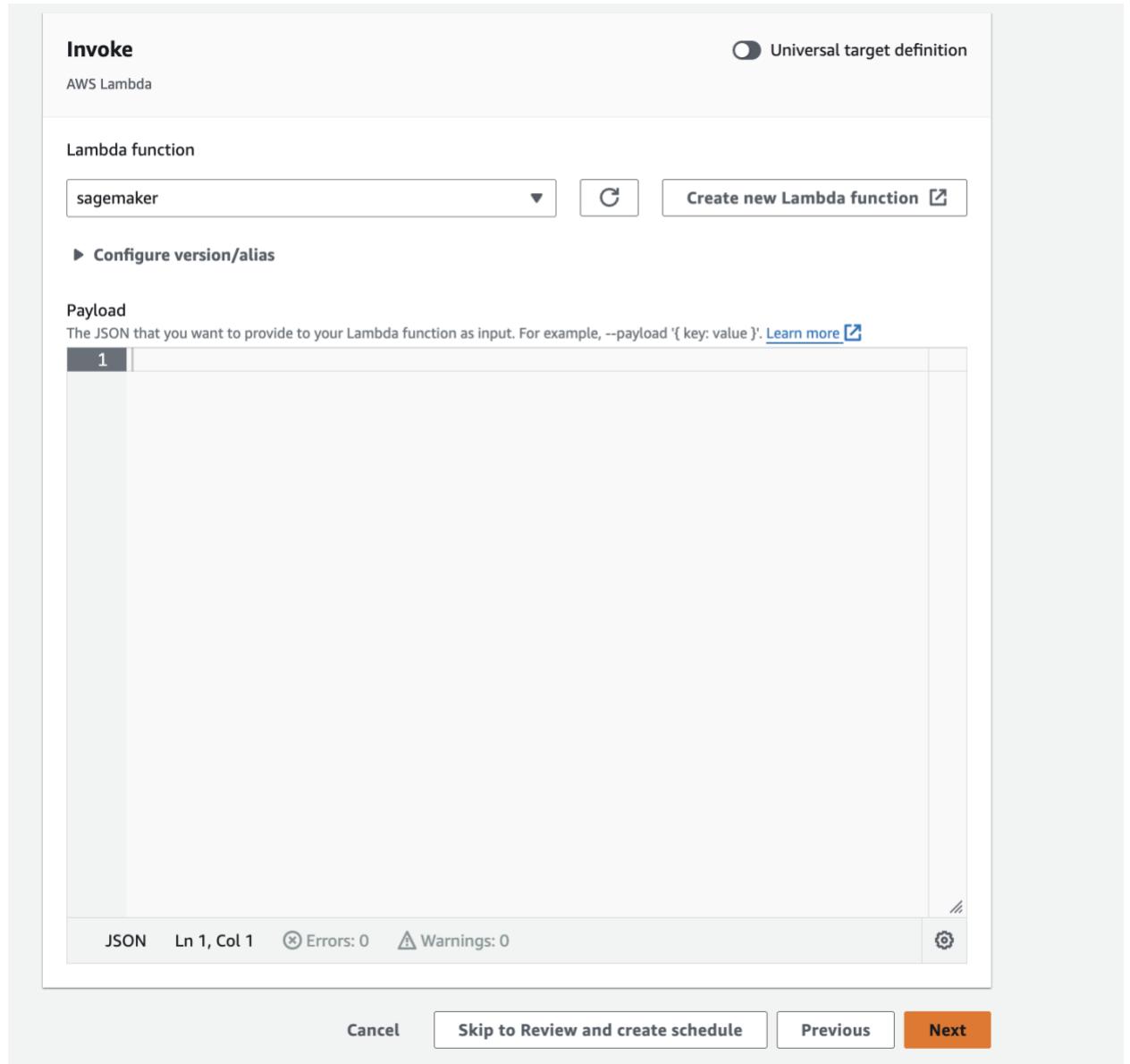
Target detail

Target API [Info](#)
Select an API that will be invoked as a target for your schedule.

Templated targets All APIs

 CodeBuild StartBuild	 CodePipeline StartPipelineExecut...	 Amazon ECS RunTask
 Amazon EventBridge PutEvents	 Kinesis Data Firehose PutRecord	 Amazon Inspector V1 StartAssessmentRun
 Kinesis Data Streams PutRecord	 AWS Lambda Invoke	 SageMaker StartPipelineExecut...
 AWS Step Functions StartExecution	 Amazon SNS Publish	 Amazon SQS SendMessage

2. Instead of using lambda function for data ingestion, we would invoke the lambda function for triggering Sagemaker job.



Utilized AWS SageMaker for feature engineering and to train a machine learning model capable of predicting stock prices.

CONCLUSION

The infrastructure set up for predicting stock prices addresses the primary concerns of scalability, efficiency, and integration. By leveraging AWS services, the solution ensures that financial data is collected, processed, and visualized efficiently, enabling timely and informed decision-making in the financial sector.