# 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, Juypter 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

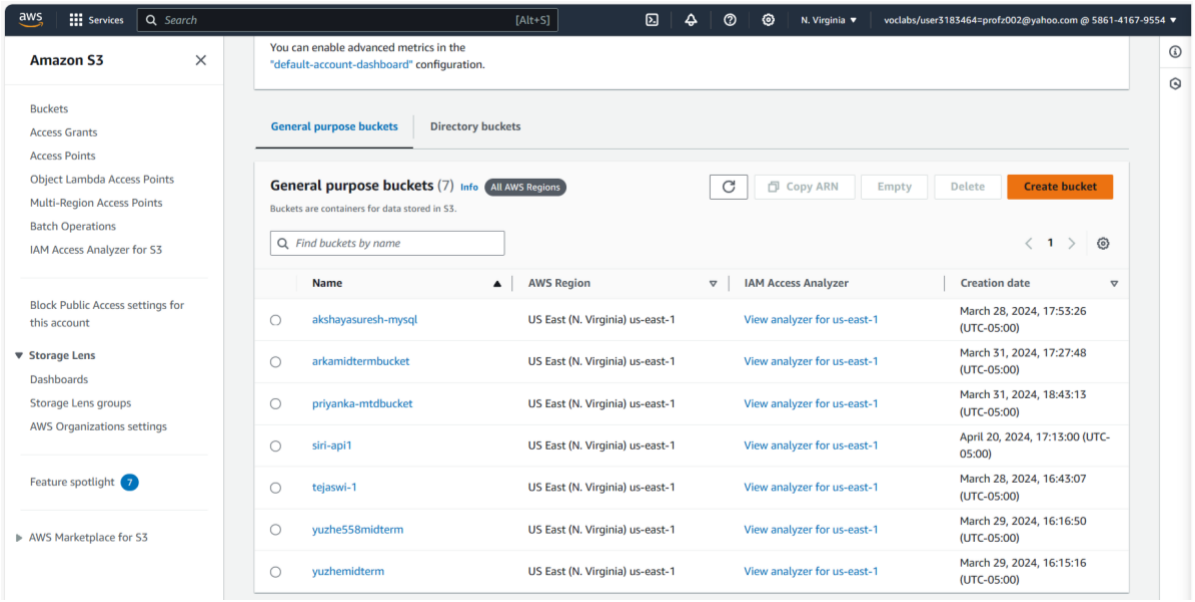
```
import csv
import requests
import boto3
from io import StringIO
def lambda_handler(event, context):
    # Define the list of stock symbols to process
    symbols = ['AAPL', 'AMZN', 'META', 'NFLX', 'GOOGL', 'NVDA']

    # 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,
ContentType='text/csv')
                results.append(f'CSV file for {symbol} successfully saved to S3 in its own folder.')
            else:
```

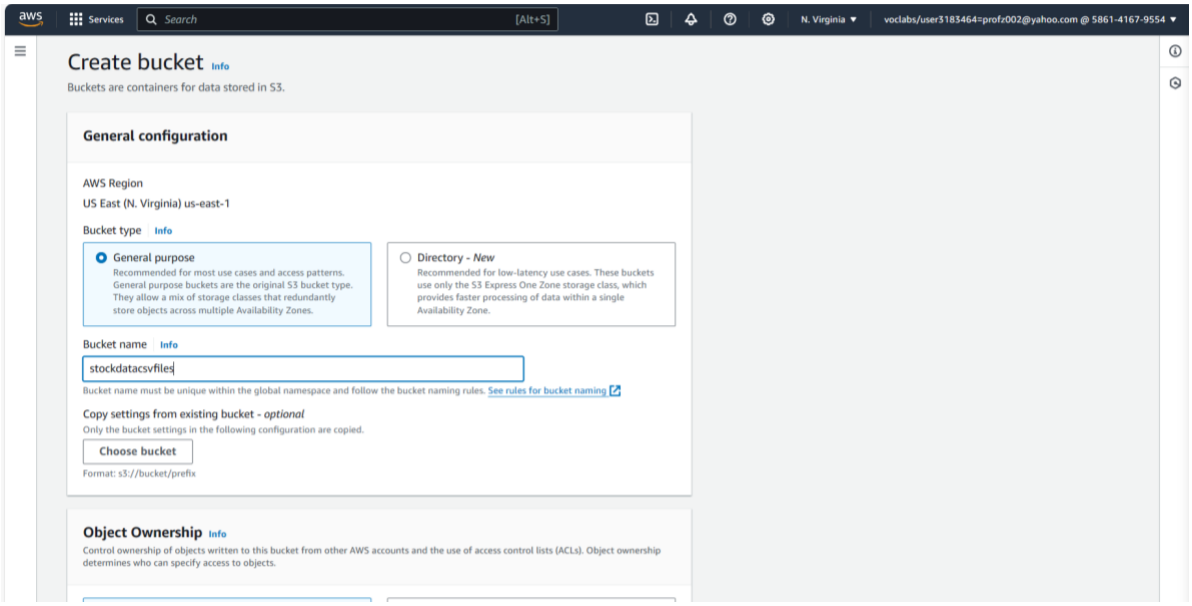
```
            raise ValueError(f"Unexpected content type for {symbol}:
{response.headers['Content-Type']}")
        except Exception as e:
            results.append(f'Error processing {symbol}: {str(e)}')

    return {
        'statusCode': 200,
        'body': results
    }
```

**A. S3**

1. Open S3 and create a bucket

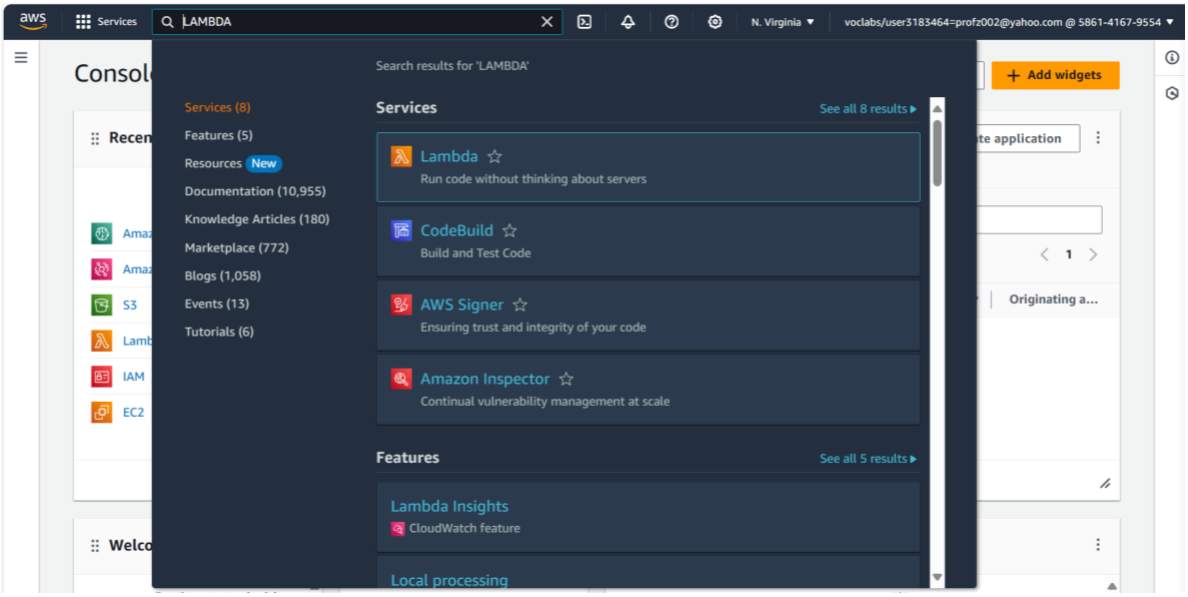


2. Give a bucket name

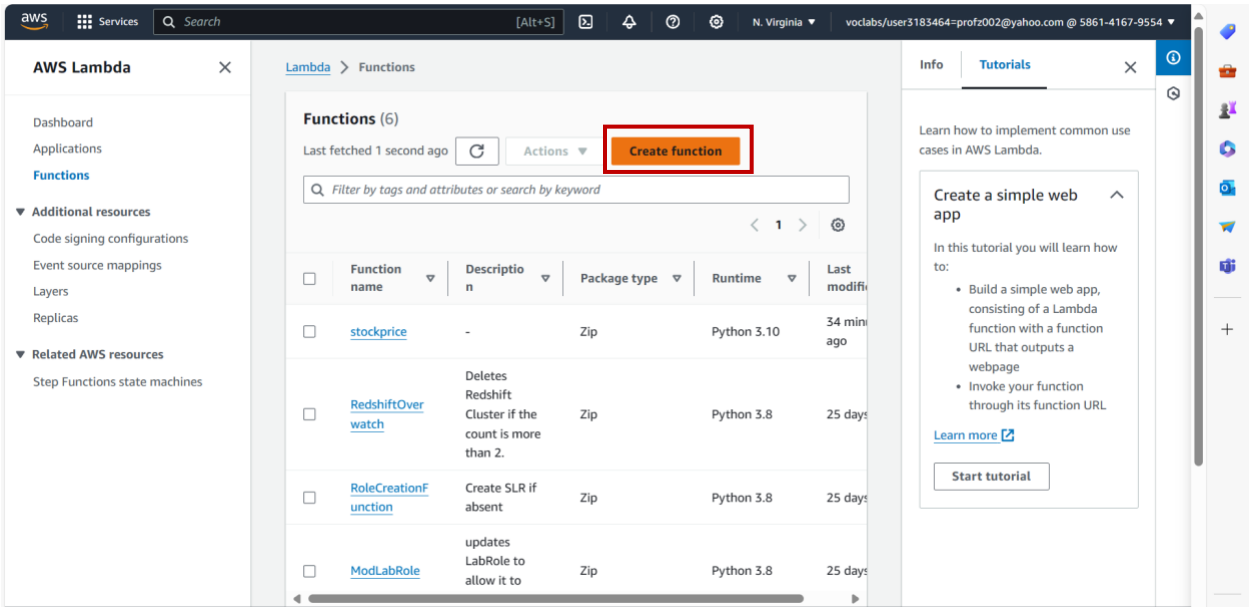



## B. LAMBDA

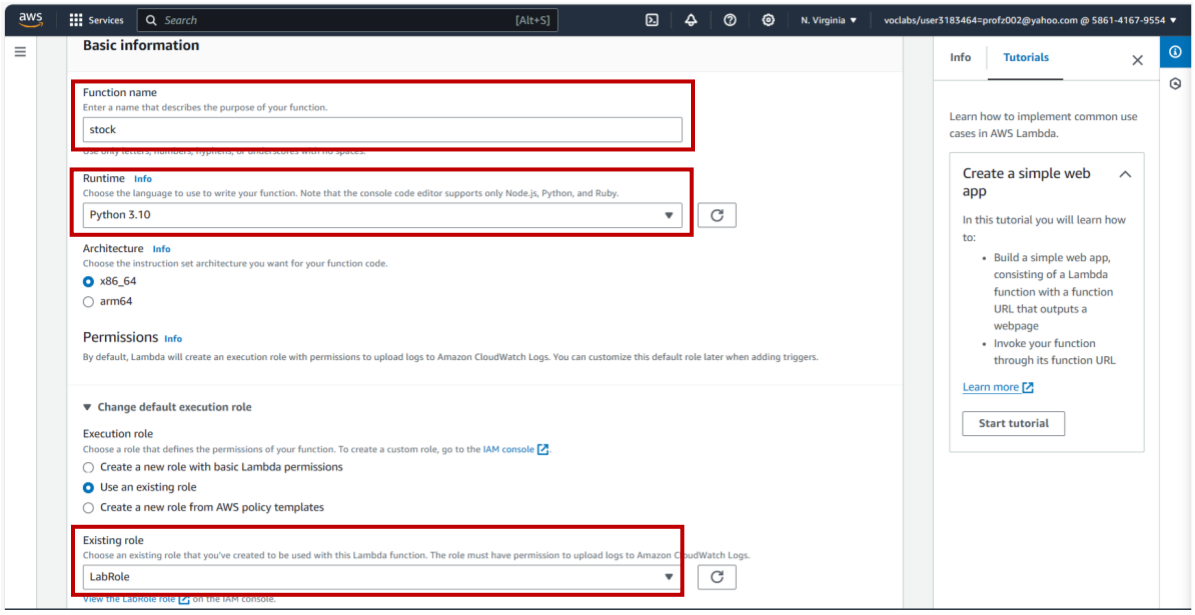
1. Open AWS Lambda



2. Click on create function



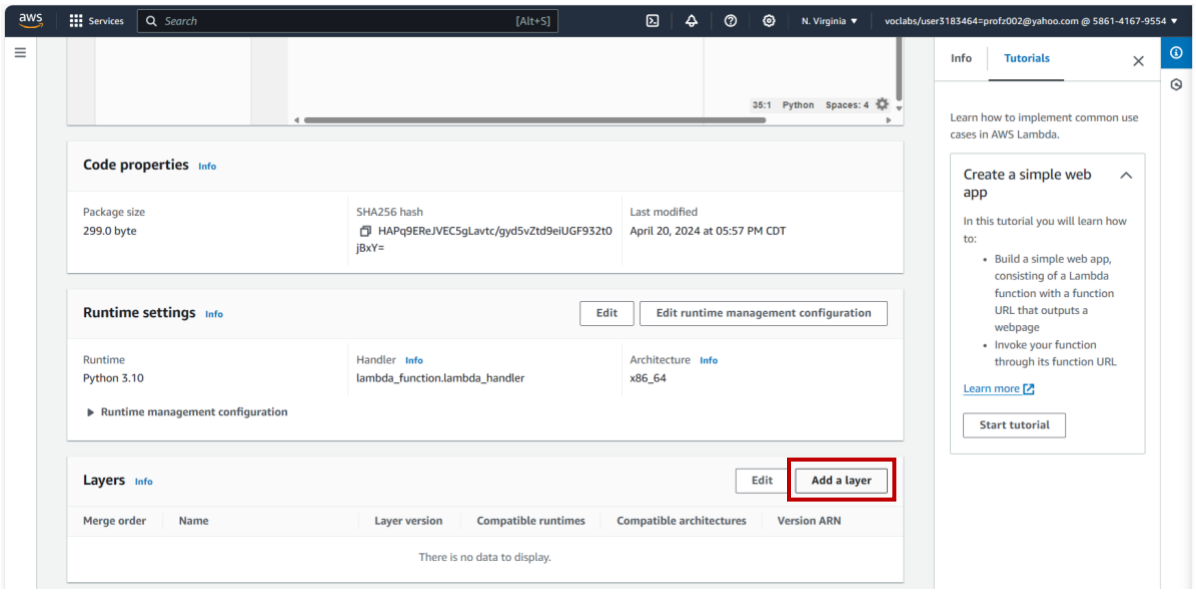

3. Give a function name, Select run time (Python 3.10 here ) and Select an IAM Role and click on create function



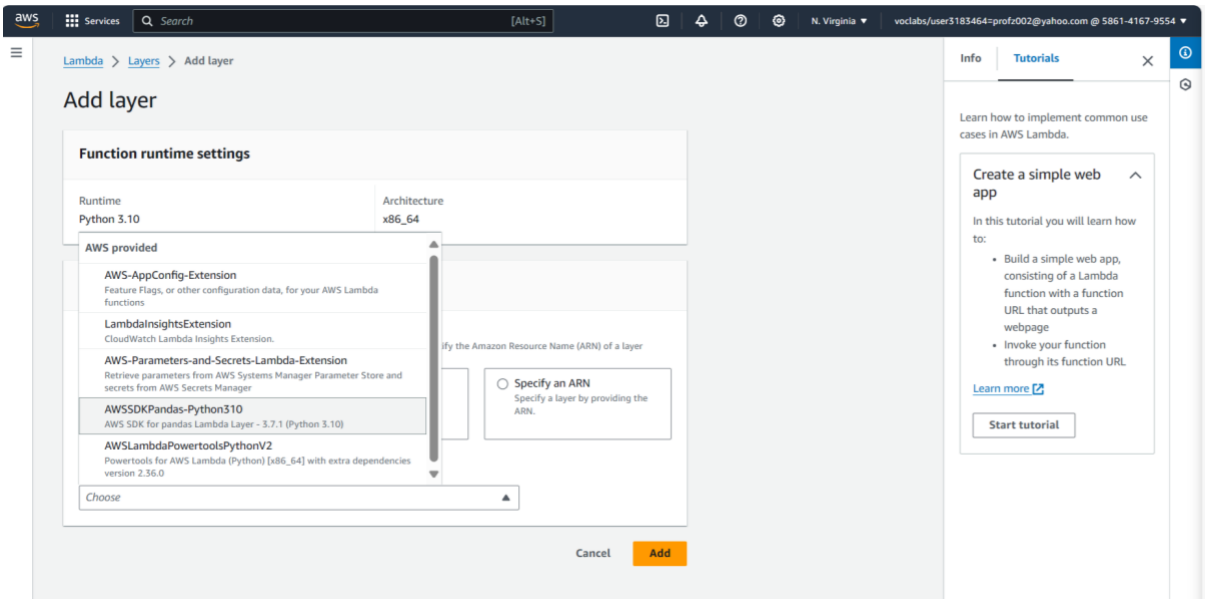
4. Enter the code in the ‘Code’ section of the function

```python
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, ContentType='text/csv')
                results.append(f'CSV file for {symbol} successfully saved to S3 in its own folder.')
            else:
                raise ValueError(f"Unexpected content type for {symbol}: {response.headers['Content-Type']}")
        except Exception as e:
            results.append(f'Error processing {symbol}: {str(e)}')

    return {
        'statusCode': 200,
        'body': results
    }

```

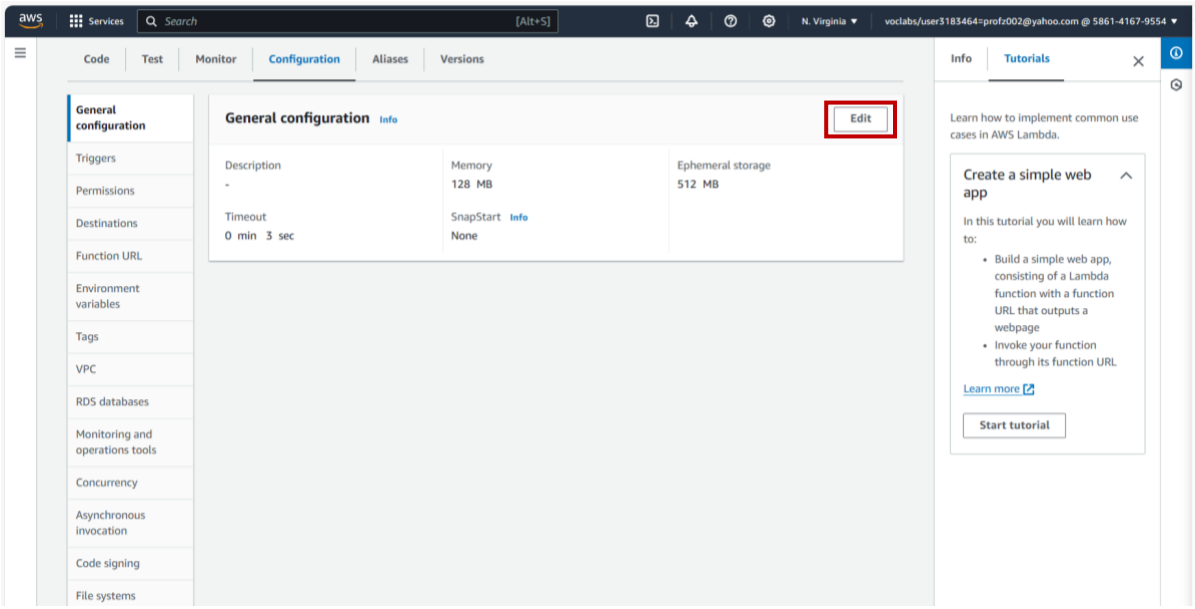
5. Click on ‘add a layer’ in layers section



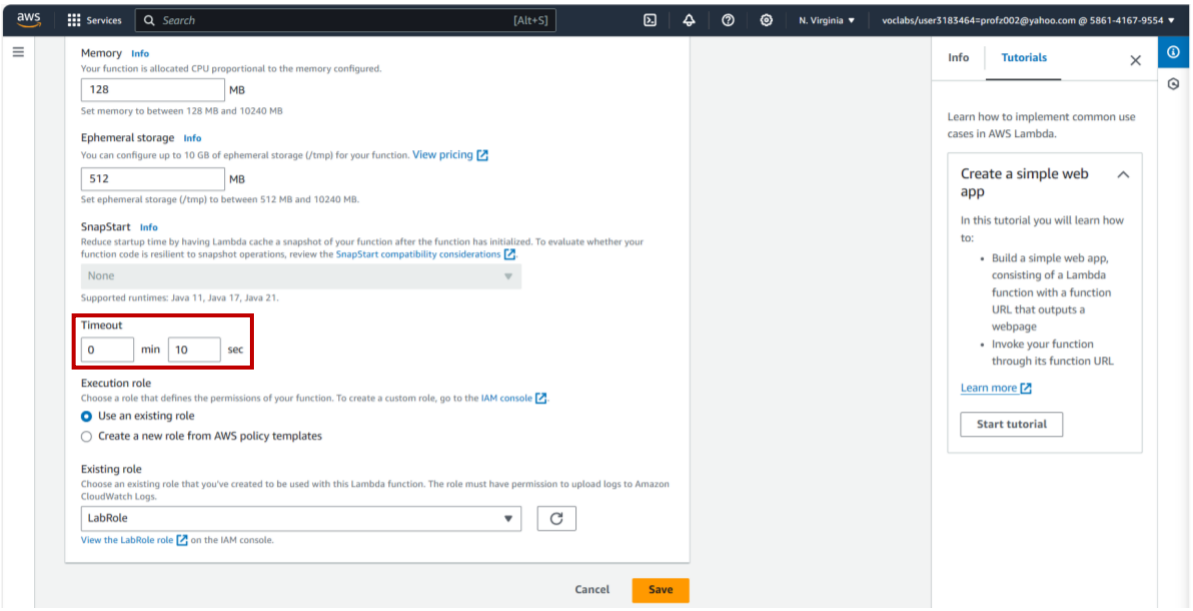
6. Select AWS layer with pandas and other related packages installed and click on ‘add’



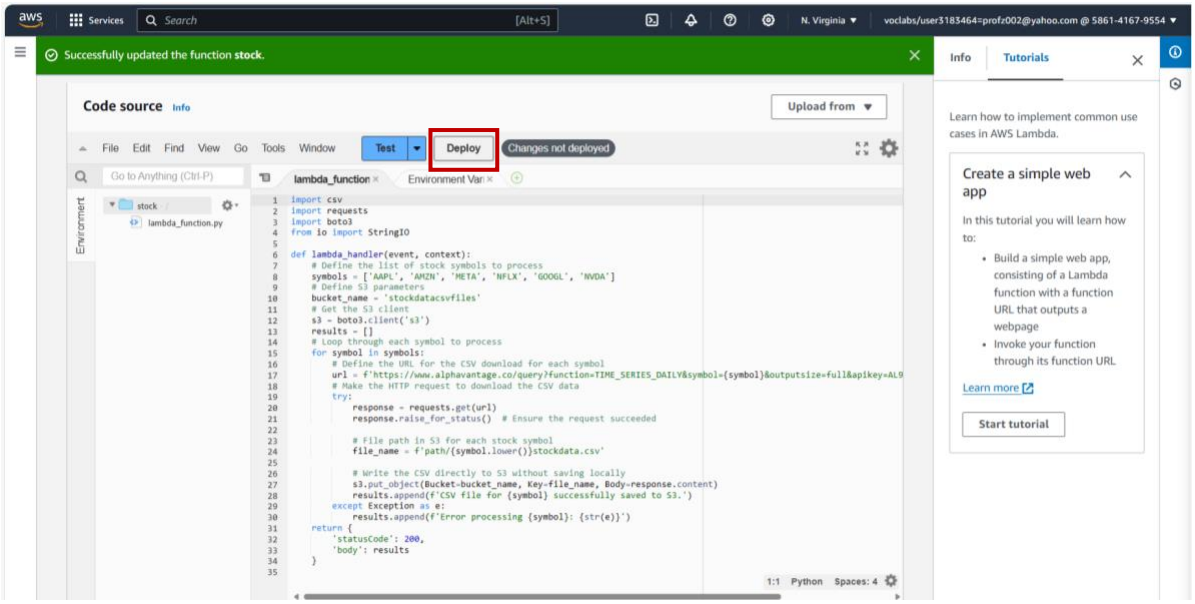

7. Go to ‘Configuration’ and click on Edit



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

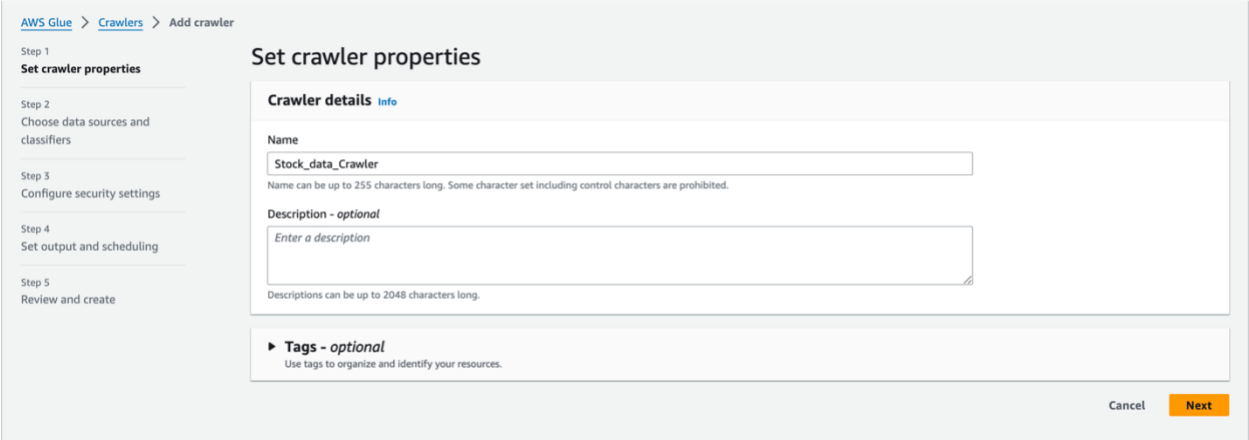
Setting up Glue Crawler

1. Open AWS Glue from the main menu.

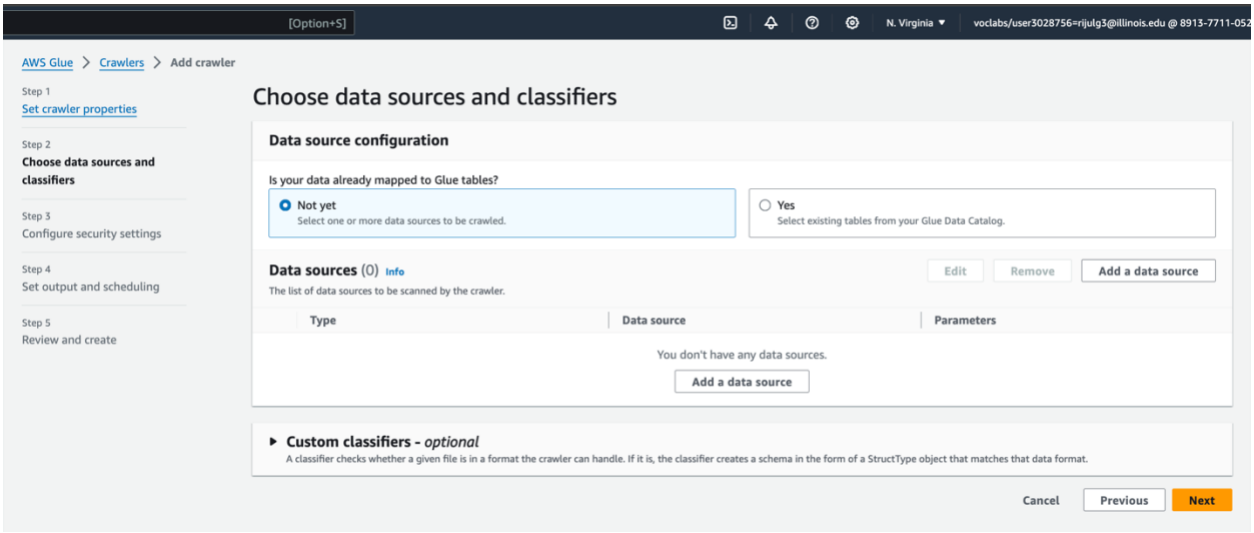
2. Click on Crawler tab from the left-hand menu on the AWS Glue page.

3. Click create crawler.

4. Give your crawler a name: Here we are Stock_data_Crawler as the name for the crawler.




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.




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).

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.

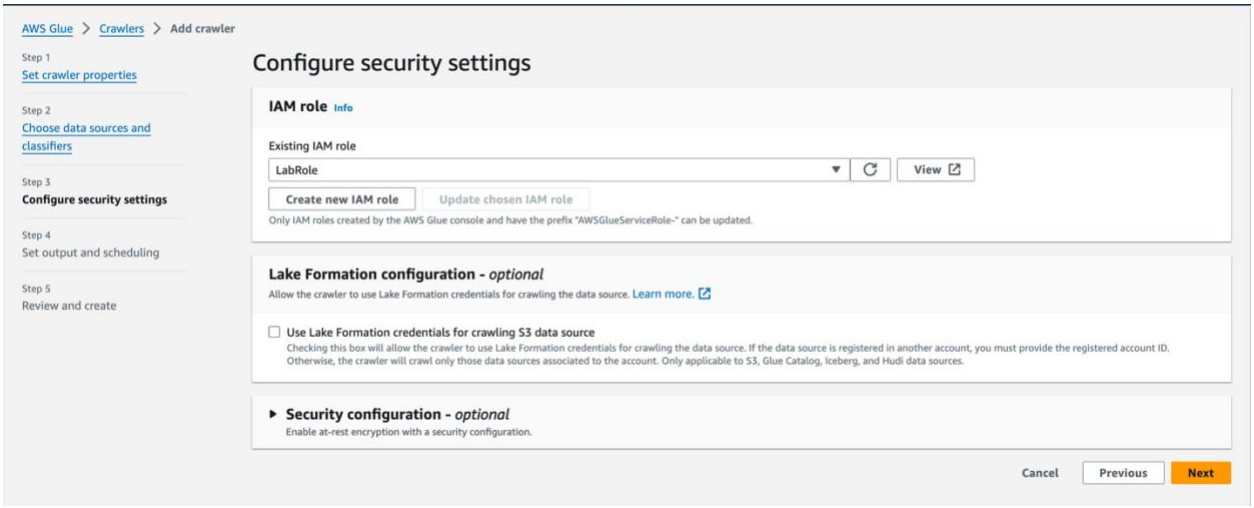
- (•) Crawl all sub-folders
  Crawl all folders again with every subsequent crawl.
- ( ) Crawl new sub-folders only
  Only Amazon S3 folders that were added since the last crawl will be crawled. If the schemas are compatible, new partitions will be added to existing tables.
- ( ) Crawl based on events
  Rely on Amazon S3 events to control what folders to crawl.

[ ] Sample only a subset of files

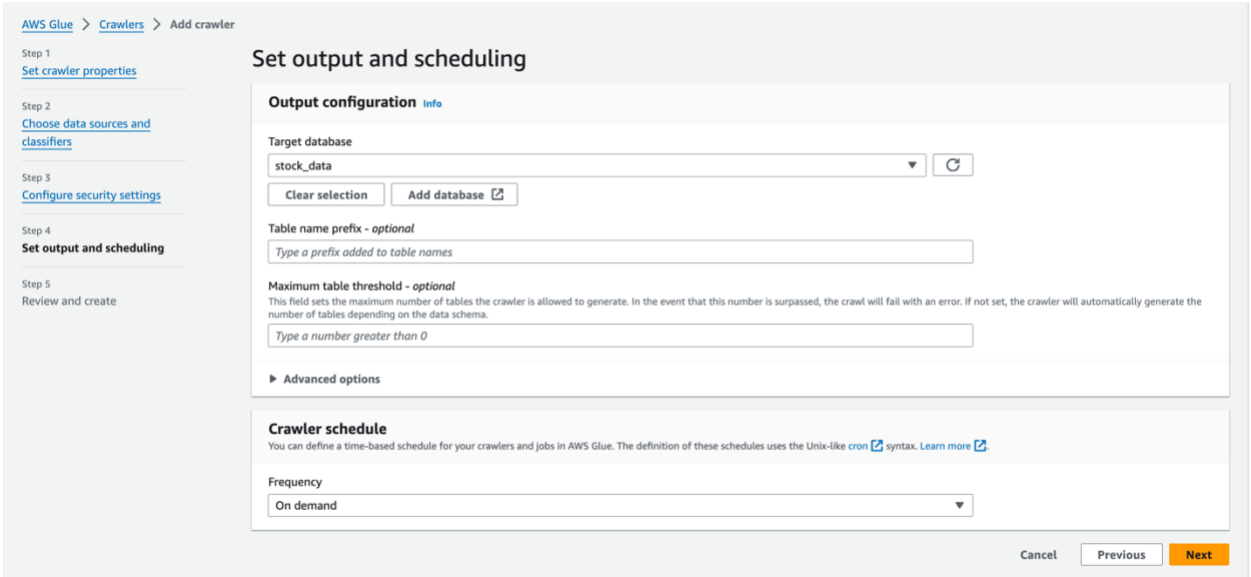
[ ] Exclude files matching pattern

Cancel | Add an S3 data source

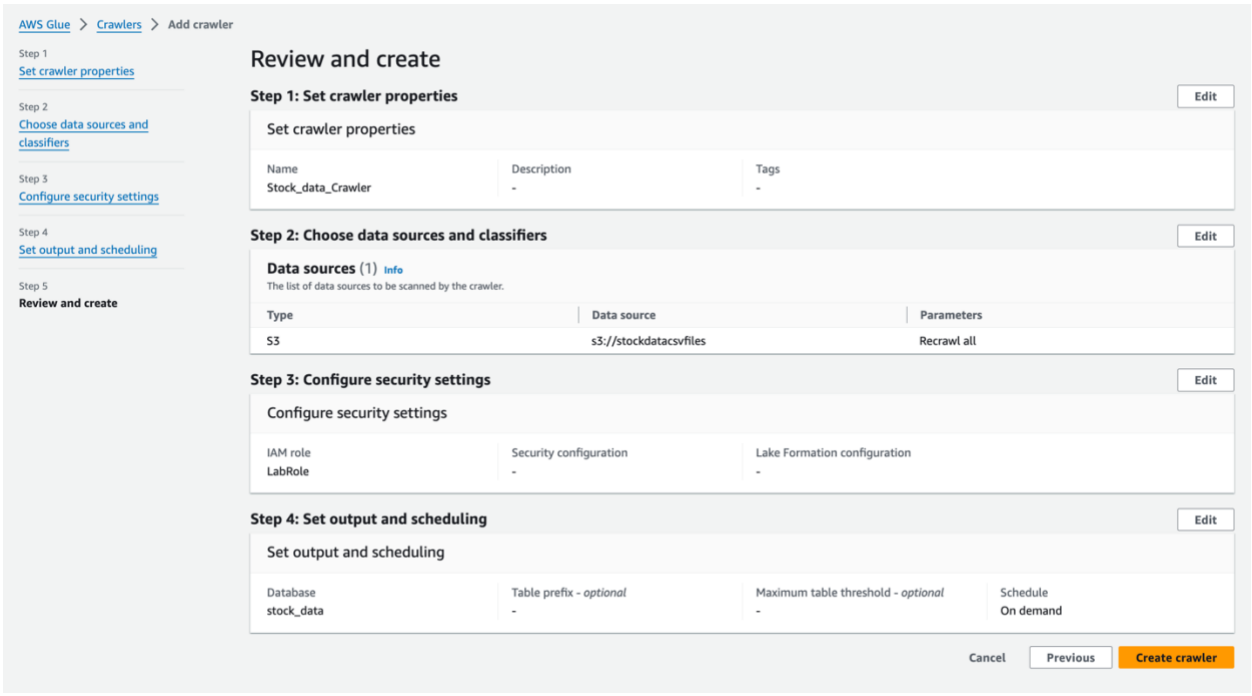
8. Next, we select the IAM role which is LabRole in our case.



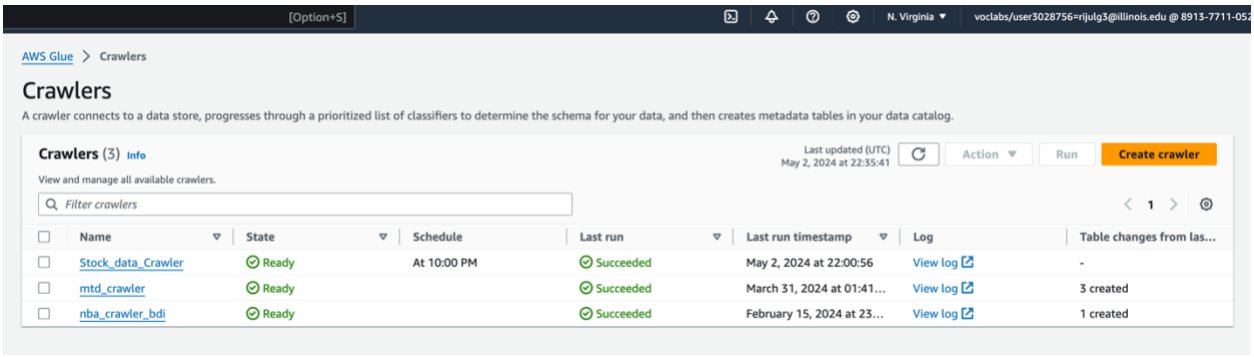

9. Select the database for storing the table data. We created a database which can be done by selecting the databases option below the data catalog tab in AWS Glue.



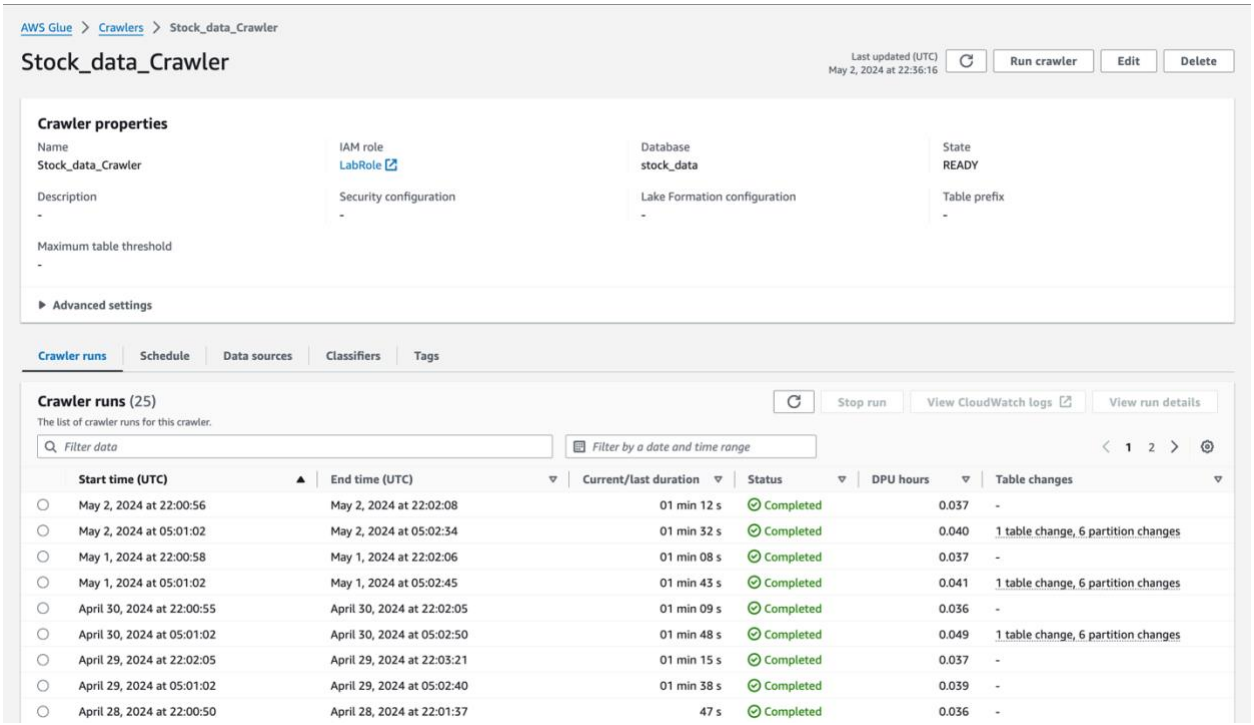

10. Review the configuration settings and click create crawler.



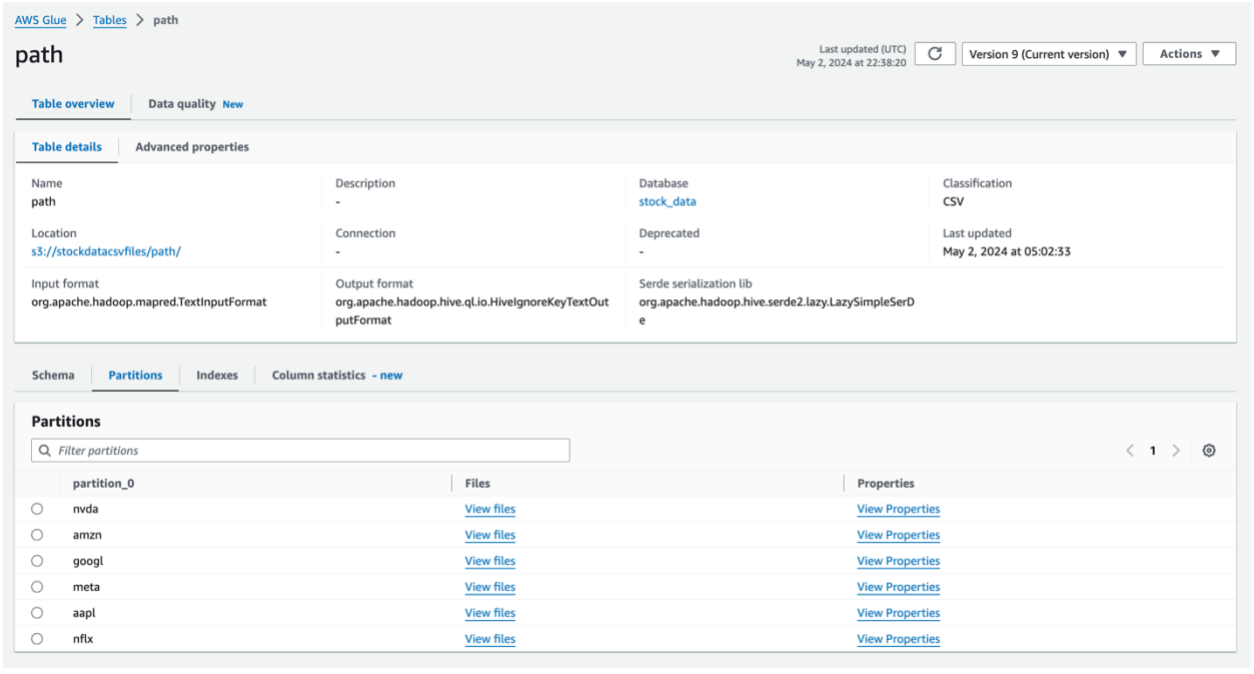

11. Once created you would be able to select the crawler from the list and run it.



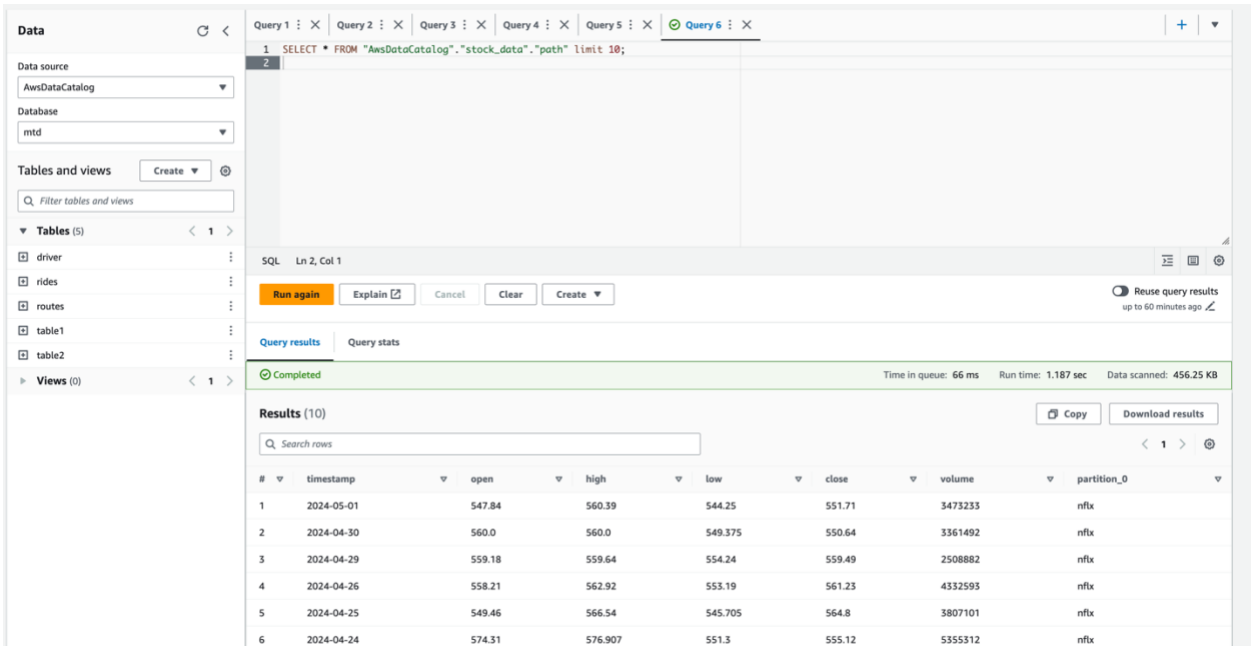

12. Below is the screenshot of all the run by the crawler on a daily basis. The crawler is scheduled to run daily with the help of Eventbridge.



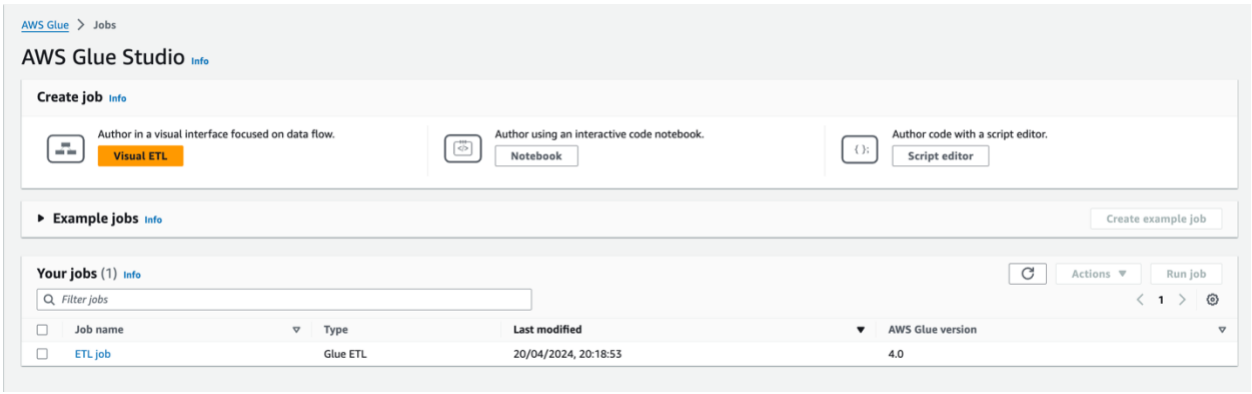

13. These are the partitions that are created within the main table. Each partition containing data for one stock.




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




## AWS GLUE ETL JOBS




## 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()
```

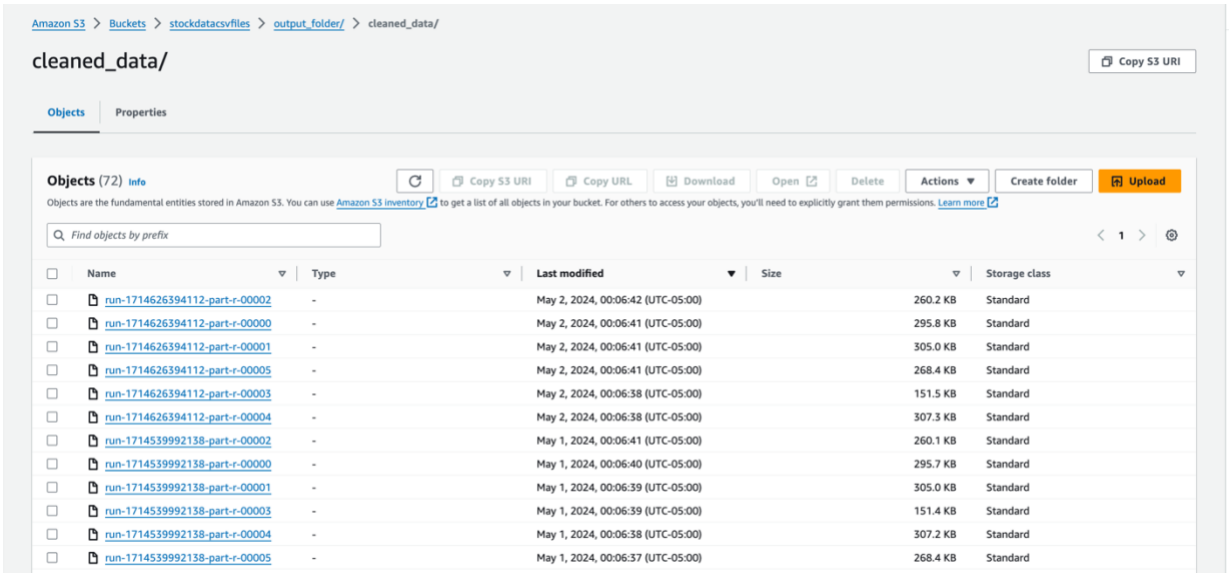
```
glueContext = GlueContext(sc)
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# 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

```
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()
```




## 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.

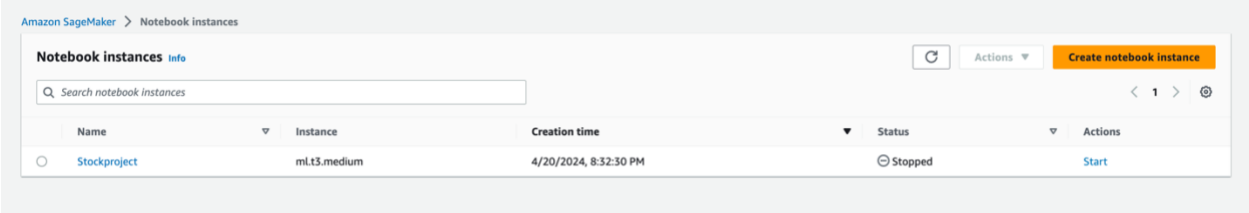
The Python Script does the following:

- 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.

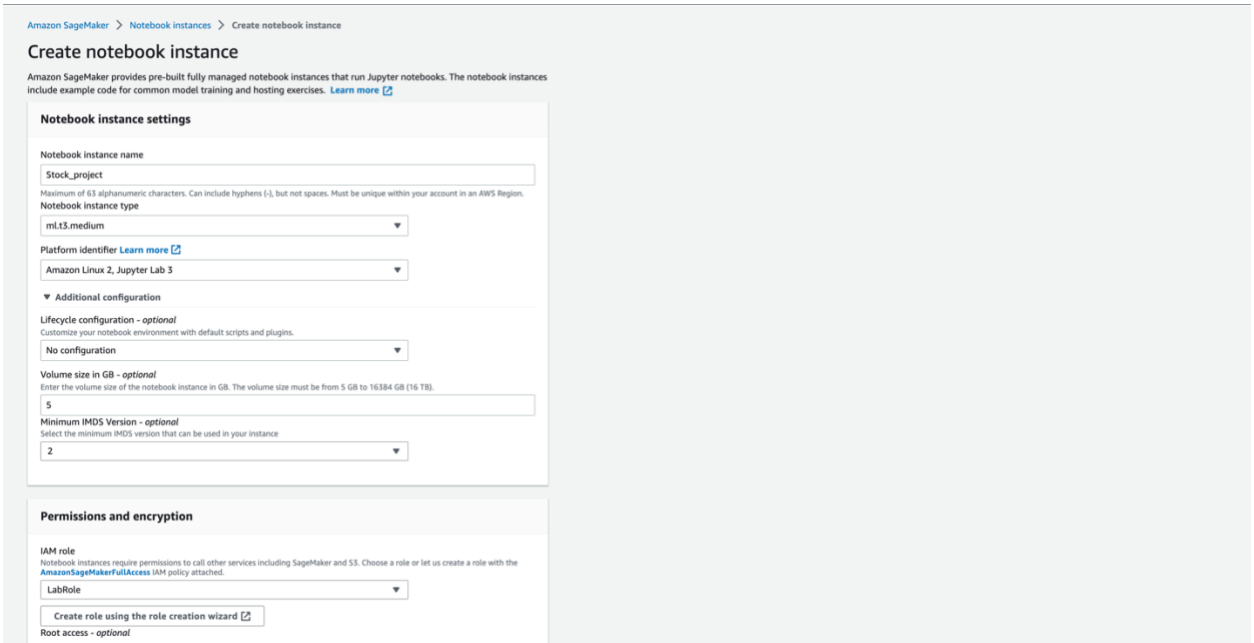
- Storing the output of each stock's predicted price in form of a csv file in the S3 bucket.

**Steps to create a notebook instance:**

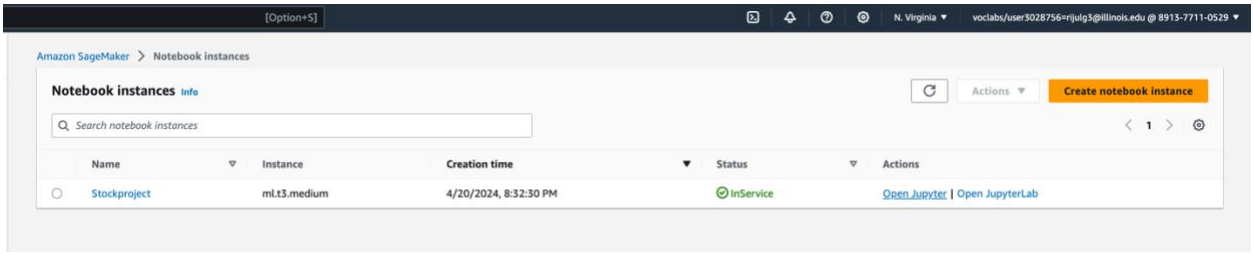
1. Open Amazon Sagemaker service by searching in the search bar.
2. Click on the notebook instance tab on the left-hand menu. It will a page like the one shown in the screenshot below:

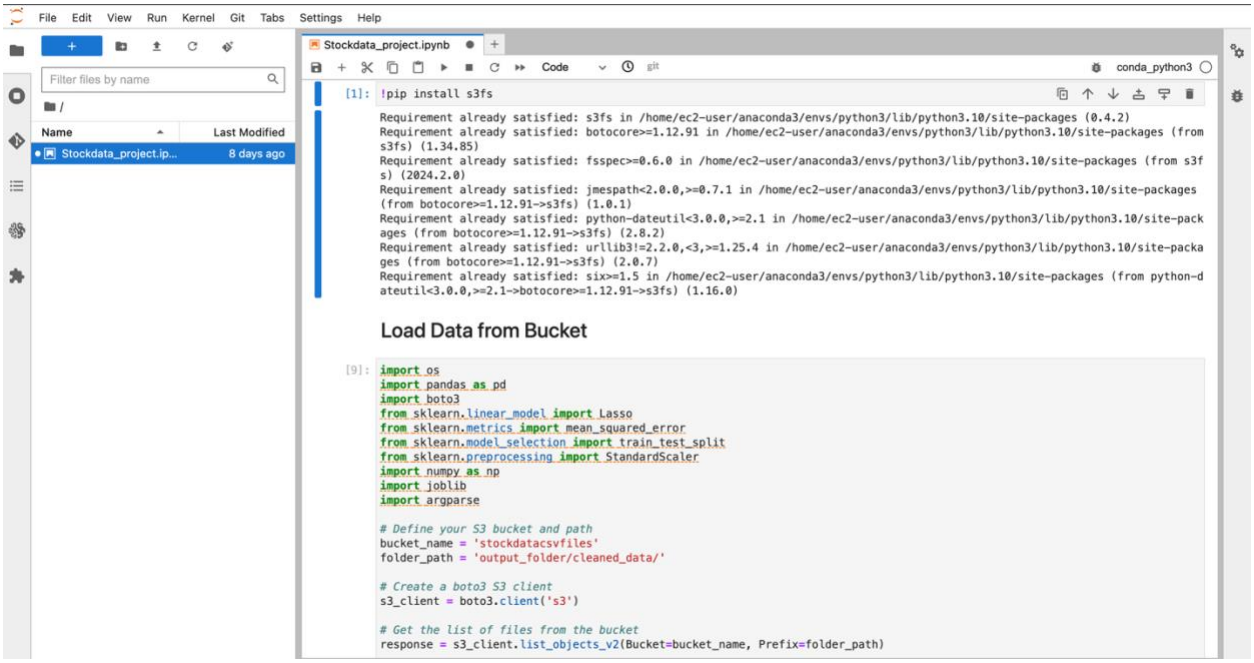



3. Click on Create notebook instance and configure the notebook instance as per your requirement. For our project we chose to utilize the ml.t3. medium instance type as it comes in free tier, and we are using it for development and testing. Additionally, we stuck with default configuration settings for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

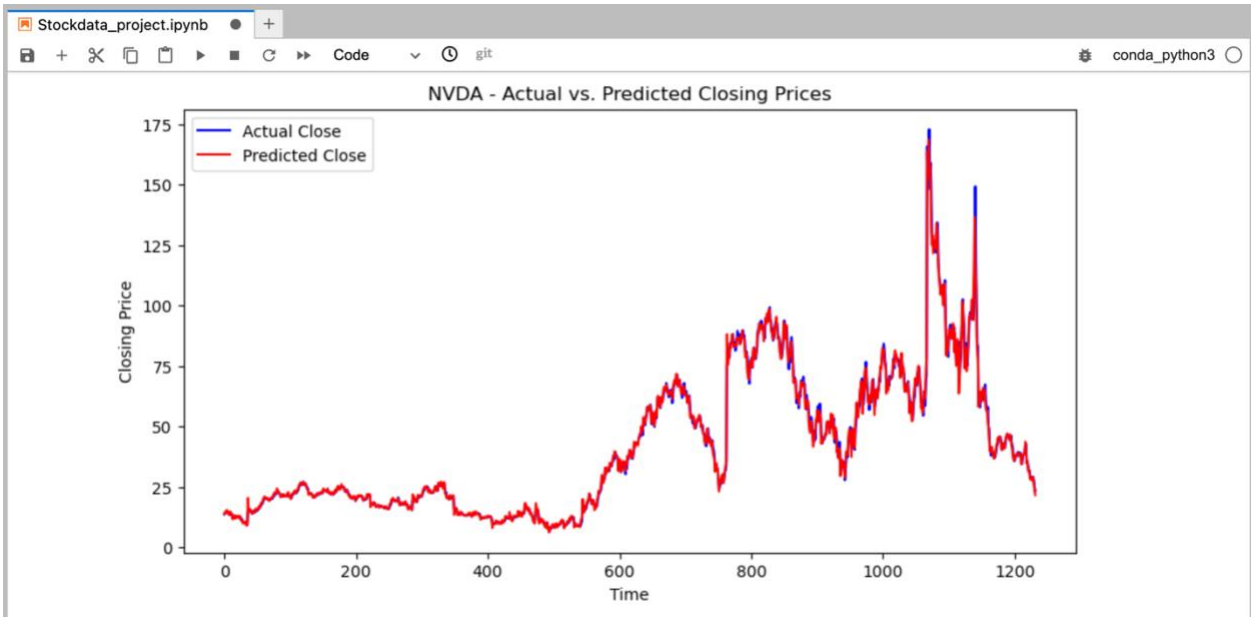


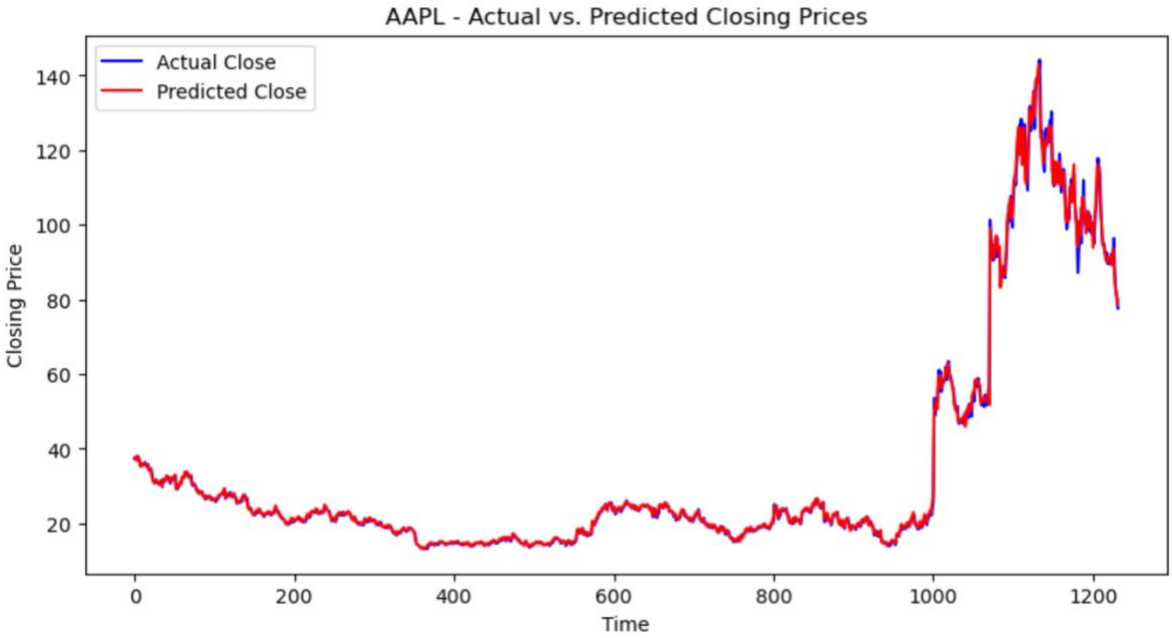

4. Click on create notebook instance button once you finish with the configuration settings. Once the notebook instance is created it will look like in the screenshot below. Now you can click on Open JupyterLab to open the notebook for writing the code.




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





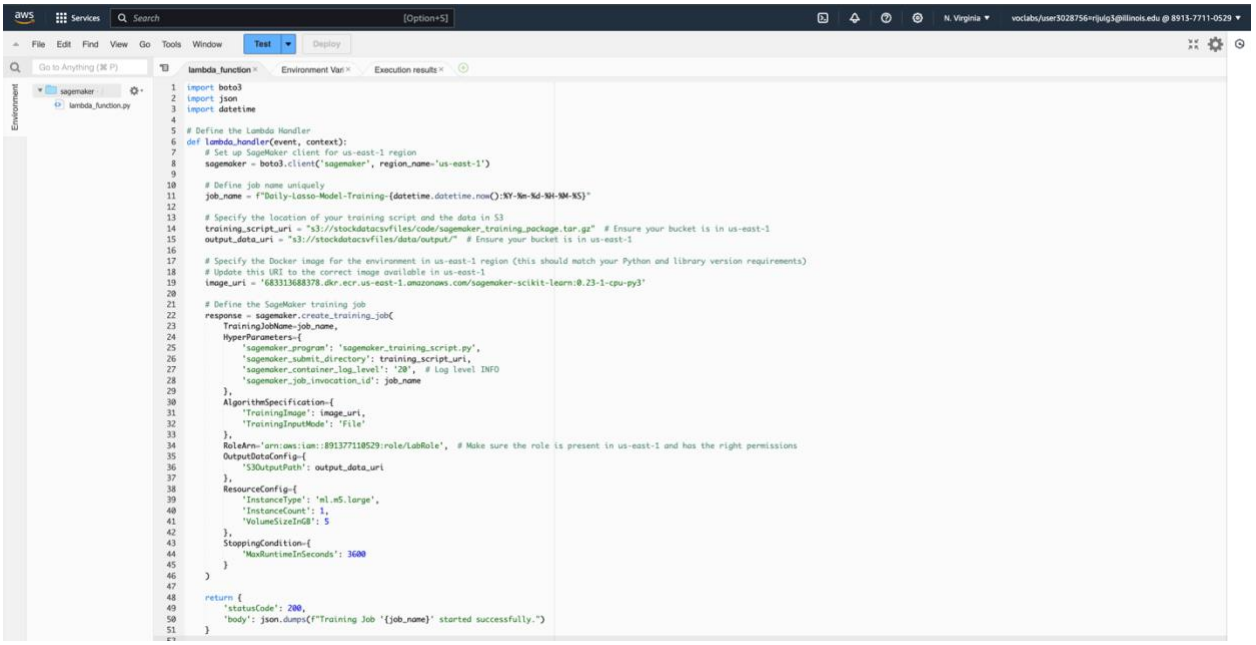

**SAGEMAKER JOB**

**Lambda Script**

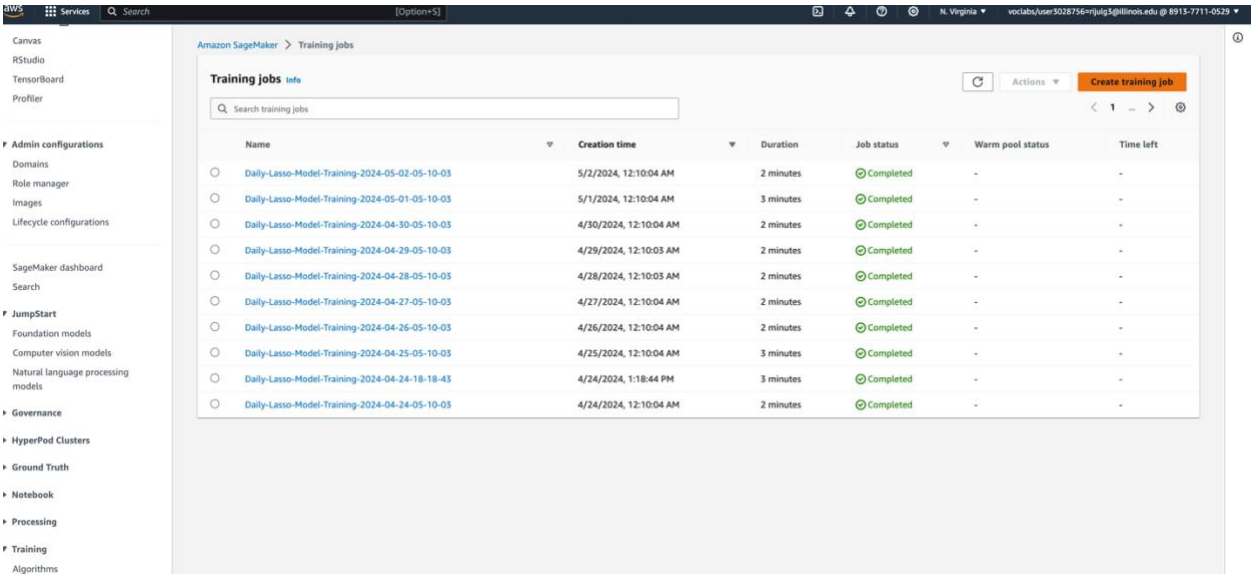
```
import boto3
import json
import datetime

# 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.")
    }
```




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.

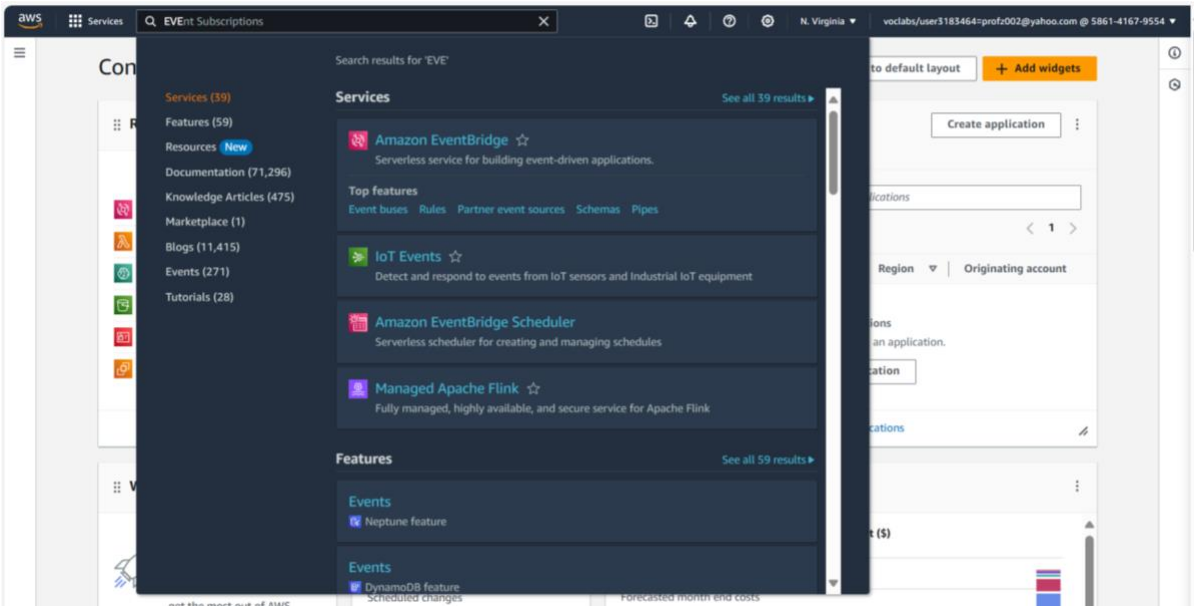



At Every stage, amazon event bridge is used to schedule other AWS services

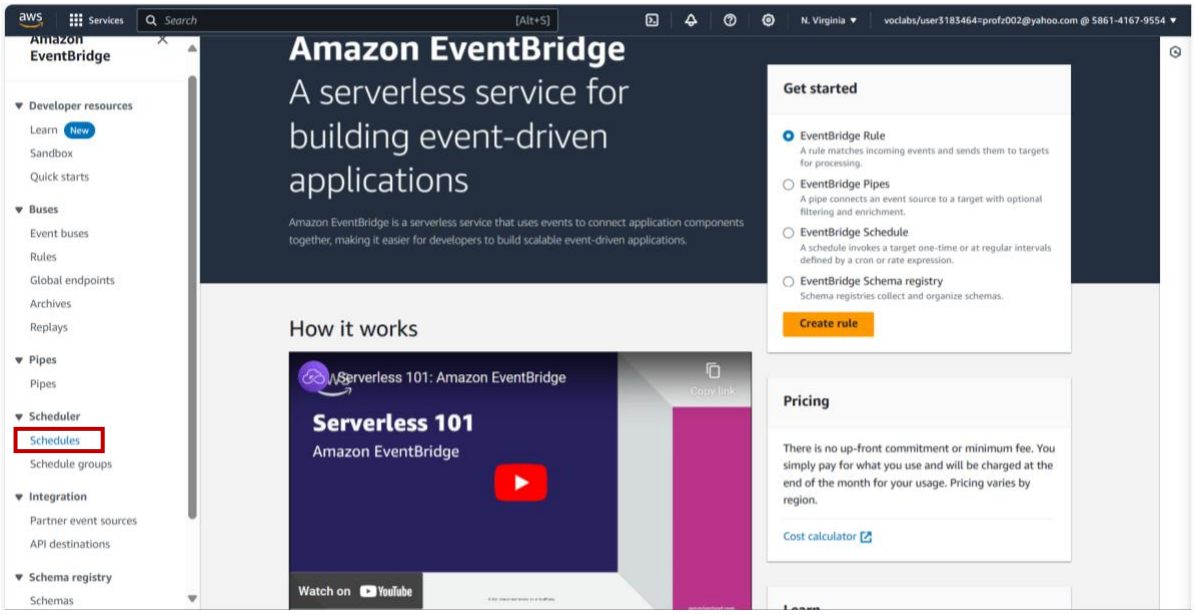
## B. AMAZON EVENTBRIDGE

## Scheduling data extraction

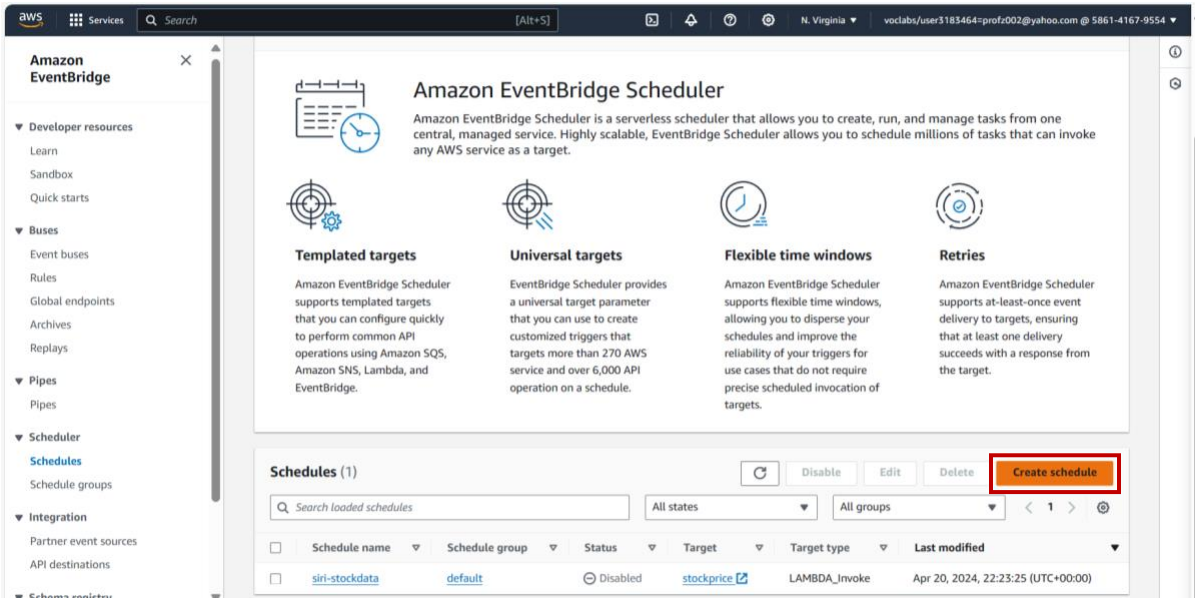
1. Open Amazon EventBridge



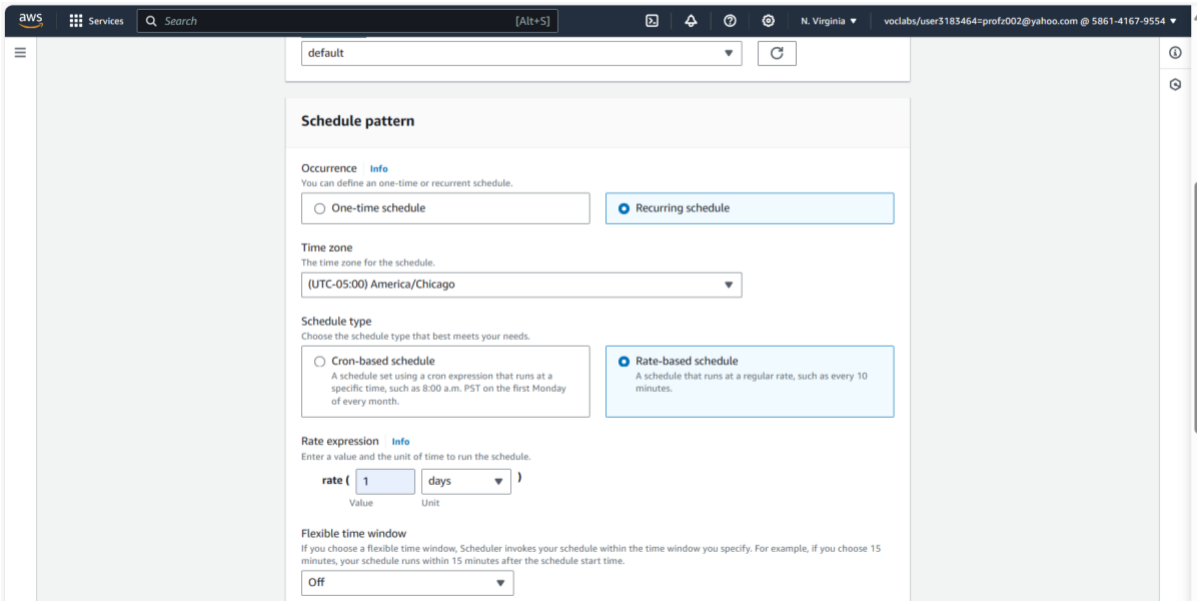

2. Click on Schedules in the left panel



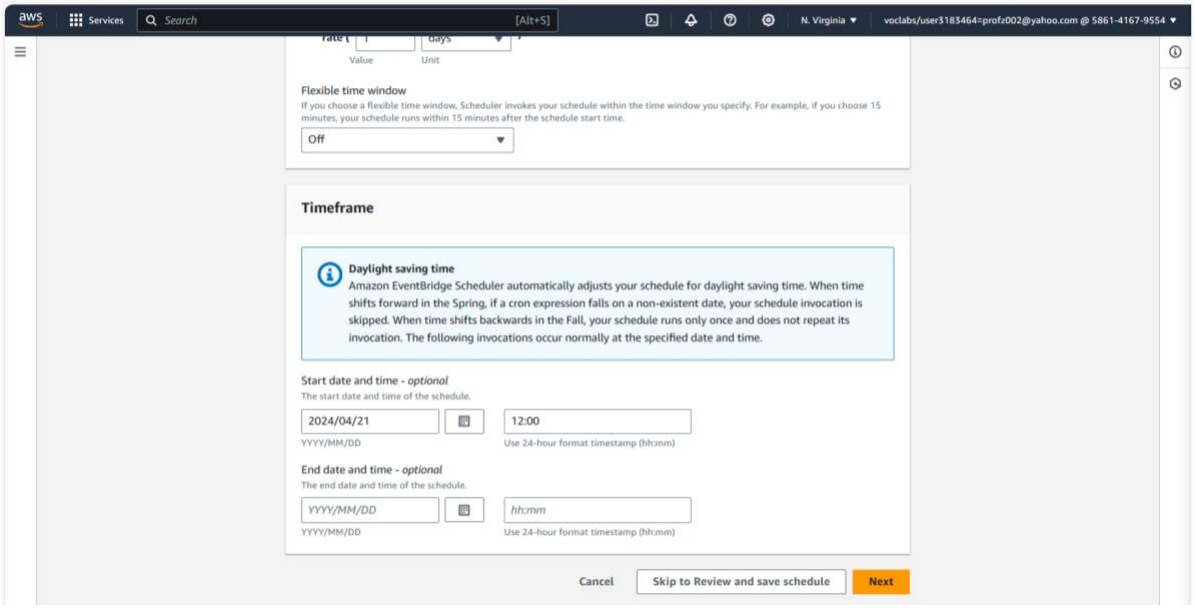
3. Click on ‘Create Schedule’



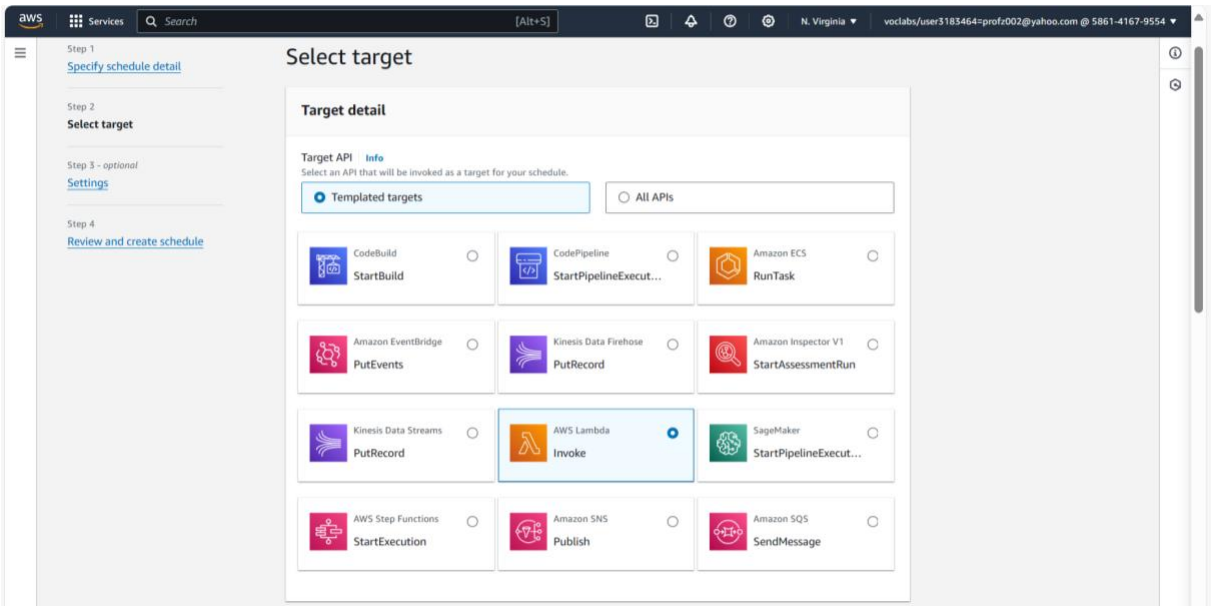

4. Give name, select ‘recurring schedule’ and ‘rate-based schedule’ and give rate as 1 per day. Make flexible time window to ‘off’



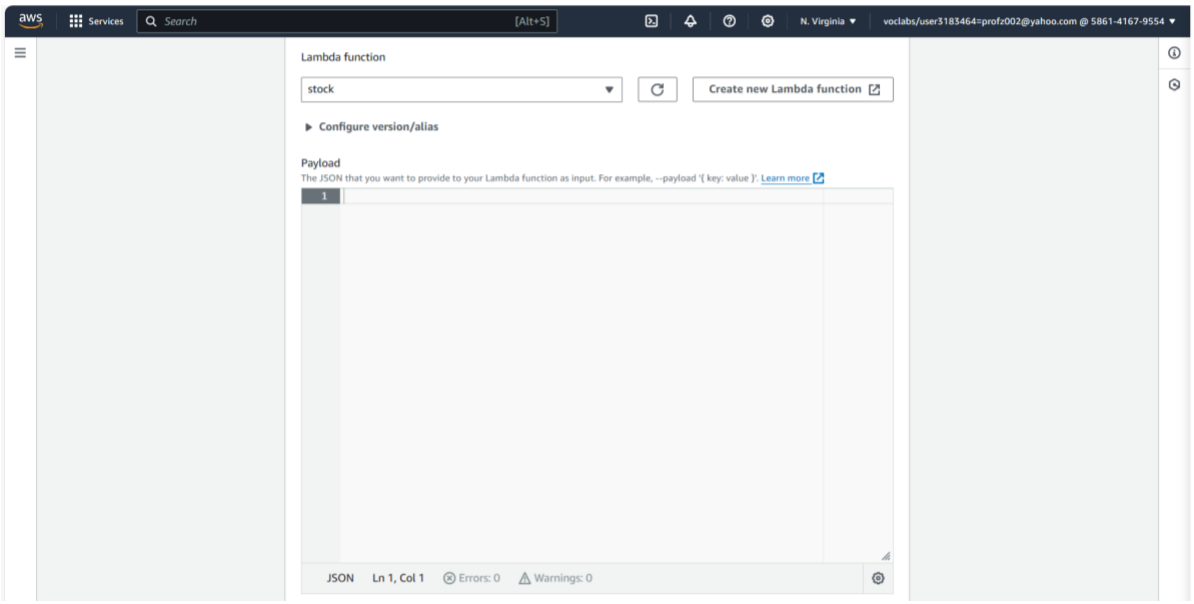

5. Give a time when we want to reload the data everyday. Here it is 12:00 AM



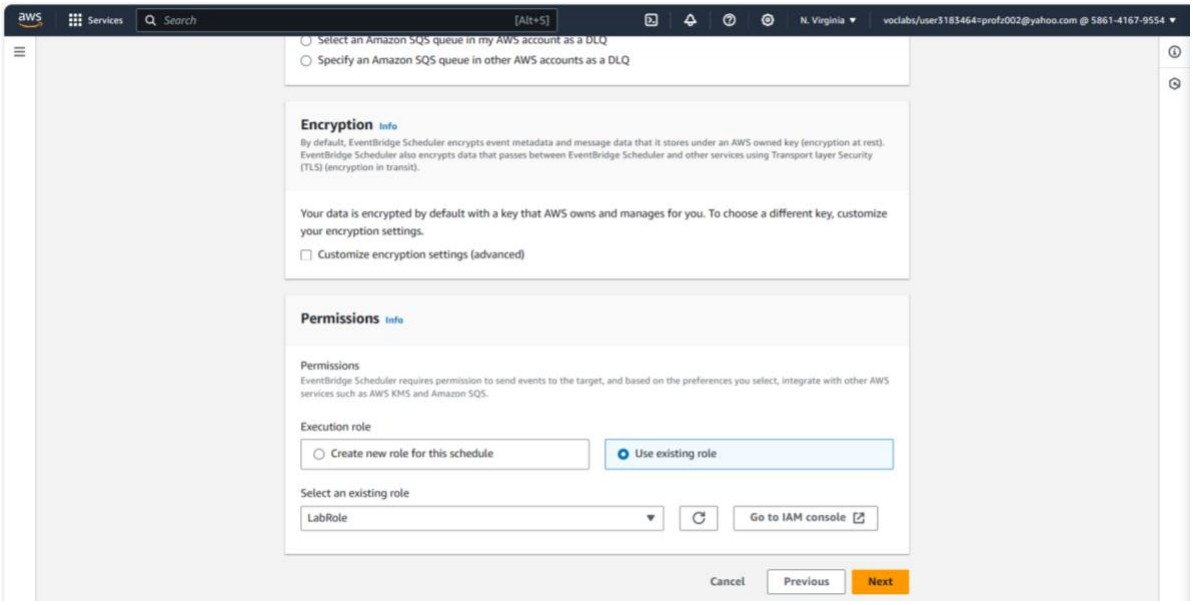

6. After Clicking ‘next’, Select ‘AWS Lambda’



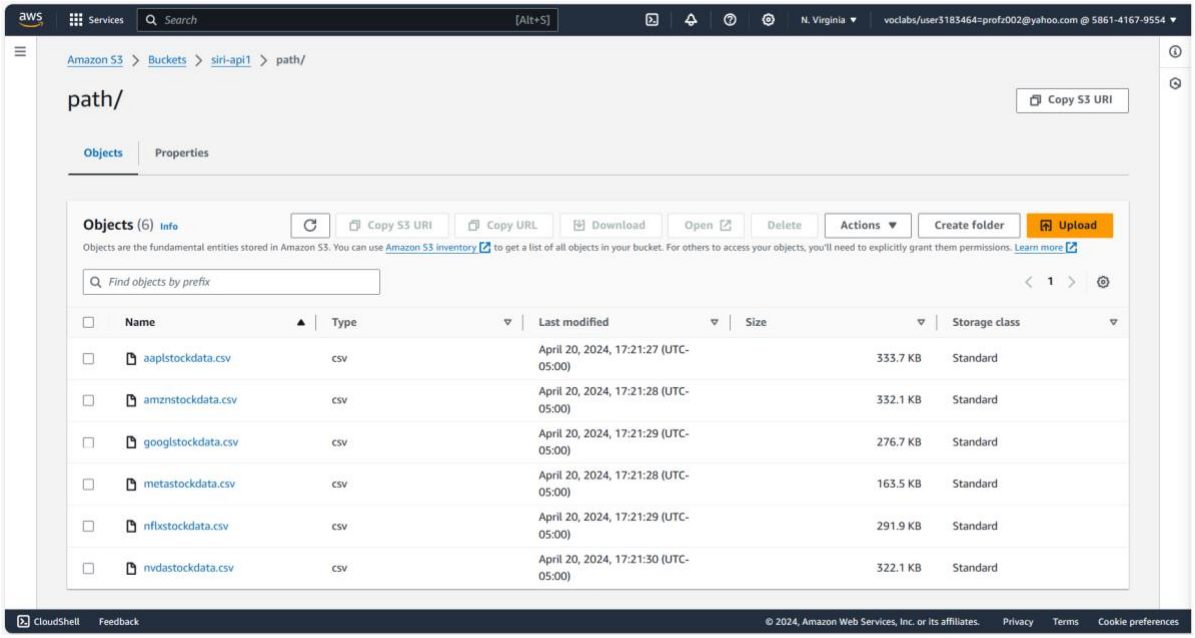
7. Select the lambda function created in the previous step and click next




8. Select IAM role, click next and create schedule

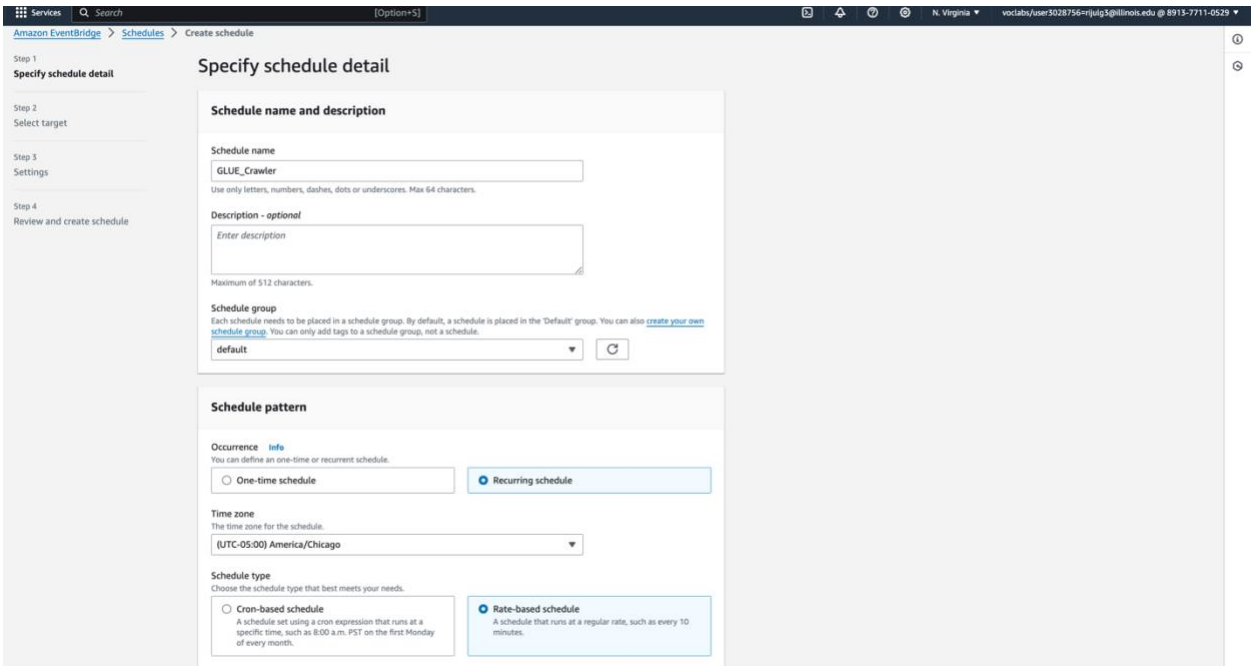



9. The new data will be stored everyday at 12:00AM

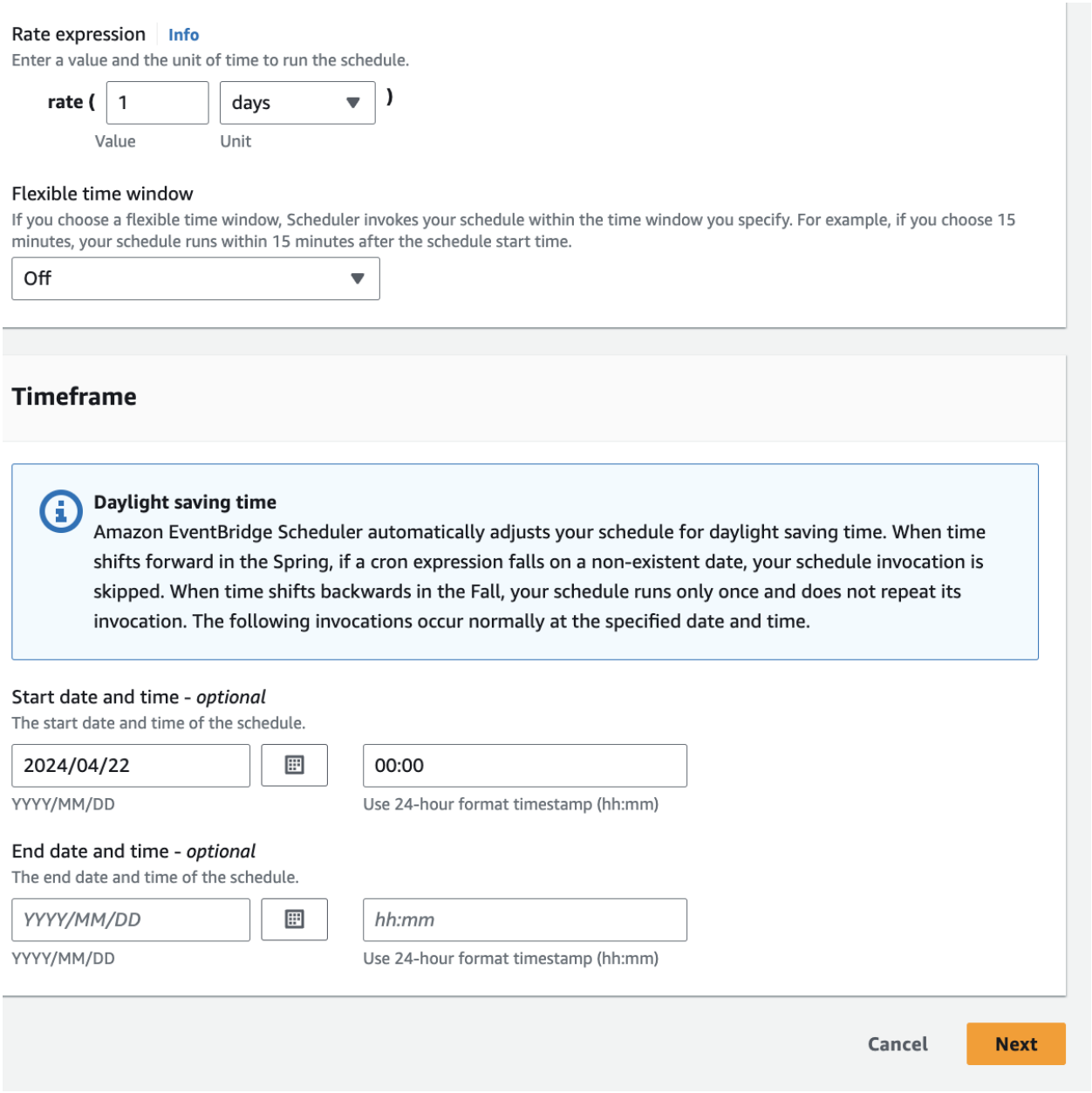



**Scheduling GLUE Crawler for data cleaning**

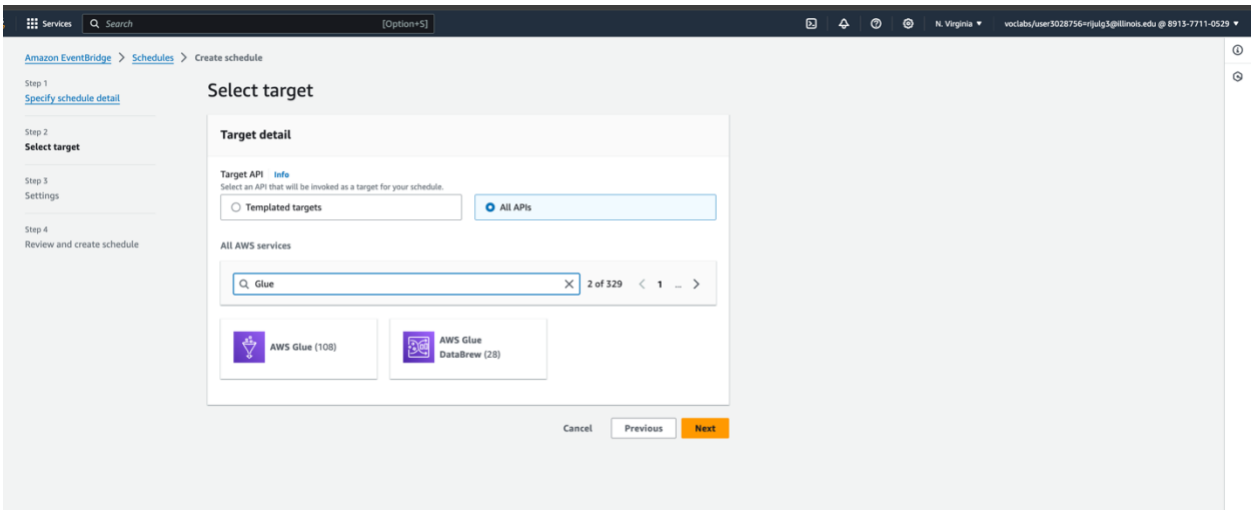
1. Create a new schedule in Eventbridge for scheduling the GLUE crawler. Give the name of the Schedule as GLUE_crawler



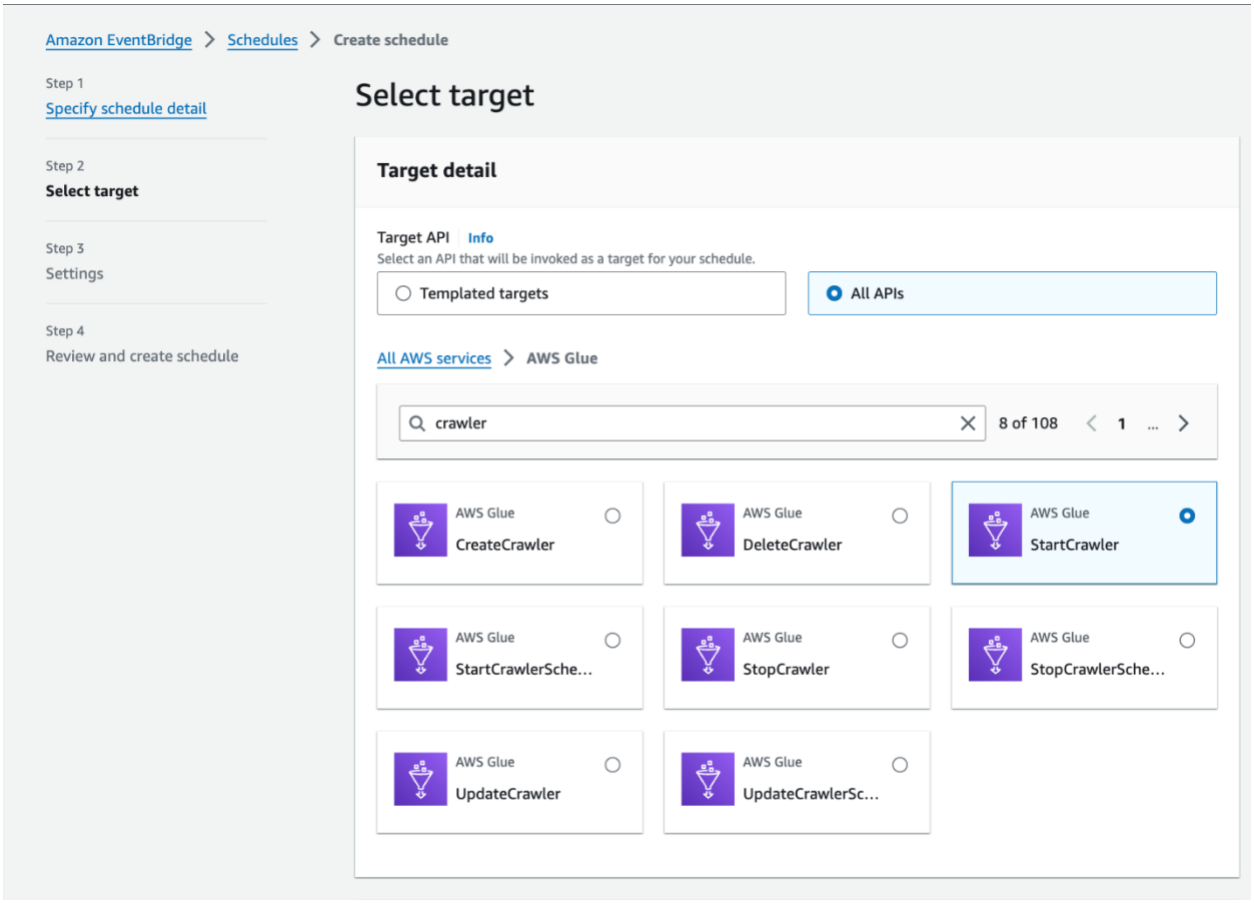

2. Next, we select a recurring schedule as we want to execute the script daily once. So, I select rate expression as 1 and the unit as days to run the crawler 1 time a day.



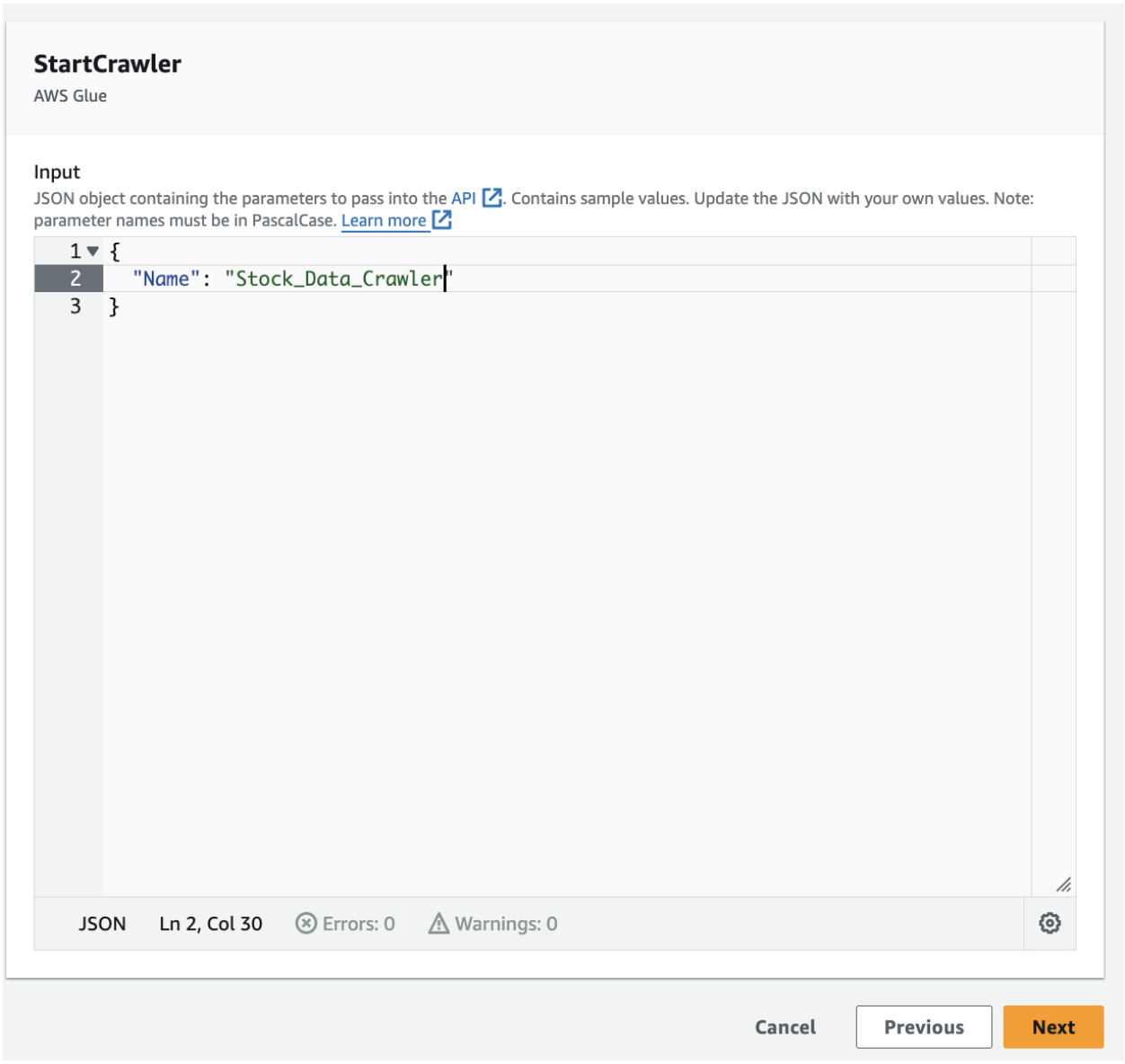

3. Next, we will define the start date and time for the schedule. We chose to not fill the end date but that can all be set for a different use-case.
4. Now we need to select the appropriate API's for scheduling the crawler. For this select All API's and enter crawler in the search bar. Select AWS Glue.



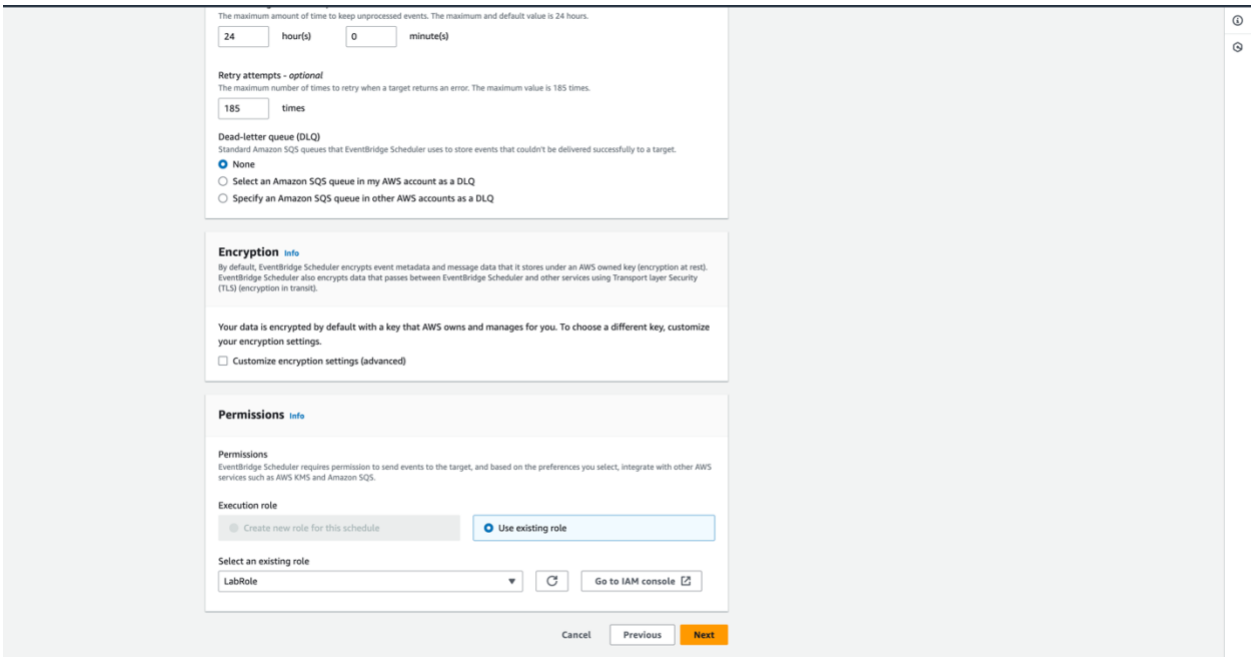

5. Glue has a lot of APIs which serve different purpose. For running the crawler, we need to use Start Crawler API from this list. For this enter crawler in the search bar and select start crawler API.



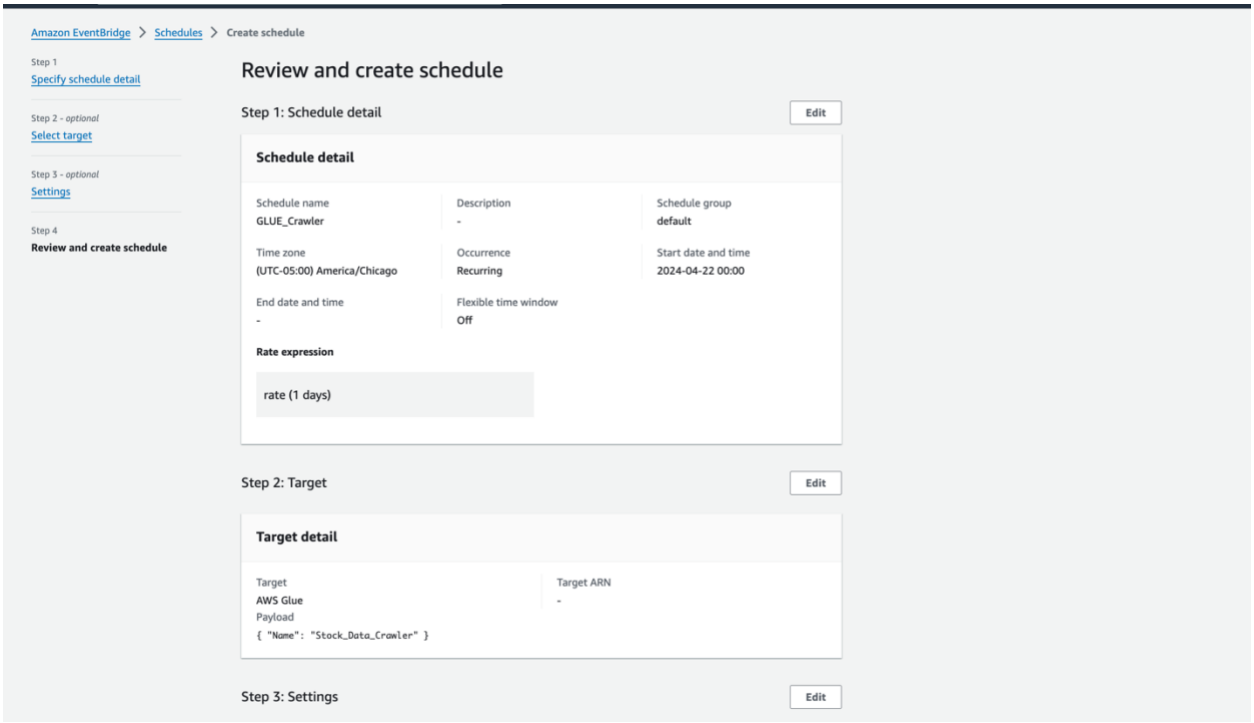

6. Give the name of the crawler to schedule and click on Next.

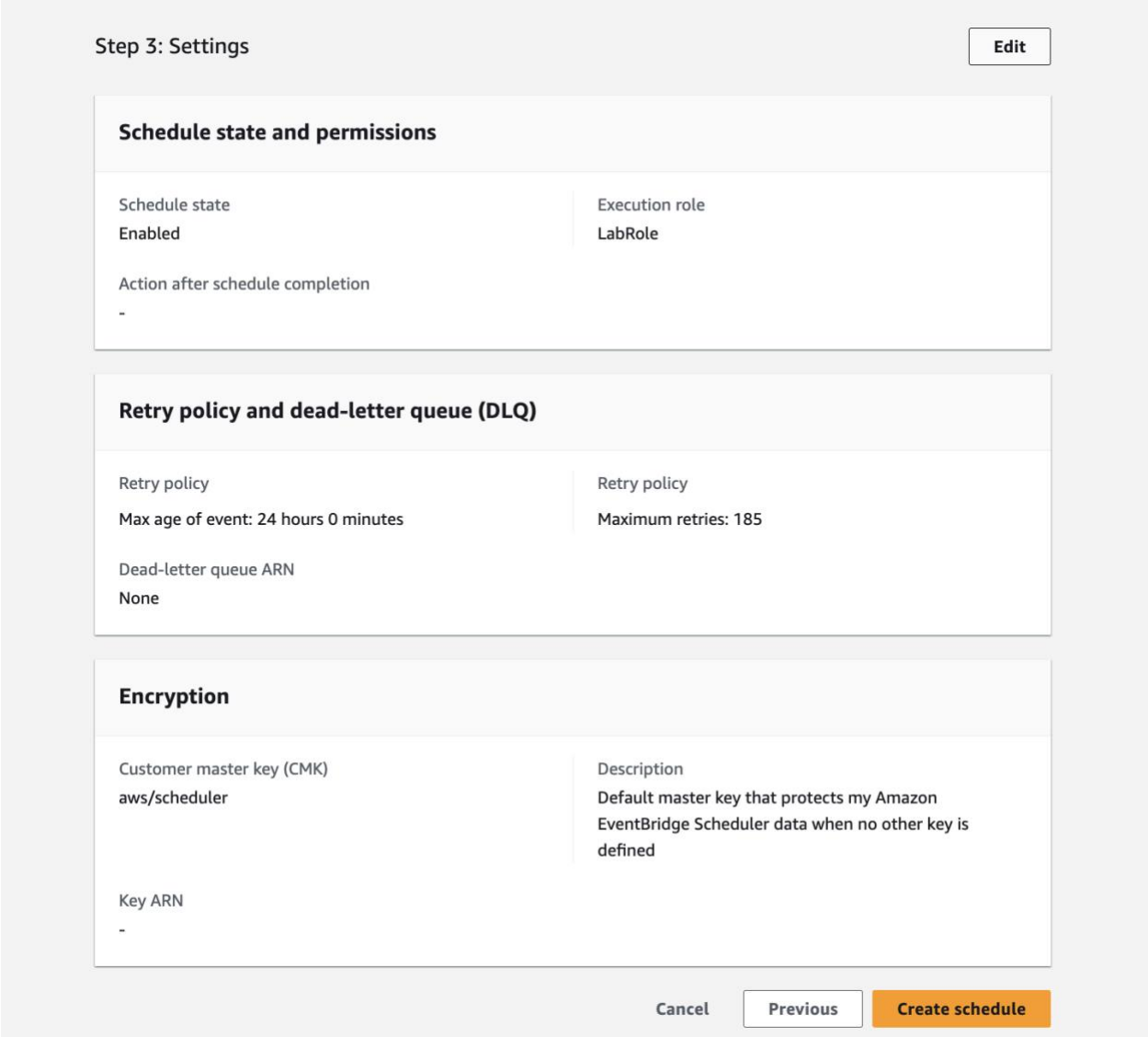



7. Select the IAM role as LabRole. All other settings can be let to be default.



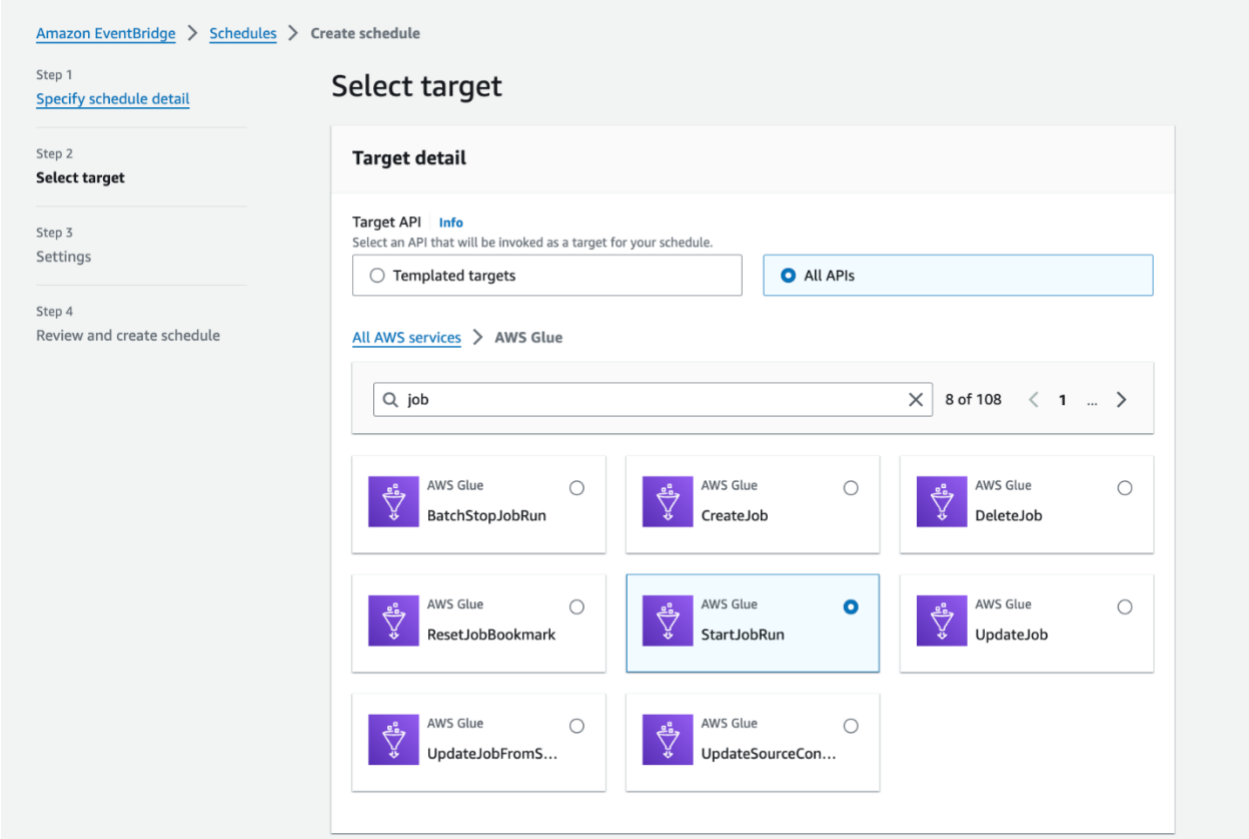

8. Finally review all settings for the schedule and click Create schedule.



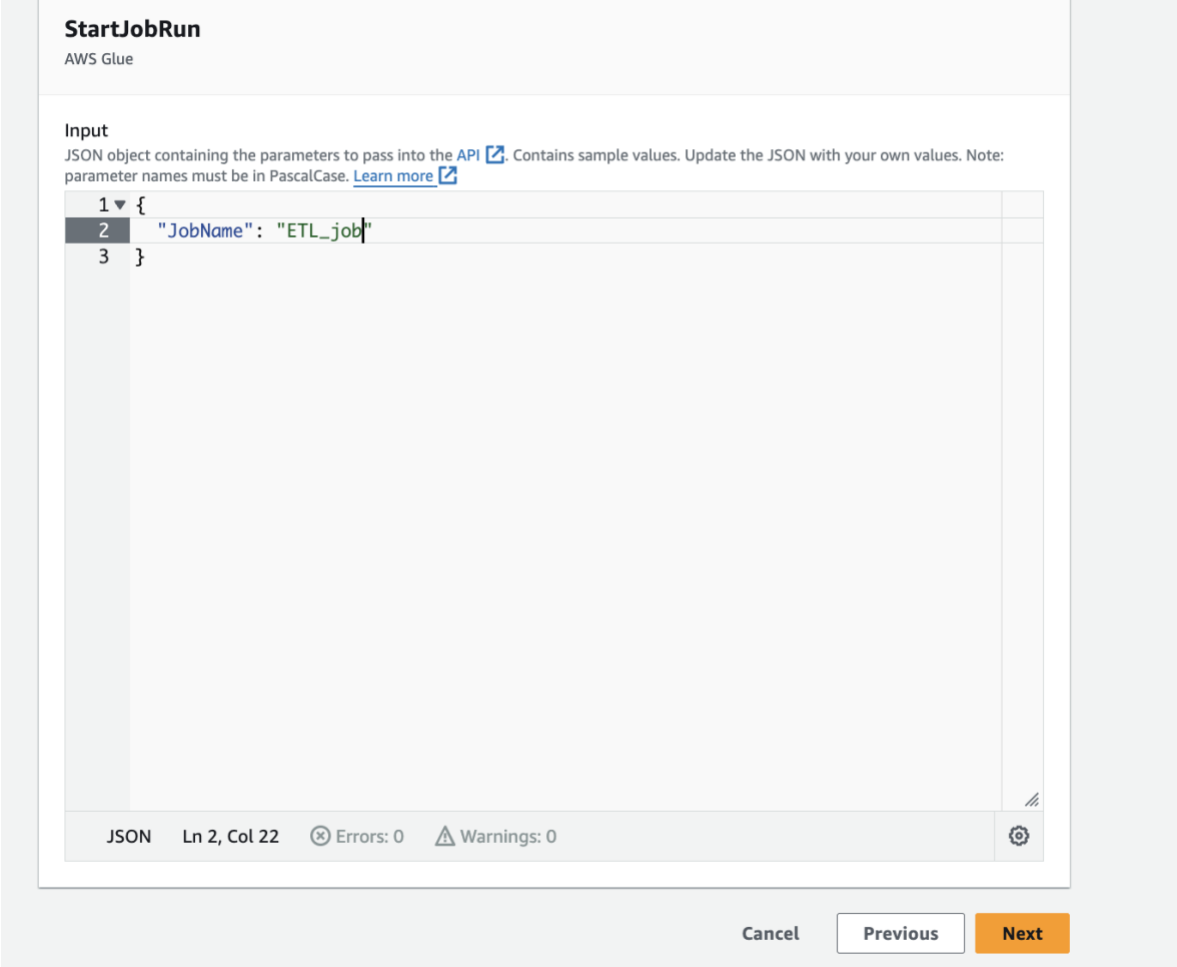



**Scheduling Glue ETL Job**

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.

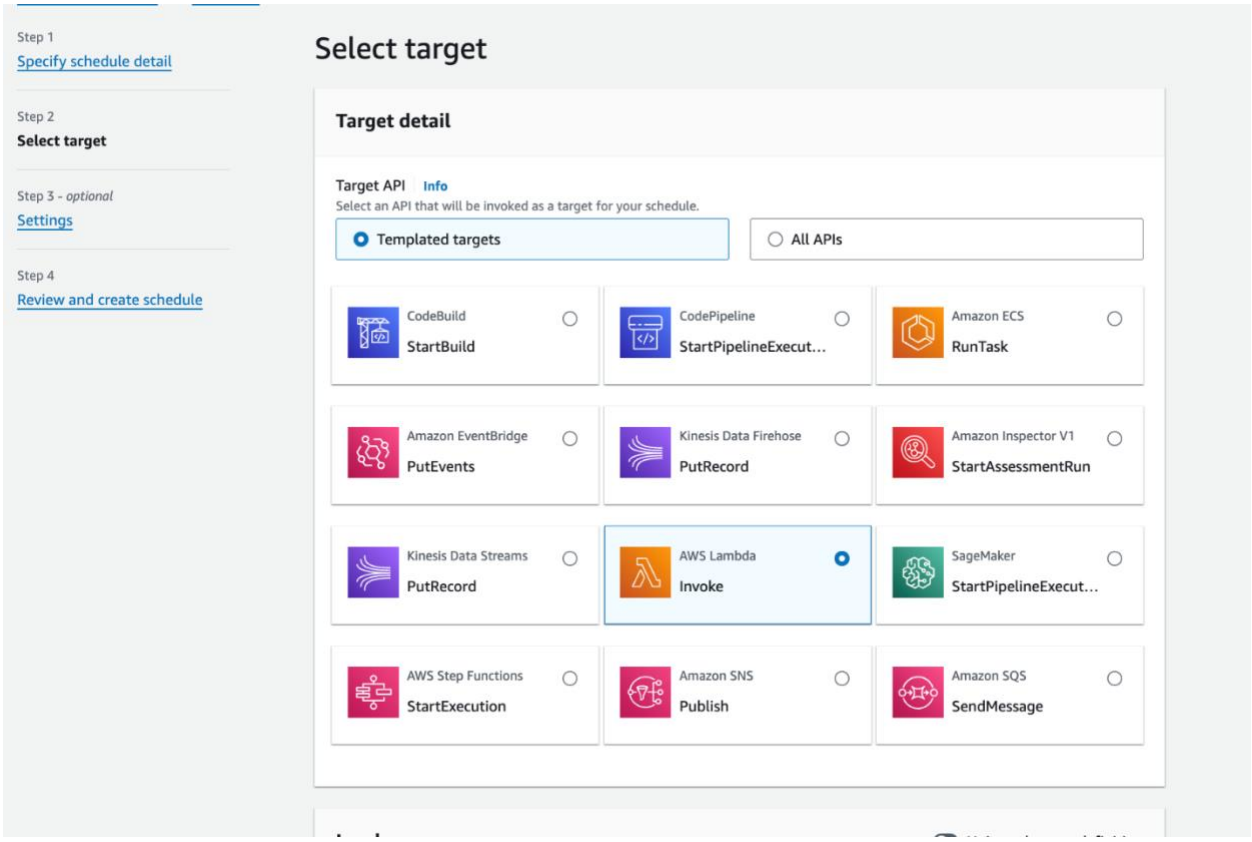



2. Now give the name of the ETL job to be scheduled.

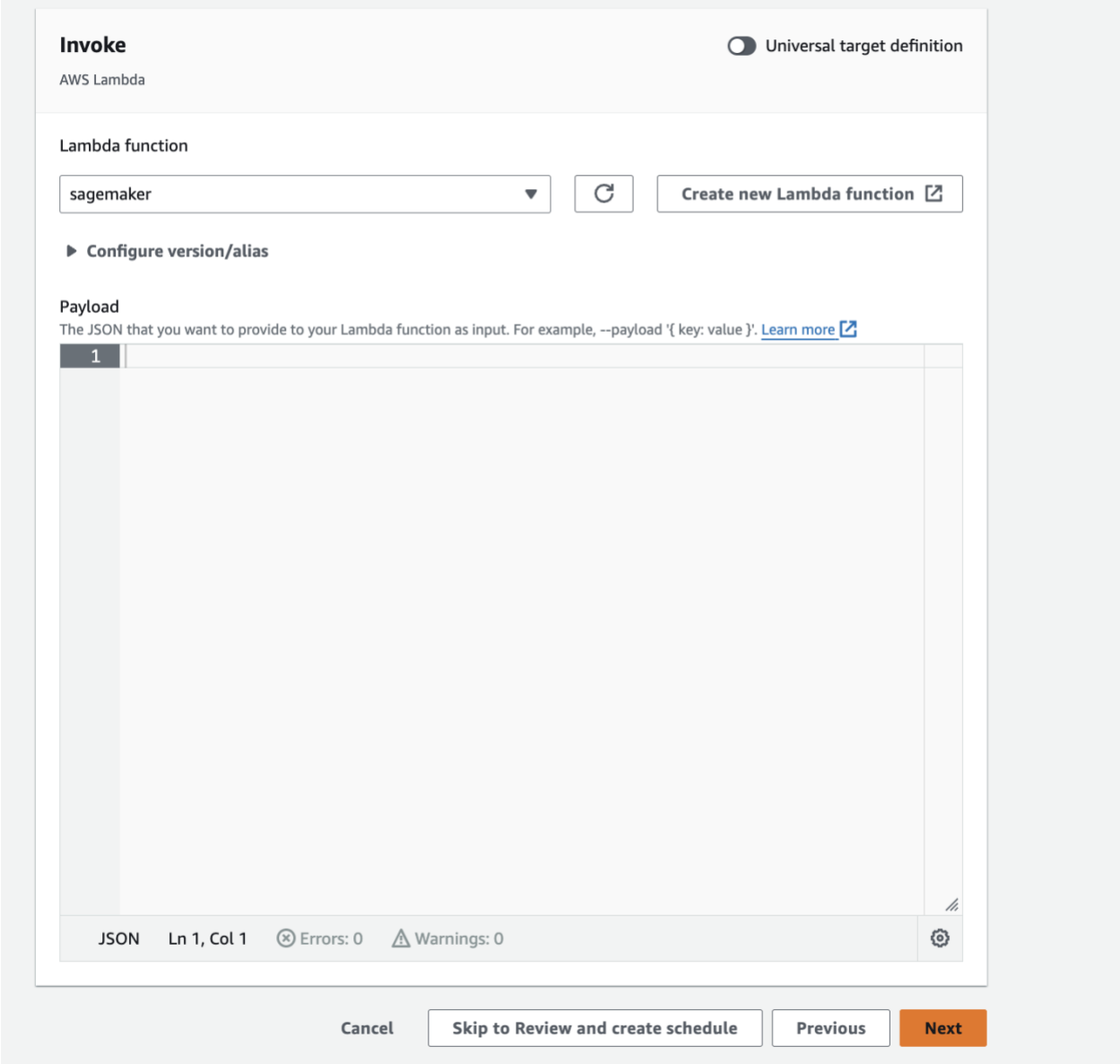



**Scheduling Sagemaker Job Trigger.**

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




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.