Project – 1

Name : Ishan Aakash Patel

Student ID : 146151238

Course : Cloud Programming

Prof : Vahab Shalchian

Serverless Web Scraper with AWS Lambda and S3

Step 1 : Website and Data

1. The Website You Are Scraping

For this project, I am scraping stock market data from Groww (https://groww.in). Groww is a financial platform that provides stock prices, company details, and market insights.

Why did I choose Groww?

- It provides real-time stock information.
- It is a reliable source for stock market data.
- It displays key financial details like open price, previous close, and market cap for each company.

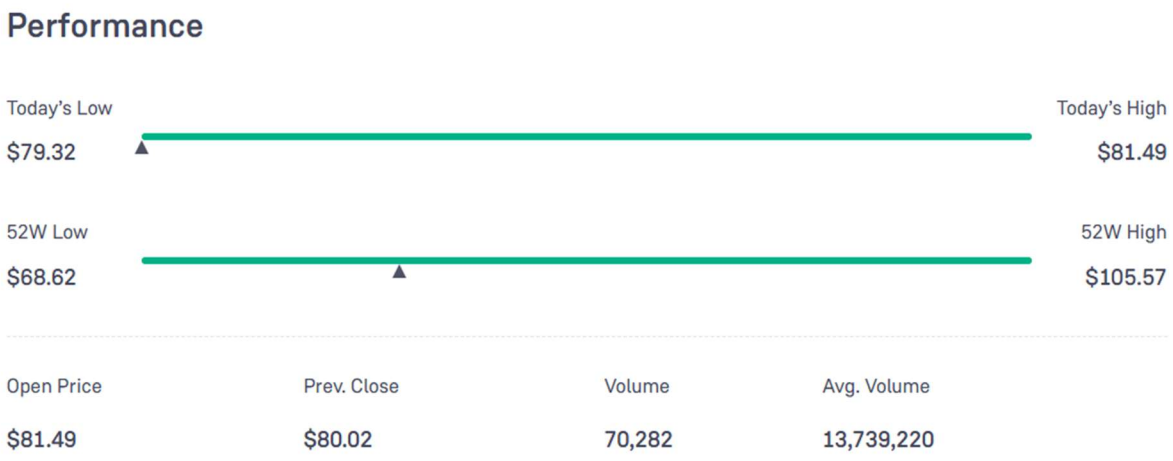
2. The Specific Data Being Retrieved

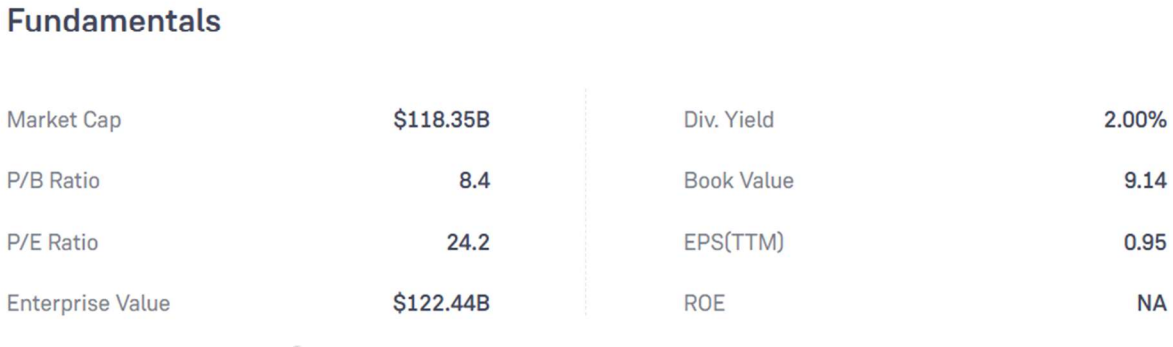
From each stock page, my script extracts:

1. Company Name – Name of the company whose stock is being analyzed.
2. Open Price – The price of the stock when the market opened.
3. Previous Close Price – The stock's closing price from the previous trading session.
4. Market Cap – The total market value of the company's outstanding shares.

Why these data points?

- These values help in tracking stock performance.
- Investors use this information to make financial decisions.
- It allows users to compare different stocks based on real-time data.





3. Why This Website and Data?

- Automation – Instead of manually checking stock prices, your AWS Lambda function does it automatically.
- Data Storage – The extracted data is saved in Amazon S3, making it easy to access and analyze later.
- Scalability – If needed, you can expand this scraper to collect more details (e.g., stock trends, historical data).

Step 2 : Scraper Functionality

1. How Your Scraper Collects Data
This scraper is a Python script that automates the extraction of stock market data from Groww. It follows these steps:

1. Sends an HTTP request to the stock page (e.g., https://groww.in/us-stocks/nke).
2. Parses the HTML content using BeautifulSoup.
3. Extracts specific stock details like Company Name, Open Price, Previous Close Price, and Market Cap.

Code Implementation:
My script first defines a list of stock URLs that need to be scraped:

```
urls = [
    "https://groww.in/us-stocks/nke", "https://groww.in/us-stocks/ko",
    "https://groww.in/us-stocks/msft", "https://groww.in/us-stocks/axp",
    "https://groww.in/us-stocks/amgn", "https://groww.in/us-stocks/aapl",
    "https://groww.in/us-stocks/ba", "https://groww.in/us-stocks/csco",
    "https://groww.in/us-stocks/gs", "https://groww.in/us-stocks/ibm",
    "https://groww.in/us-stocks/intc", "https://groww.in/us-stocks/jpm",
    "https://groww.in/us-stocks/mcd", "https://groww.in/us-stocks/crm",
    "https://groww.in/us-stocks/dis"
]
```

Then, it fetches the webpage content using the requests library while also setting a User-Agent header. This makes the request look like it's coming from a real browser:

```
response = requests.get(url, headers={"User-Agent": "Mozilla/5.0"})
soup = BeautifulSoup(response.content, "html.parser")
```

2. How this Scraper Processes Data
Once the HTML content is retrieved, the scraper looks for specific data points:
Extracting Company Name:

- The script searches for a table cell (<td>) with the text "Organisation".
- If found, it grabs the next <td> that contains the actual company name.

```
company = soup.find("td", string="Organisation")
company_name = company.find_next("td").text.strip() if company else "Not
Found"
```

Extracting Market Cap:

- The script follows the same approach for Market Cap by finding the next <td> after "Market Cap".

```
market_cap = soup.find("td", string="Market Cap")
market_cap_value = market_cap.find_next("td").text.strip() if market_cap
else "Not Found"
```

Extracting Open Price & Previous Close Price:

- These values are located in <td> elements with the class "bodyLargeHeavy".
- The script collects all such elements and extracts the first two as Open Price and Previous Close.

```
rows = soup.find_all("td", class_="bodyLargeHeavy")
open_price = rows[0].text.strip() if len(rows) > 0 else "Not Found"
prev_close = rows[1].text.strip() if len(rows) > 1 else "Not Found"
```

Handling Errors:

- If anything goes wrong (e.g., the website changes, data is missing), the script captures the error instead of crashing.
- This ensures that the scraper continues running even if some pages fail.

```
except Exception as e:
    return {"URL": url, "Error": str(e)}
```

3. How the Data is Stored

After scraping stock data from multiple pages, the script:

1. Converts the collected data into JSON format.
2. Uploads the JSON file to an Amazon S3 bucket for storage.

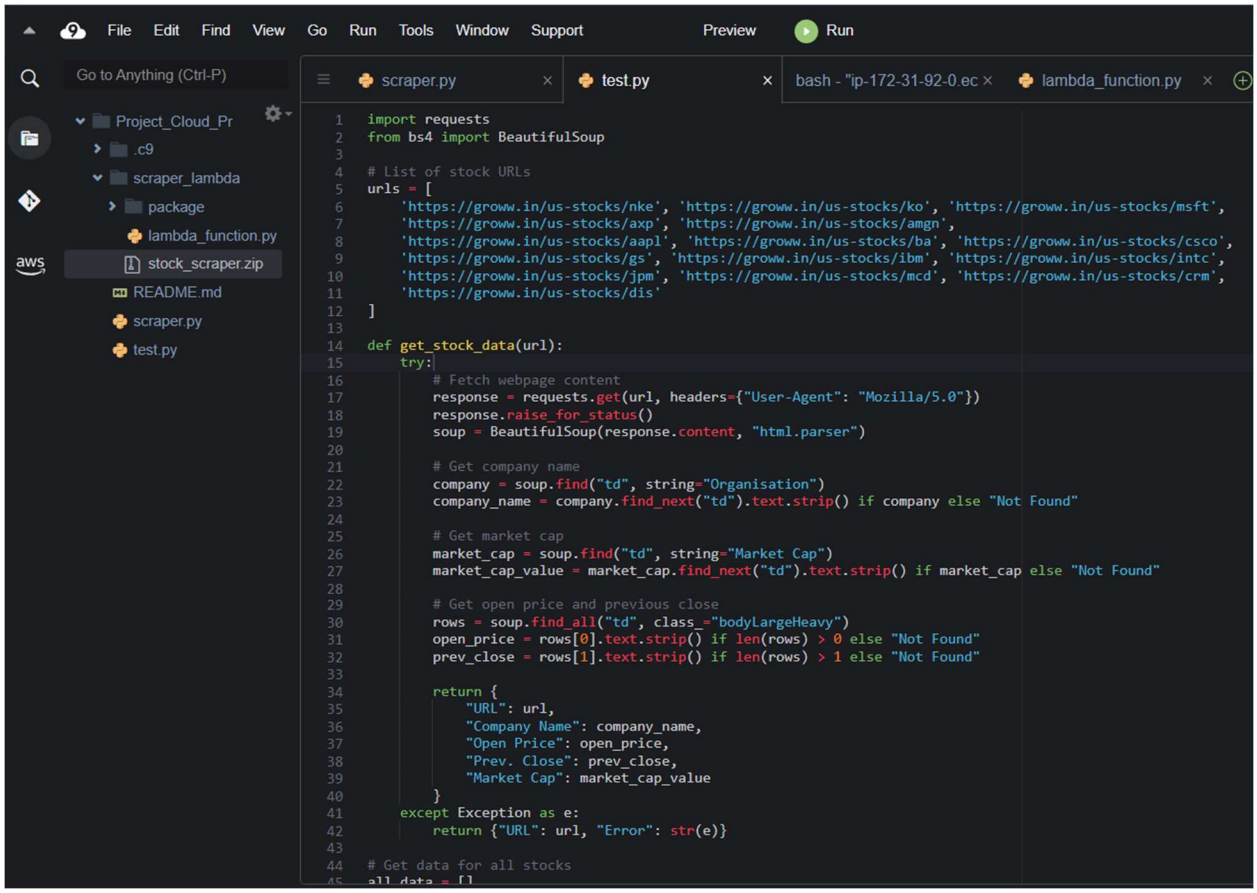
Code for Storing Data in S3:

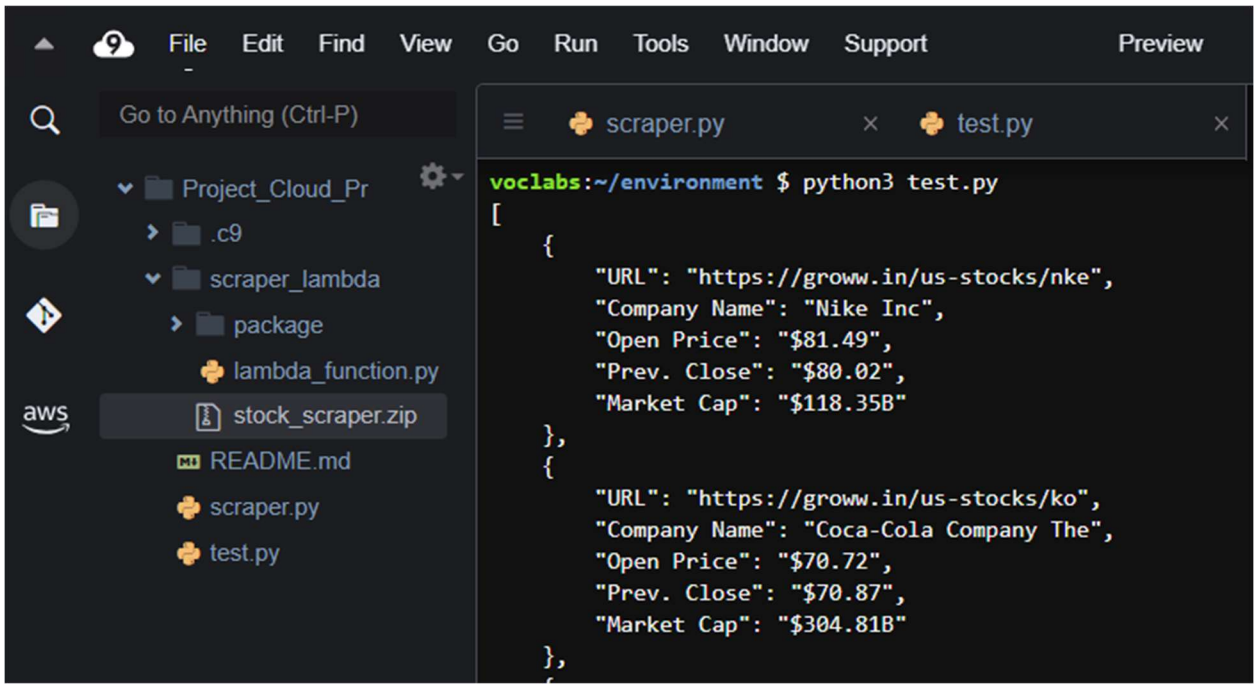
```
# Convert data to JSON format
json_data = json.dumps(all_data, indent=4)

# Define the file name
file_name = "stock_data.json"

# Upload to S3 bucket
s3.put_object(Body=json_data, Bucket=S3_BUCKET, Key=file_name)
```

Why store data in S3?

- Scalability: Data is stored securely and can be accessed from anywhere.
- Automation: Future Lambda functions can process and analyze this data.
- Reliability: AWS S3 ensures data is not lost.

I am using Cloud 9 here for the compilation of the code.


File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (Ctrl-P)
scraper.py
test.py
bash - "ip-172-31-92-0.ec
lambda_function.py
Project_Cloud_Pr
.c9
scraper_lambda
package
lambda_function.py
stock_scraper.zip
README.md
scraper.py
test.py
import requests
from bs4 import BeautifulSoup
# List of stock URLs
urls = [
'https://groww.in/us-stocks/nke', 'https://groww.in/us-stocks/ko', 'https://groww.in/us-stocks/msft',
'https://groww.in/us-stocks/axp', 'https://groww.in/us-stocks/amgn',
'https://groww.in/us-stocks/aapl', 'https://groww.in/us-stocks/ba', 'https://groww.in/us-stocks/csco',
'https://groww.in/us-stocks/gs', 'https://groww.in/us-stocks/ibm', 'https://groww.in/us-stocks/intc',
'https://groww.in/us-stocks/jpm', 'https://groww.in/us-stocks/mcd', 'https://groww.in/us-stocks/crm',
'https://groww.in/us-stocks/dis'
]
def get_stock_data(url):
try:
# Fetch webpage content
response = requests.get(url, headers={"User-Agent": "Mozilla/5.0"})
response.raise_for_status()
soup = BeautifulSoup(response.content, "html.parser")
# Get company name
company = soup.find("td", string="Organisation")
company_name = company.find_next("td").text.strip() if company else "Not Found"
# Get market cap
market_cap = soup.find("td", string="Market Cap")
market_cap_value = market_cap.find_next("td").text.strip() if market_cap else "Not Found"
# Get open price and previous close
rows = soup.find_all("td", class_="bodyLargeHeavy")
open_price = rows[0].text.strip() if len(rows) > 0 else "Not Found"
prev_close = rows[1].text.strip() if len(rows) > 1 else "Not Found"
return {
"URL": url,
"Company Name": company_name,
"Open Price": open_price,
"Prev. Close": prev_close,
"Market Cap": market_cap_value
}
except Exception as e:
return {"URL": url, "Error": str(e)}
Get data for all stocks

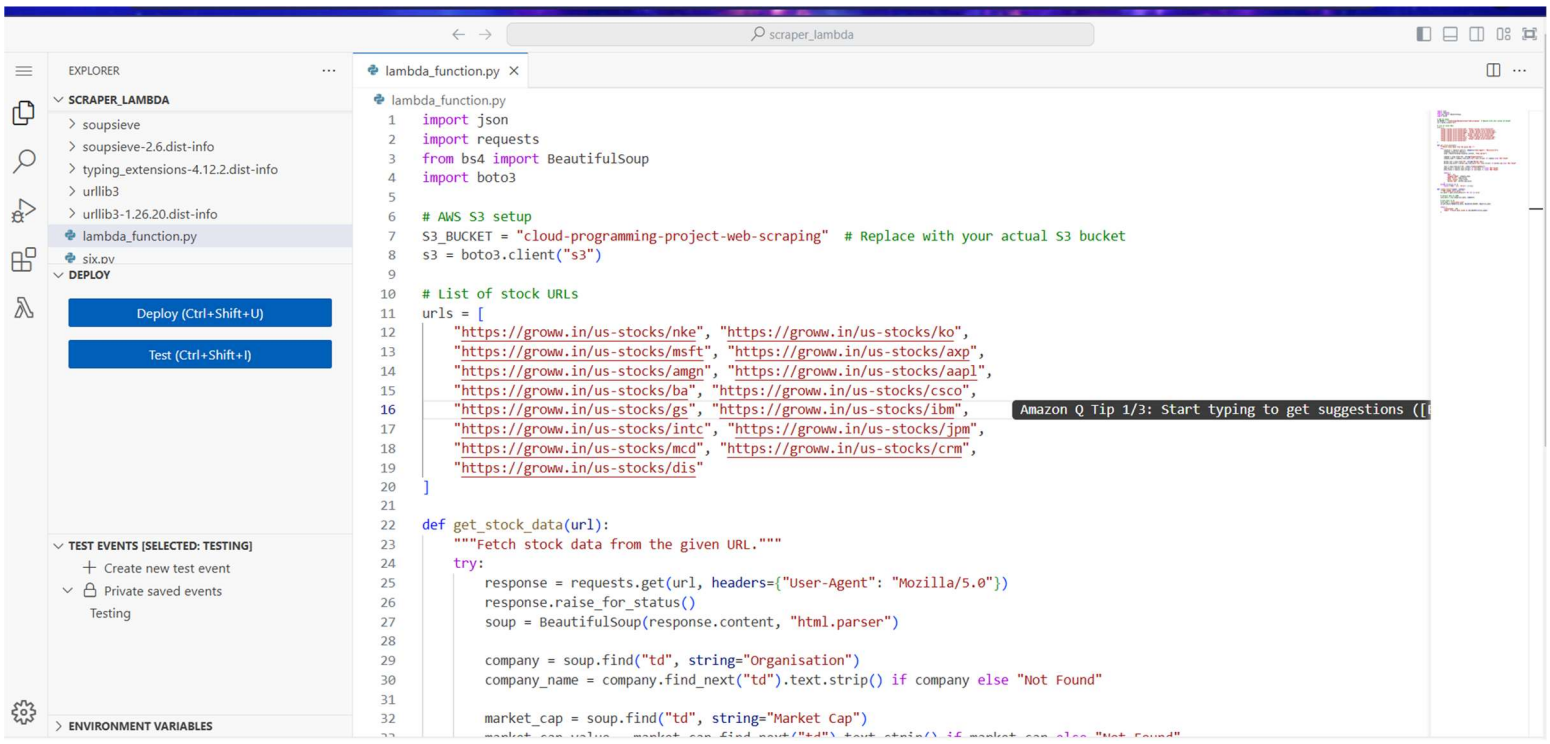

File Edit Find View Go Run Tools Window Support Preview
Go to Anything (Ctrl-P)
scraper.py
test.py
Project_Cloud_Pr
.c9
scraper_lambda
package
lambda_function.py
stock_scraper.zip
README.md
scraper.py
test.py
voclabs:~/environment $ python3 test.py
[
{
"URL": "https://groww.in/us-stocks/nke",
"Company Name": "Nike Inc",
"Open Price": "$81.49",
"Prev. Close": "$80.02",
"Market Cap": "$118.35B"
},
{
"URL": "https://groww.in/us-stocks/ko",
"Company Name": "Coca-Cola Company The",
"Open Price": "$70.72",
"Prev. Close": "$70.87",
"Market Cap": "$304.81B"
},

Step 3 : Lambda Function Code

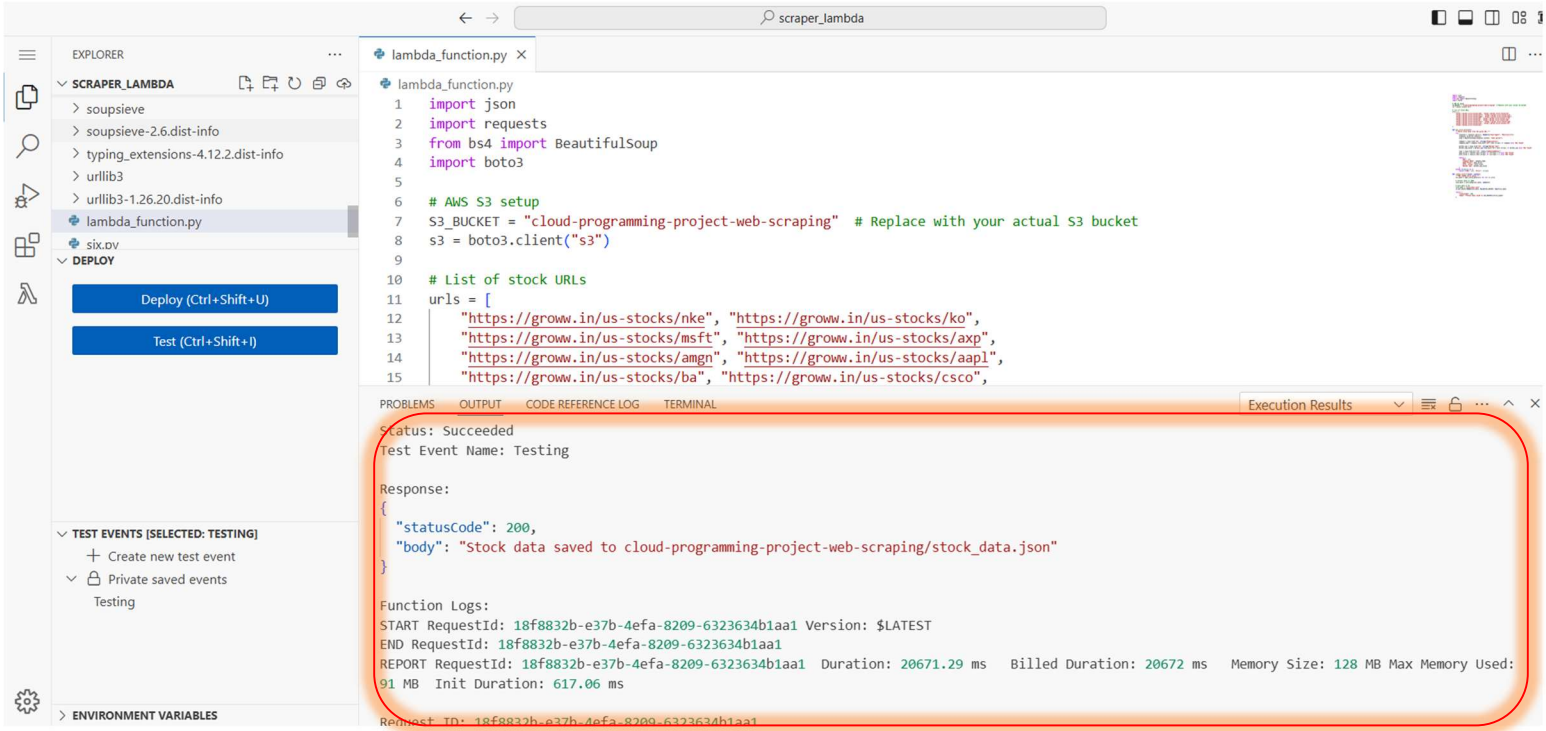
```
import json
import requests
from bs4 import BeautifulSoup
import boto3

# AWS S3 setup
S3_BUCKET = "cloud-programming-project-web-scraping"  # Replace with your
actual S3 bucket
s3 = boto3.client("s3")

# List of stock URLs
urls = [
    "https://groww.in/us-stocks/nke", "https://groww.in/us-stocks/ko",
    "https://groww.in/us-stocks/msft", "https://groww.in/us-stocks/axp",
    "https://groww.in/us-stocks/amgn", "https://groww.in/us-stocks/aapl",
    "https://groww.in/us-stocks/ba", "https://groww.in/us-stocks/csco",
    "https://groww.in/us-stocks/gs", "https://groww.in/us-stocks/ibm",
    "https://groww.in/us-stocks/intc", "https://groww.in/us-stocks/jpm",
    "https://groww.in/us-stocks/mcd", "https://groww.in/us-stocks/crm",
    "https://groww.in/us-stocks/dis"
]

def get_stock_data(url):
    """Fetch stock data from the given URL."""
    try:
        response = requests.get(url, headers={"User-Agent":
"Mozilla/5.0"})
        response.raise_for_status()
        soup = BeautifulSoup(response.content, "html.parser")

        company = soup.find("td", string="Organisation")
        company_name = company.find_next("td").text.strip() if company
else "Not Found"

        market_cap = soup.find("td", string="Market Cap")
        market_cap_value = market_cap.find_next("td").text.strip() if
market_cap else "Not Found"

        rows = soup.find_all("td", class_="bodyLargeHeavy")
        open_price = rows[0].text.strip() if len(rows) > 0 else "Not
Found"
        prev_close = rows[1].text.strip() if len(rows) > 1 else "Not
Found"

        return {
            "URL": url,
            "Company Name": company_name,
            "Open Price": open_price,
            "Prev. Close": prev_close,
            "Market Cap": market_cap_value
        }
    except Exception as e:
```

```
        return {"URL": url, "Error": str(e)}

def lambda_handler(event, context):
    """AWS Lambda entry point."""
    all_data = [get_stock_data(url) for url in urls]

    # Convert data to JSON
    json_data = json.dumps(all_data, indent=4)

    # Save data to S3
    file_name = "stock_data.json"
    s3.put_object(Body=json_data, Bucket=S3_BUCKET, Key=file_name)

    return {
        "statusCode": 200,
        "body": f"Stock data saved to {S3_BUCKET}/{file_name}"
    }
```

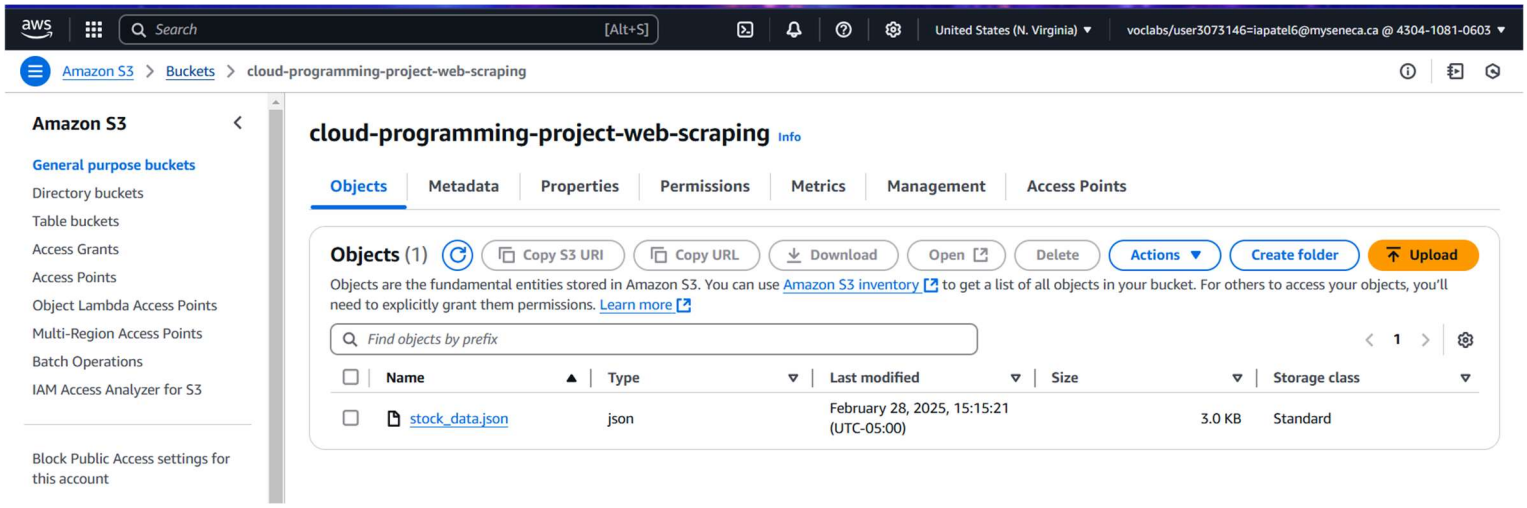
- **Scrapes stock data** from multiple URLs.
- **Extracts important details** like company name, open price, previous close, and market cap.
- **Formats data into JSON** and **uploads it to AWS S3** for storage.
- **Executes automatically** whenever triggered by AWS EventBridge (scheduled execution).



Successfully got the output file stored in S3 we created.



Here is the S3 bucket



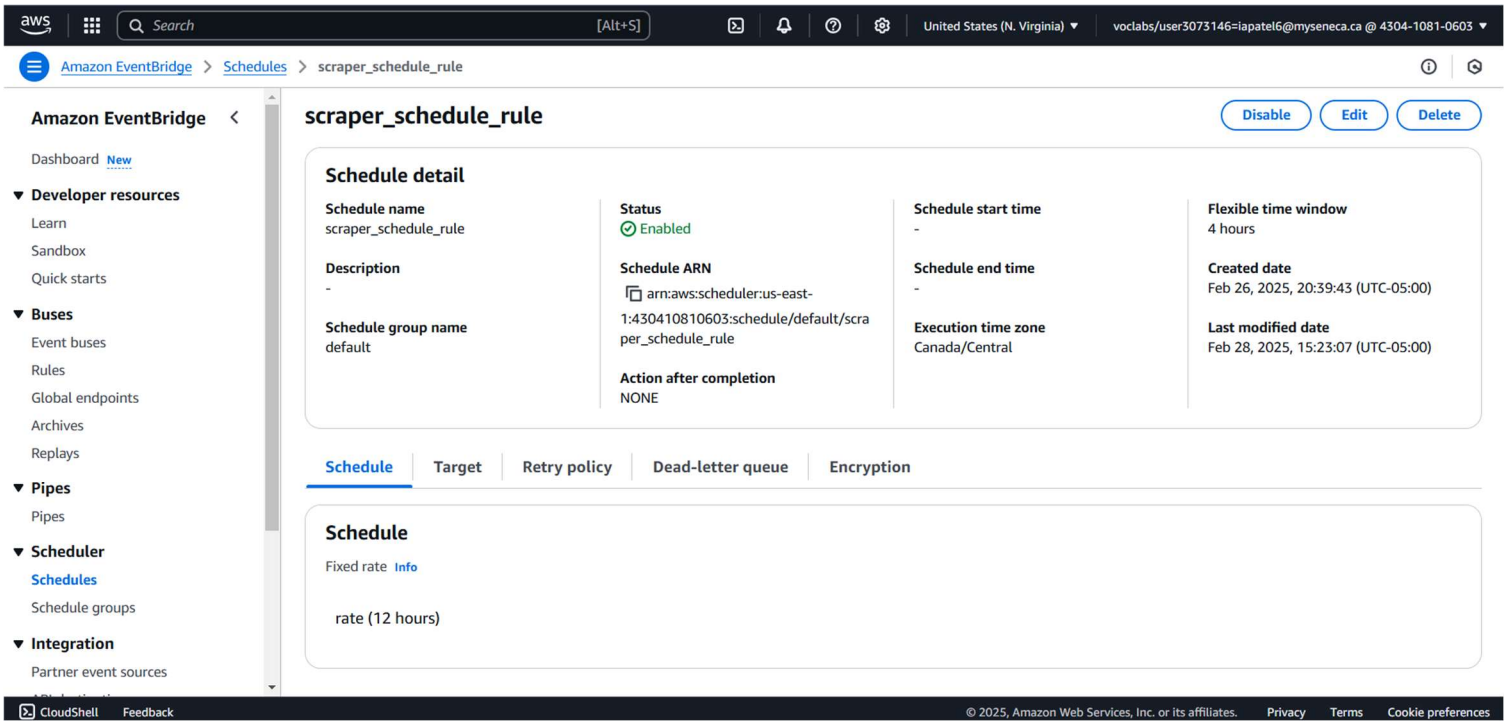
Here is the output file.

Amazon EventBridge Recurring Schedule

To automate our stock scraper Lambda function, we created an **Amazon EventBridge rule** that triggers the function at a fixed interval.

Why Use EventBridge?

- **Automates Lambda execution** without manual intervention.
- **Reduces costs** by running only at scheduled intervals instead of constantly polling data.
- **Ensures reliability** by running at a fixed schedule without external triggers.



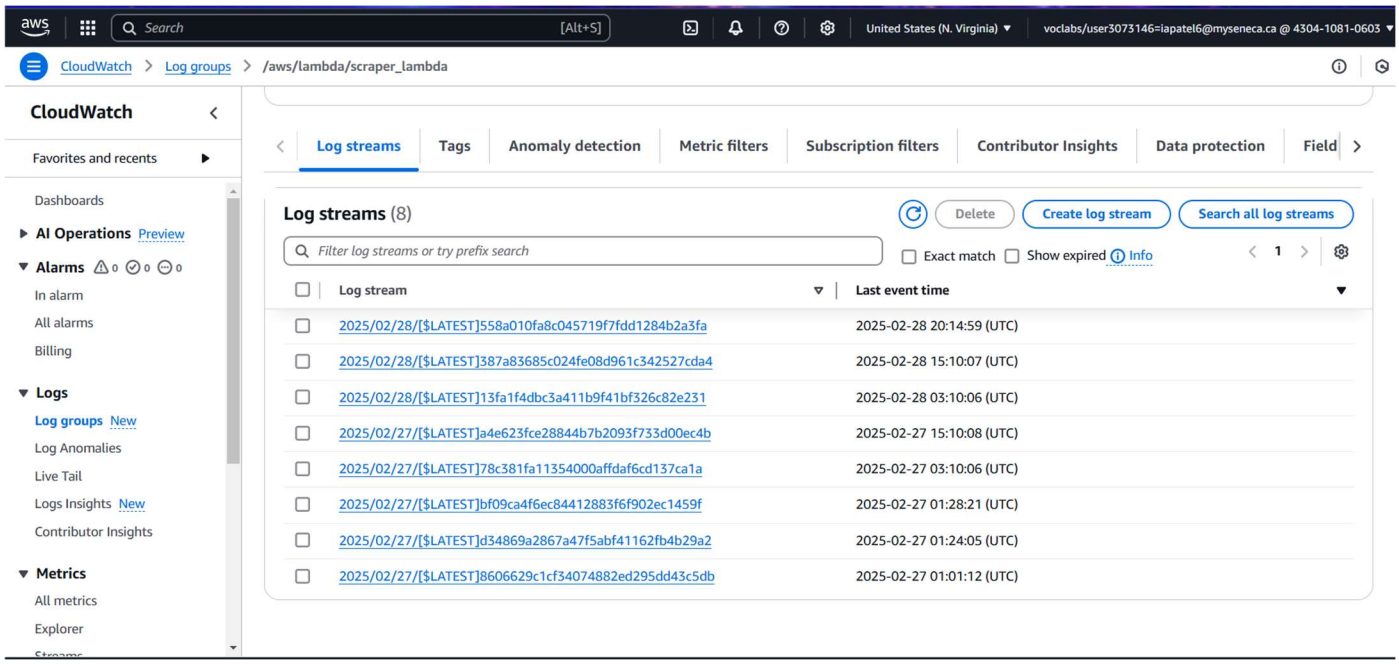
Amazon EventBridge Rule:

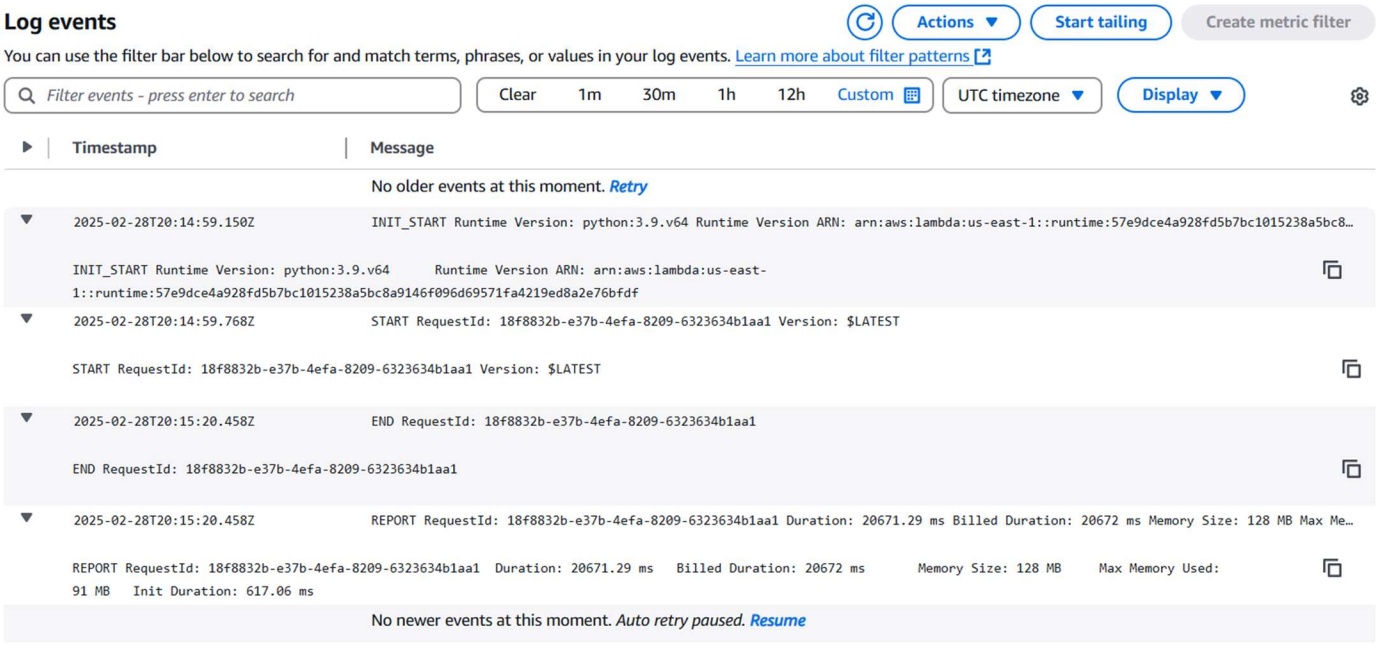
We set up an **Amazon EventBridge rule** to automatically trigger our Lambda function every **12 hours**. This ensures our stock scraper runs on a schedule without manual execution.

Since AWS Lambda doesn't run continuously, EventBridge acts as a scheduler, invoking the function at fixed intervals. This helps in automating data collection while keeping costs low.

Setting Up CloudWatch Logs to Monitor AWS Lambda Execution

Amazon CloudWatch allows you to **monitor and troubleshoot AWS Lambda executions** by capturing logs, setting alarms, and analyzing performance metrics. Integrating CloudWatch with Lambda provides **real-time visibility** into function invocations, execution times, errors, and resource usage.



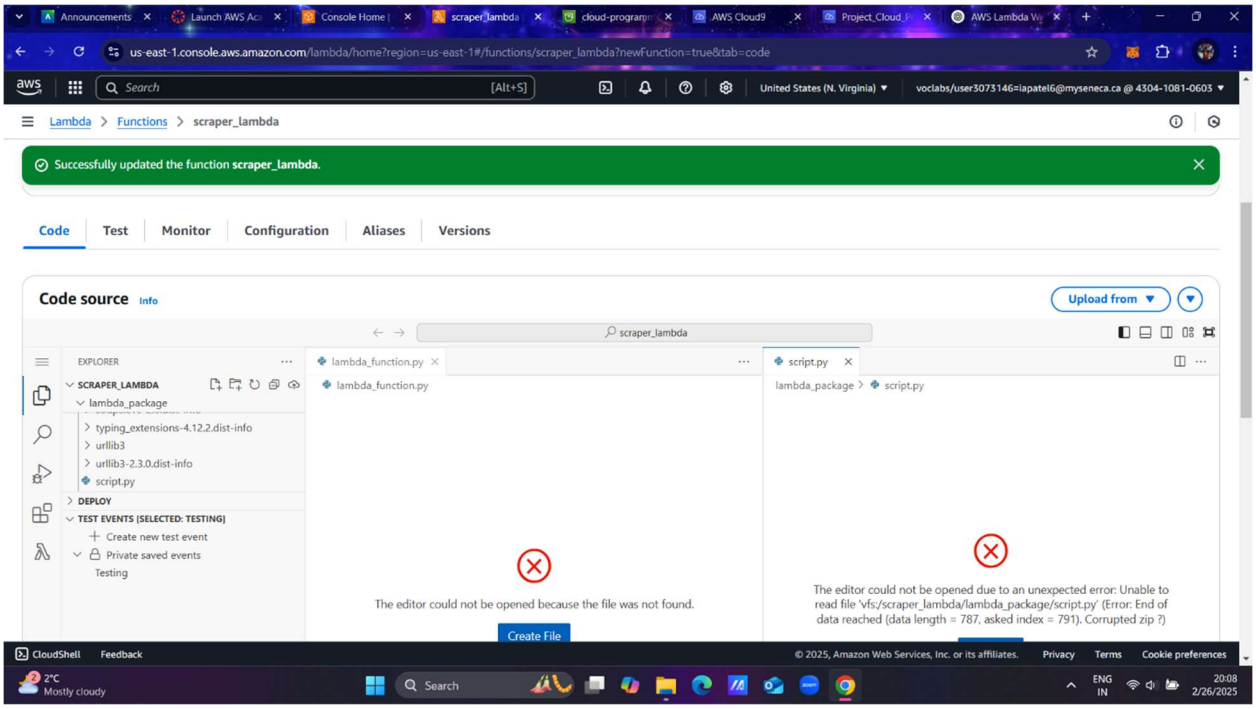


Step 4 : Issues Faced and Resolutions

1. Issue: Lambda Function Failing to Read Uploaded Python Script

Description:
Initially, I uploaded a simple ZIP file containing my Python script to AWS Lambda. However, the script could not be opened, and I encountered an error stating that the file was corrupted or unreadable.

Error Message:
"The editor could not be opened due to an unexpected error: Unable to read file '/vfs/scraper_lambda/lambda_package/script.py' (Error: End of data reached (data length = 787, asked index = 791). Corrupted zip?)."



Cause:
I later realized that simply uploading a ZIP file with the script was not sufficient. AWS Lambda requires that all dependencies be included in the package when uploading a ZIP file.

Solution:
To fix this, I created a new folder, installed the necessary dependencies using pip, and then re-zipped the entire package. Here's the process I followed:

Create a new directory for the Lambda function:

```
mkdir lambda_package
cd lambda_package
```

Install dependencies inside the directory:

```
pip install -r requirements.txt -t .
```

1. Add my script (lambda_function.py) to the folder.
2. Zip everything:

```
zip -r lambda_package.zip .
```

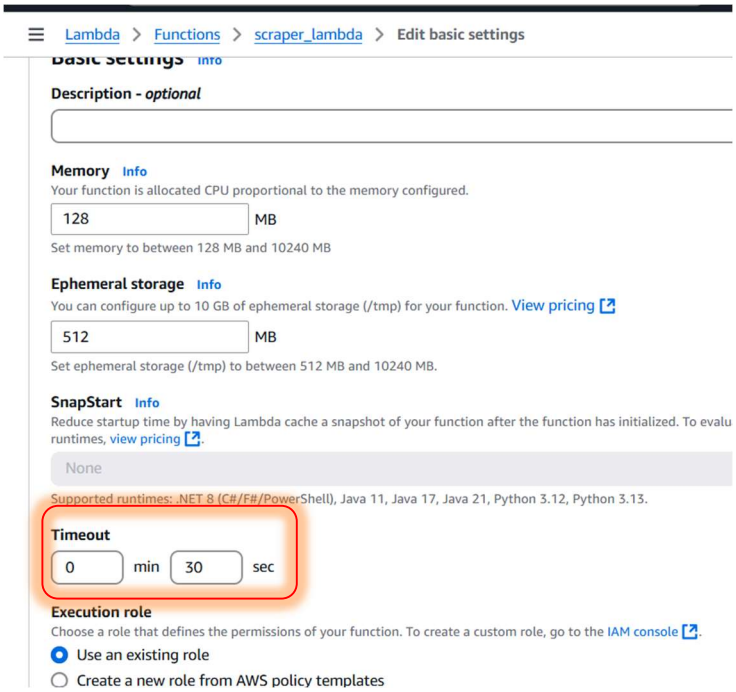
3. Upload the lambda_package.zip file to AWS Lambda.

After this, my function was able to access the script without errors.

2. Issue: Lambda Execution Timeout

Description:
By default, AWS Lambda sets a 3-second timeout for function execution. Since my script was performing web scraping, it sometimes took longer than 3 seconds to complete execution, causing it to fail.

Error Message:
"Task timed out after 3.00 seconds."



Cause:
The default execution time was too short for my function to complete its task.

Solution:
I increased the timeout limit to **30 seconds** in the Lambda settings:

1. Navigate to **AWS Lambda Console**.
2. Select the function (scraper_lambda).
3. Click on **Configuration** → **General Configuration**.
4. Change the **Timeout** value from **3 seconds to 30 seconds**.