

## 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

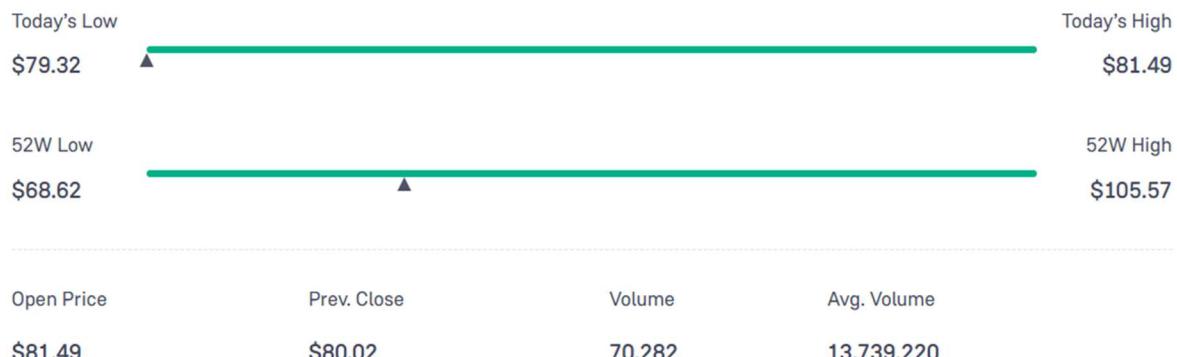
From each stock page, my script extracts:

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.

## Performance



## Fundamentals

Market Cap	\$118.35B	Div. Yield	2.00%
P/B Ratio	8.4	Book Value	9.14
P/E Ratio	24.2	EPS(TTM)	0.95
Enterprise Value	\$122.44B	ROE	NA

### 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:

- Converts the collected data into JSON format.
- 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.

```
1 import requests
2 from bs4 import BeautifulSoup
3
4 # List of stock URLs
5 urls = [
6     'https://groww.in/us-stocks/nke', 'https://groww.in/us-stocks/ko', 'https://groww.in/us-stocks/msft',
7     'https://groww.in/us-stocks/axp', 'https://groww.in/us-stocks/amgn',
8     'https://groww.in/us-stocks/aapl', 'https://groww.in/us-stocks/ba', 'https://groww.in/us-stocks/cSCO',
9     'https://groww.in/us-stocks/gs', 'https://groww.in/us-stocks/ibm', 'https://groww.in/us-stocks/intc',
10    'https://groww.in/us-stocks/jpm', 'https://groww.in/us-stocks/mcd', 'https://groww.in/us-stocks/crm',
11    'https://groww.in/us-stocks/dis'
12 ]
13
14 def get_stock_data(url):
15     try:
16         # Fetch webpage content
17         response = requests.get(url, headers={"User-Agent": "Mozilla/5.0"})
18         response.raise_for_status()
19         soup = BeautifulSoup(response.content, "html.parser")
20
21         # Get company name
22         company = soup.find("td", string="Organisation")
23         company_name = company.find_next("td").text.strip() if company else "Not Found"
24
25         # Get market cap
26         market_cap = soup.find("td", string="Market Cap")
27         market_cap_value = market_cap.find_next("td").text.strip() if market_cap else "Not Found"
28
29         # Get open price and previous close
30         rows = soup.findall("td", class_="bodyLargeHeavy")
31         open_price = rows[0].text.strip() if len(rows) > 0 else "Not Found"
32         prev_close = rows[1].text.strip() if len(rows) > 1 else "Not Found"
33
34         return {
35             "URL": url,
36             "Company Name": company_name,
37             "Open Price": open_price,
38             "Prev. Close": prev_close,
39             "Market Cap": market_cap_value
40         }
41     except Exception as e:
42         return {"URL": url, "Error": str(e)}
43
44 # Get data for all stocks
45 all_data = []
```

```
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).

The screenshot shows a code editor interface with the following details:

- Left Sidebar:** Explorer view showing project structure under "SCRAPER\_LAMBDA".
  - folders: soupsieve, soupsieve-2.6.dist-info, typing\_extensions-4.12.2.dist-info, urllib3, urllib3-1.26.20.dist-info.
  - files: lambda\_function.py, lambda\_function.py (selected), SIX.DV.
- Middle Area:** Editor window titled "scraper\_lambda" showing the content of "lambda\_function.py". The code is a Python script for web scraping stock data from various URLs.

```
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")

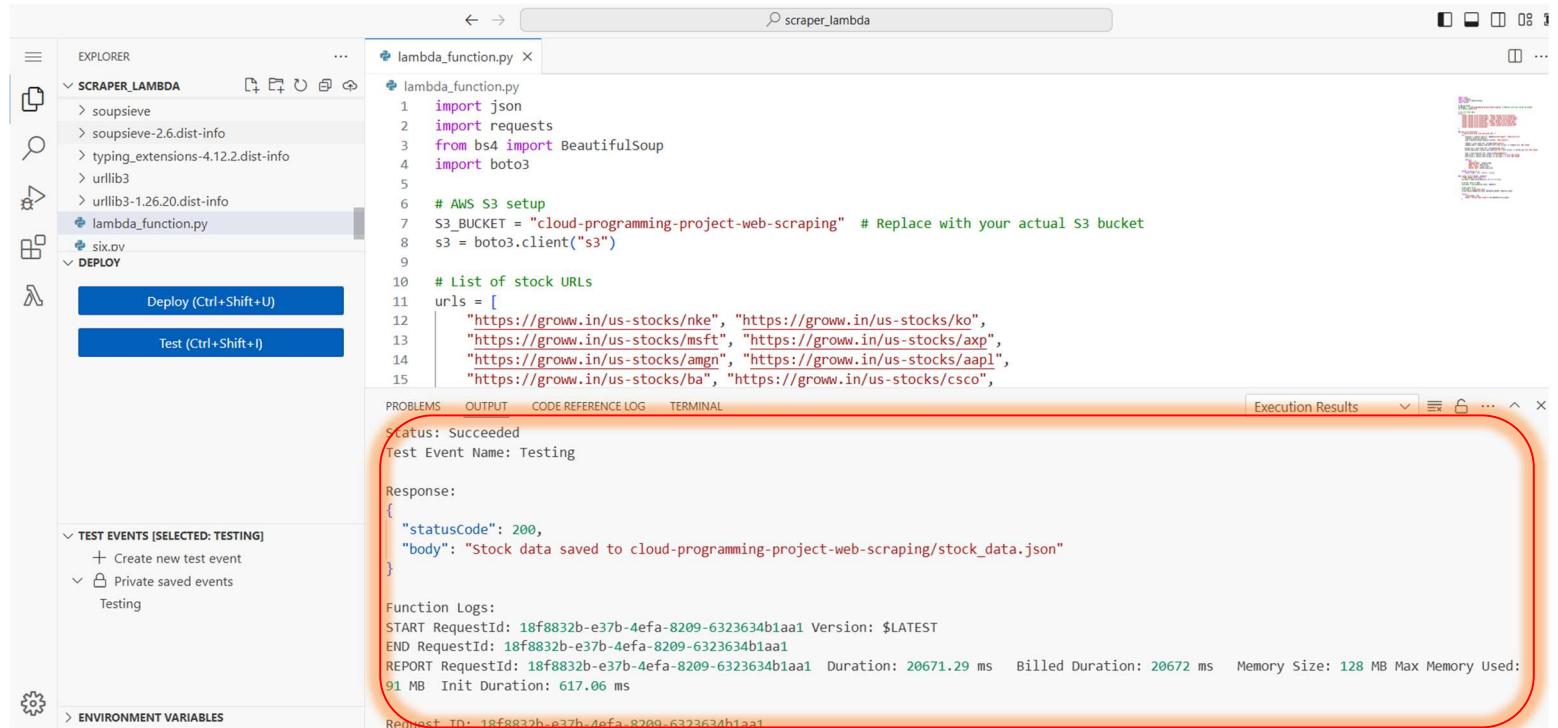
        company = soup.find("td", string="Organisation")
        company_name = company.find_next("td").text.strip() if company else "Not Found"

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

        return {
            "company": company_name,
            "market_cap": market_cap_value
        }
    except Exception as e:
        print(f"Error fetching data from {url}: {e}")

if __name__ == "__main__":
    for url in urls:
        data = get_stock_data(url)
        print(data)
```
- Bottom Status Bar:** Shows an Amazon Q tip: "Amazon Q Tip 1/3: Start typing to get suggestions ([ctrl+shift+s])."

Successfully got the output file stored in S3 we created.



The screenshot shows the AWS Lambda function configuration in the AWS Toolkit for VS Code. The code editor displays `lambda_function.py` with the following content:

```
lambda_function.py
1 import json
2 import requests
3 from bs4 import BeautifulSoup
4 import boto3
5
6 # AWS S3 setup
7 S3_BUCKET = "cloud-programming-project-web-scraping" # Replace with your actual s3 bucket
8 s3 = boto3.client("s3")
9
10 # List of stock URLs
11 urls = [
12     "https://groww.in/us-stocks/nke", "https://groww.in/us-stocks/ko",
13     "https://groww.in/us-stocks/msft", "https://groww.in/us-stocks/axp",
14     "https://groww.in/us-stocks/amgn", "https://groww.in/us-stocks/aapl",
15     "https://groww.in/us-stocks/ba", "https://groww.in/us-stocks/csco",
```

The Lambda function has been tested, and the results are shown in the "Execution Results" tab:

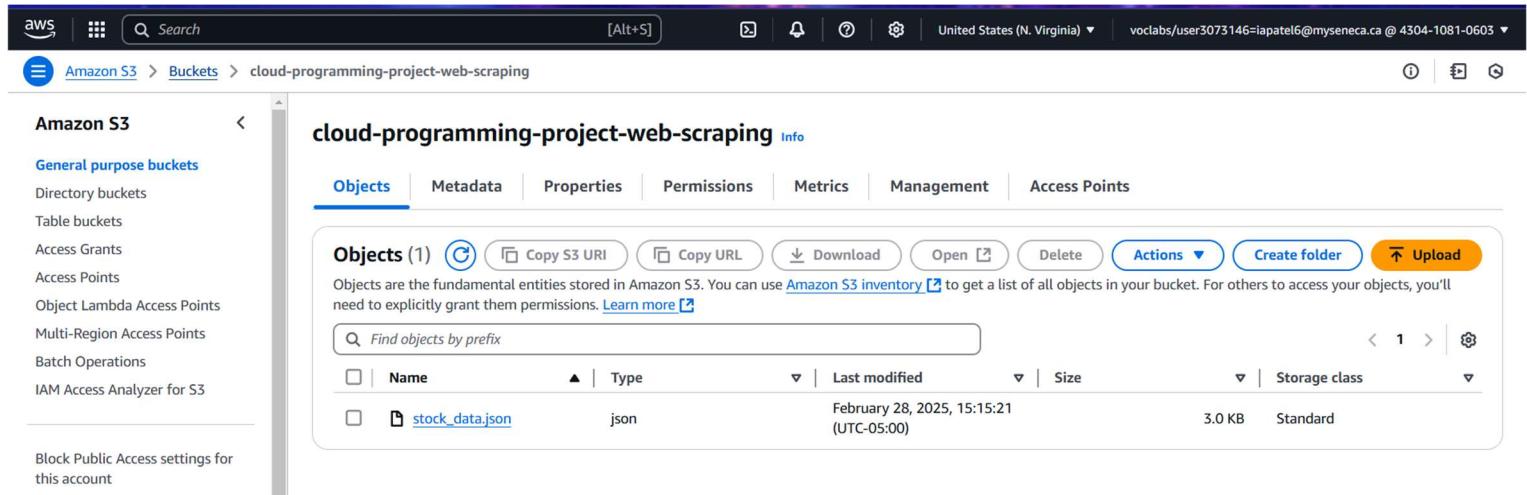
```
Status: Succeeded
Test Event Name: Testing

Response:
{
    "statusCode": 200,
    "body": "Stock data saved to cloud-programming-project-web-scraping/stock_data.json"
}

Function Logs:
START RequestId: 18f8832b-e37b-4efa-8209-6323634b1aa1 Version: $LATEST
END RequestId: 18f8832b-e37b-4efa-8209-6323634b1aa1
REPORT RequestId: 18f8832b-e37b-4efa-8209-6323634b1aa1 Duration: 20671.29 ms Billed Duration: 20672 ms Memory Size: 128 MB Max Memory Used: 91 MB Init Duration: 617.06 ms

Request ID: 18f8832b-e37b-4efa-8209-6323634b1aa1
```

Here is the S3 bucket



The screenshot shows the AWS S3 console. The left sidebar lists "General purpose buckets" and "Amazon S3". The main area shows the "cloud-programming-project-web-scraping" bucket. The "Objects" tab is selected, displaying one object:

Name	Type	Last modified	Size	Storage class
<a href="#">stock_data.json</a>	json	February 28, 2025, 15:15:21 (UTC-05:00)	3.0 KB	Standard

Here is the output file.



The screenshot shows the Notepad++ application window with the file "stock\_data.json" open. The file contains a JSON array of stock information for seven companies. Each company object includes its URL, name, open price, previous close, and market cap. The code is color-coded for readability, with URLs in blue and other text in black. Red vertical and horizontal lines highlight the JSON structure, showing the nesting of objects and arrays. The Notepad++ interface includes a menu bar with File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, and Window, and a toolbar with various icons.

```
1  [
2    {
3      "URL": "https://groww.in/us-stocks/nke",
4      "Company Name": "Nike Inc",
5      "Open Price": "$82.10",
6      "Prev. Close": "$81.72",
7      "Market Cap": "$120.87B"
8   },
9   {
10    "URL": "https://groww.in/us-stocks/ko",
11    "Company Name": "Coca-Cola Company The",
12    "Open Price": "$71.27",
13    "Prev. Close": "$71.49",
14    "Market Cap": "$307.47B"
15  },
16  {
17    "URL": "https://groww.in/us-stocks/msft",
18    "Company Name": "Microsoft Corporation",
19    "Open Price": "$398.01",
20    "Prev. Close": "$397.90",
21    "Market Cap": "$2.95T"
22  },
23  {
24    "URL": "https://groww.in/us-stocks/axp",
25    "Company Name": "American Express Co",
26    "Open Price": "$295.14",
27    "Prev. Close": "$293.34",
28    "Market Cap": "$206.08B"
29  },
30  {
31    "URL": "https://groww.in/us-stocks/amgn",
32    "Company Name": "Amgen Inc",
33    "Open Price": "$311.95",
34    "Prev. Close": "$315.63",
35    "Market Cap": "$169.55B"
36  },
37  {
38    "URL": "https://groww.in/us-stocks/aapl",
39    "Company Name": "Apple Inc",
40    "Open Price": "$244.33",
41    "Prev. Close": "$247.04",
42    "Market Cap": "$3.71T"
43  }
],
```

## 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.

The screenshot shows the AWS EventBridge console with a recurring schedule rule named "scraper\_schedule\_rule". The "Schedule detail" section displays the following information:

Setting	Value
Schedule name	scraper_schedule_rule
Status	Enabled
Description	-
Schedule group name	default
Action after completion	NONE
Schedule start time	-
Schedule end time	-
Execution time zone	Canada/Central
Flexible time window	4 hours
Created date	Feb 26, 2025, 20:39:43 (UTC-05:00)
Last modified date	Feb 28, 2025, 15:23:07 (UTC-05:00)

The "Schedule" tab is selected, showing a fixed rate of "rate (12 hours)". Other tabs include Target, Retry policy, Dead-letter queue, and Encryption.

### 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.

The screenshot shows the AWS CloudWatch Log streams interface. On the left, there's a navigation sidebar with sections like AI Operations, Alarms, Logs, and Metrics. The main area is titled "Log streams (8)" and lists eight log streams with their last event times:

Log stream	Last event time
2025/02/28/[\$LATEST]558a010fa8c045719f7fd1284b2a3fa	2025-02-28 20:14:59 (UTC)
2025/02/28/[\$LATEST]387a83685c024fe08d961c342527cd4	2025-02-28 15:10:07 (UTC)
2025/02/28/[\$LATEST]13fa1f4dbc3a411b9f41bf32682e231	2025-02-28 03:10:06 (UTC)
2025/02/27/[\$LATEST]a4e623fce28844b7b2093f733d00ec4b	2025-02-27 15:10:08 (UTC)
2025/02/27/[\$LATEST]78c381fa11354000affdaf6cd137ca1a	2025-02-27 03:10:06 (UTC)
2025/02/27/[\$LATEST]bf09ca4f6ec84412883f6f902ec1459f	2025-02-27 01:28:21 (UTC)
2025/02/27/[\$LATEST]d34869a2867a47f5abf41162fb4b29a2	2025-02-27 01:24:05 (UTC)
2025/02/27/[\$LATEST]8606629c1cf34074882ed295dd43c5db	2025-02-27 01:01:12 (UTC)

## Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

The screenshot shows the AWS CloudWatch Log events interface. At the top, there's a filter bar with a search input, time range options (1m, 30m, 1h, 12h, Custom), UTC timezone, and display settings. The main area displays log events for a Lambda function execution:

Timestamp	Message
2025-02-28T20:14:59.150Z	INIT_START Runtime Version: python:3.9.v64 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:57e9dce4a928fd5b7bc1015238a5bc8..
2025-02-28T20:14:59.768Z	START RequestId: 18f8832b-e37b-4efa-8209-6323634b1aa1 Version: \$LATEST
2025-02-28T20:15:20.458Z	END RequestId: 18f8832b-e37b-4efa-8209-6323634b1aa1
2025-02-28T20:15:20.458Z	REPORT RequestId: 18f8832b-e37b-4efa-8209-6323634b1aa1 Duration: 20671.29 ms Billed Duration: 20672 ms Memory Size: 128 MB Max Me..

At the bottom, it says "No newer events at this moment. Auto retry paused. [Resume](#)"

## Step 4 : Issues Faced and Resolutions

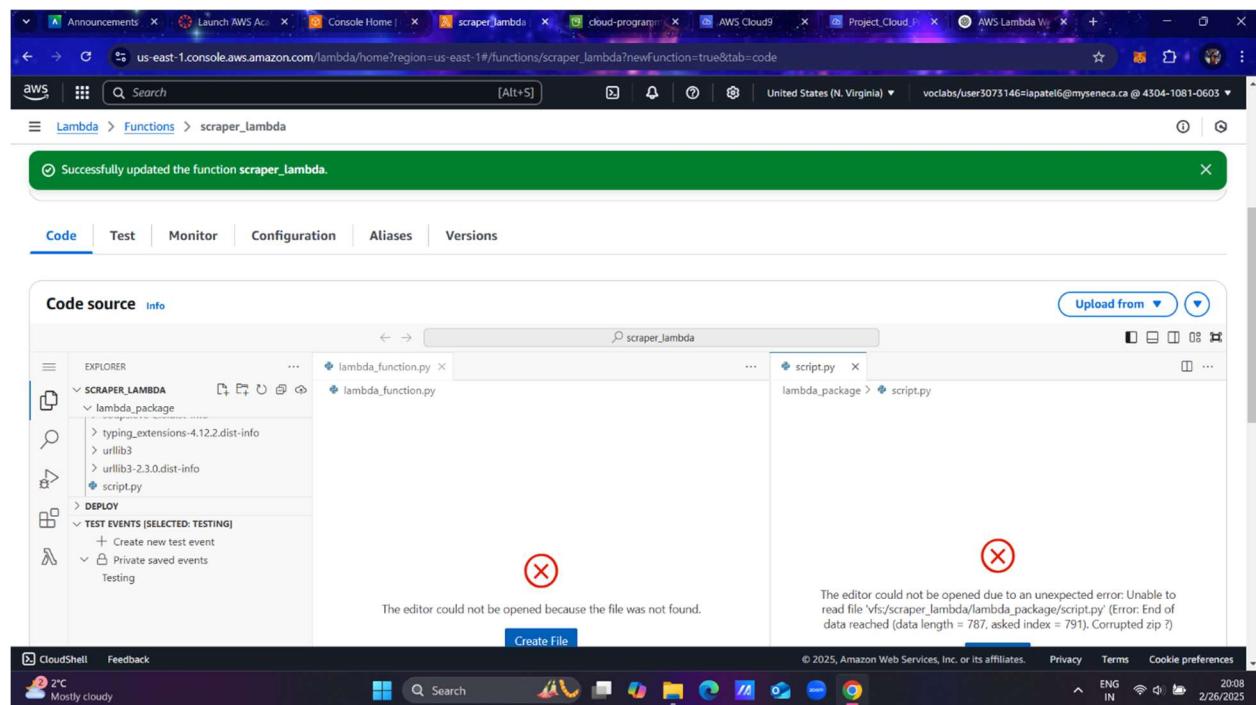
### 1. Issue: Lambda Function Failing to Read Uploaded Python Script

#### 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/scrapers\_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.

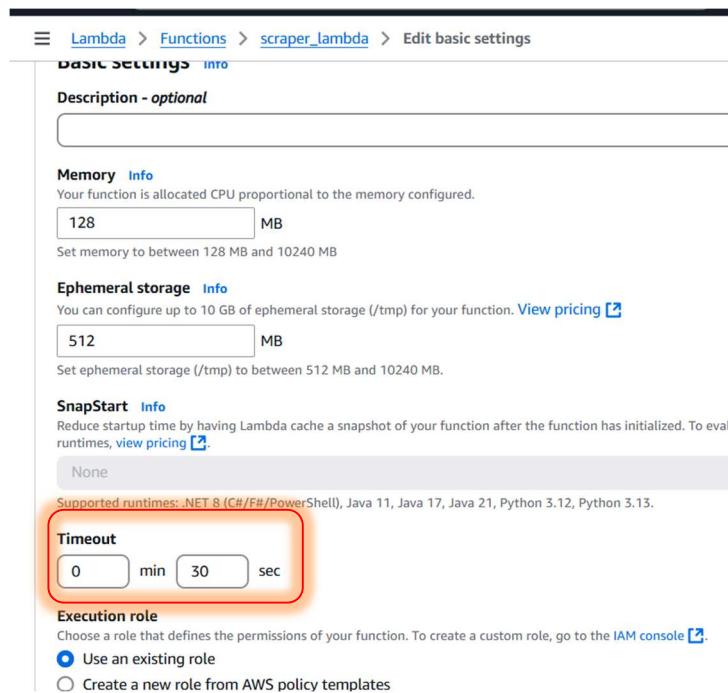
## 2. Issue: Lambda Execution Timeout

### 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**.