**CHAPTER 1**

**INTRODUCTION**

The Real-Time Stock Price Alert System automatically monitors stock prices and notifies users when a specific stock reaches a predefined price threshold. This automation saves users the time of manually tracking prices. Upon reaching the target price, the system sends real-time alerts via SMS, email, or mobile notifications, enabling investors and traders to respond quickly to market fluctuations and seize opportunities.

**CLOUD COMPUTING:**

The system is hosted in a scalable, highly available cloud environment without relying on physical servers. Utilizing AWS's global infrastructure provides flexibility, scalability, and reliability. It automatically adjusts resources based on demand, managing usage fluctuations without downtime, simplifying deployment and maintenance, and reducing operational complexity.

**SERVERLESS ARCHITECTURE:**

Serverless architecture, primarily using AWS Lambda, is vital to the system. Functions execute only when triggered, minimizing the need for continuous server management and reducing costs by utilizing resources only as needed. AWS Lambda delivers high performance and low latency, providing immediate actions in response to real-time events.

The serverless model ensures that:

* The system can automatically scale with demand.
* There is no need to manage or update the underlying infrastructure.
* The system only incurs costs for actual usage, optimizing budget efficiency.
* Performance remains high, as resources are instantly allocated to handle alerts and notifications, ensuring users receive timely updates.

**REAL-TIME DATA PROCESSING:**

In the fast-moving stock market, real-time data processing is crucial. AWS serverless services manage stock data and trigger notifications based on preset conditions. Amazon EventBridge, integrated with CloudWatch, monitors data flow, ensuring instant reaction to stock price changes and timely alerts.

**MESSAGING AND NOTIFICATION SYSTEMS:**

Amazon Simple Notification Service (SNS) is used to send alerts through various channels, such as SMS and email. This flexibility enhances user experience by providing timely updates, while SNS guarantees high availability and reliable message delivery, keeping users informed and able to respond swiftly to market changes.

**SCALABILITY AND FLEXIBILITY:**

Scalability is a significant advantage of cloud infrastructure, allowing the system to adjust resources automatically as stock data volumes grow without manual intervention. AWS Cloud9 serves as the development platform, enabling collaborative real-time coding and improving the development process, allowing for quick adaptation to changing requirements.

**CHAPTER 2**

**EXECUTIVE SUMMARY**

The **Real-Time Stock Price Alert System** is an automated AWS-based solution that continuously monitors stock prices and sends alerts when predefined thresholds are reached. This eliminates manual tracking, helping investors make timely decisions in a fast-moving market. By using AWS Lambda, Amazon SNS, and CloudWatch, along with a third-party API like Alpha Vantage, the system provides real-time notifications via email or SMS. The cloud integration ensures high availability and reliability, simplifying stock monitoring for users.

**Key Components:**

* **Amazon SNS (Simple Notification Service):**

Amazon SNS (Simple Notification Service) is a managed messaging service that decouples microservices and serverless applications. In your Real-Time Stock Price Alert System, SNS serves as the communication hub, sending alerts based on stock price thresholds through channels like SMS, email, and custom endpoints, ensuring real-time updates without delay.

* **AWS Lambda:**

AWS Lambda is a serverless compute service that allows users to run code without managing servers. In your Real-Time Stock Price Alert System, Lambda analyzes stock prices and compares them against preset conditions, triggering notifications as needed. This ensures resources are used only when checking stock prices, leading to cost efficiency.

* **Amazon CloudWatch (with EventBridge):**

Amazon CloudWatch is a monitoring service that provides real-time insights into system performance. In your Real-Time Stock Price Alert System, it continuously tracks stock prices, ensuring effective monitoring. Paired with EventBridge, it triggers Lambda functions when predefined stock conditions are met, enabling event-driven automation.

* **IAM (Identity and Access Management):**

IAM (Identity and Access Management) is a service that controls access between different AWS services. In your Real-Time Stock Price Alert System, IAM ensures that only authorized services and users can access sensitive data or resources, such as those associated with Lambda, SNS, and CloudWatch.

* **AWS Cloud9:**

AWS Cloud9 is an integrated development environment (IDE) that enables real-time coding and testing. In your Real-Time Stock Price Alert System, Cloud9 is used as the development environment, facilitating easy deployment of updates or patches to the system.

**Use Cases:**

* Investment Monitoring: Sends alerts when stock prices hit user-defined thresholds, aiding in timely decision-making.
* Risk Management: Helps users manage portfolio risks by notifying them of price drops.
* Day Trading: Provides real-time stock alerts for day traders to act on short-term fluctuations.
* Automated Trading: Can trigger buy or sell orders based on stock price changes.
* Watchlist Notifications: Keeps users informed about stocks on their watchlist.
* Educational Tool: Useful for teaching real-world stock tracking and alert systems.

**CHAPTER 3**

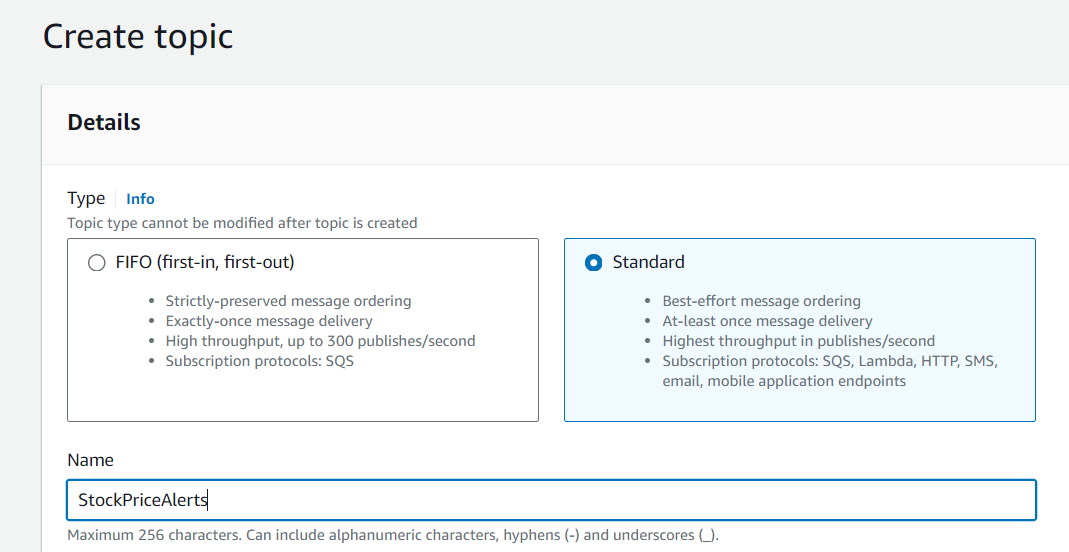
**STOCK PRICE MONITORING AND ALERT SYSTEM**

Implementing a Stock Price Alert System that leverages the reliability and flexibility of AWS services, including Lambda for real-time price analysis, SNS for immediate notifications, and CloudWatch for system monitoring. By integrating these services, the system can effectively track stock prices, trigger alerts based on user-defined thresholds, and maintain optimal performance while ensuring robust security measures are in place. This comprehensive approach enables users to receive timely updates, empowering them to make informed investment decisions swiftly.

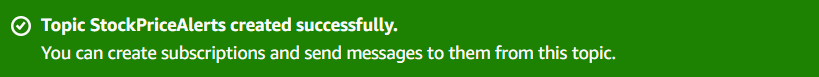
**Steps Involved:**

1.Access the Amazon SNS Console and Create a Topic

* Log in to your AWS Management Console and search for SNS to select Simple Notification Service.
* Click on Topics in the left navigation pane, then select Create Topic.
* Choose Standard as the Type, and enter StockPriceAlerts as the Name.
* Click “Create topic”.



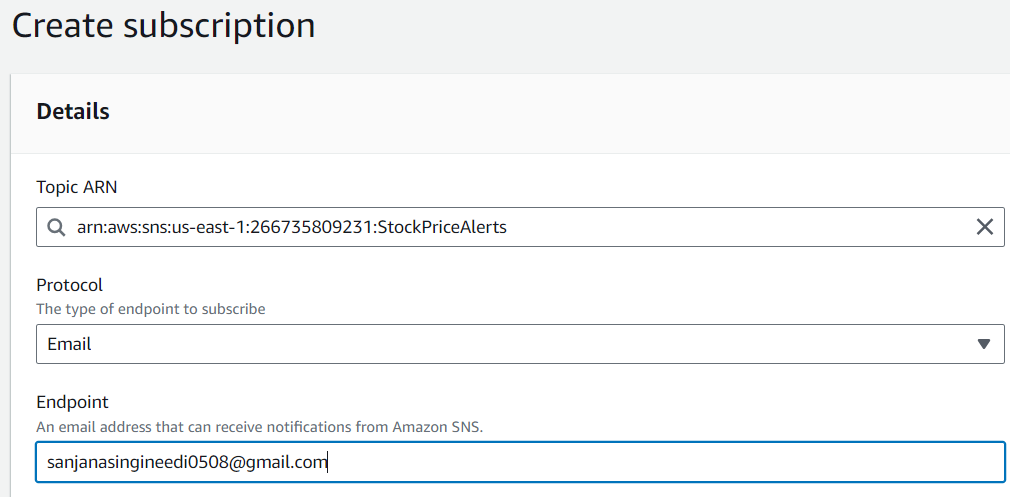
**Figure 3.1 Create a Topic**



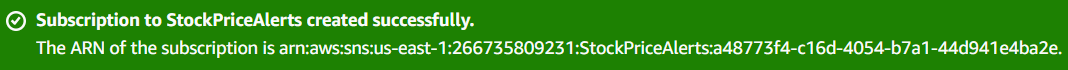
**Figure 3.2 Topic Successfully Created**

2.Set Up a Subscription

* Navigate to the Subscriptions section in the left navigation pane and click on Create Subscription.
* Select the StockPriceAlerts topic from the Topic ARN dropdown.
* Choose Email or SMS as the Protocol:
  + For Email: Enter your email address (e.g., youremail@example.com).
  + For SMS: Enter your phone number in the appropriate format (e.g., +11234567890).
* Click Create subscription.



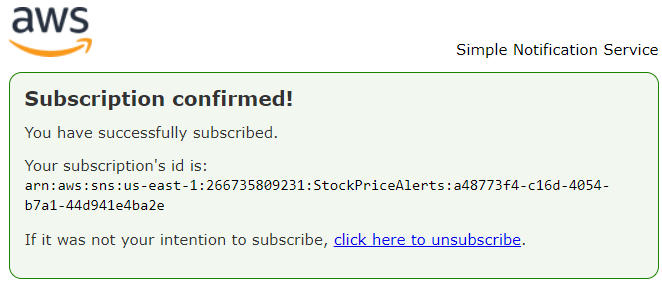
**Figure 3.3 Create Subscription**



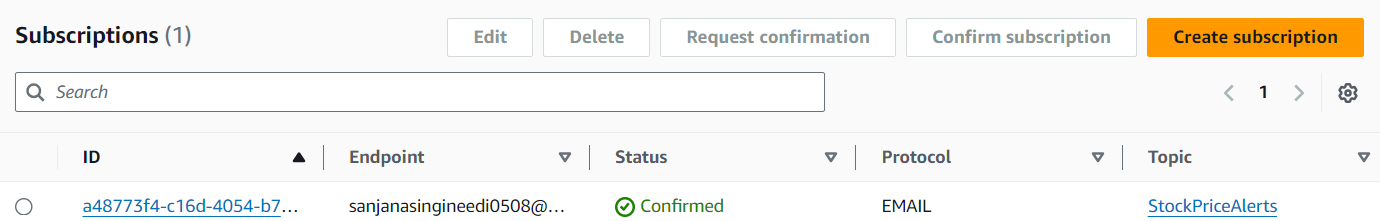
**Figure 3.4 Subscription Successfully Created**

3. Confirm the Subscription (for Email Alerts):

* If you chose Email, check your inbox for a confirmation email from Amazon SNS.
* Open the email and click on the confirmation link to activate your subscription.



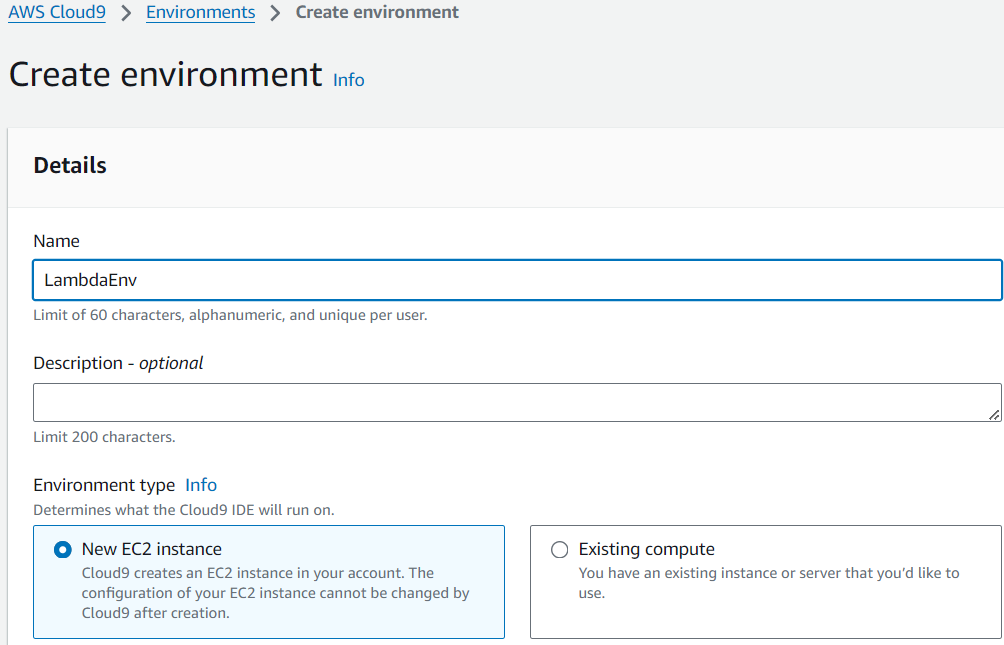
**Figure 3.5 Subscribe through Endpoint**



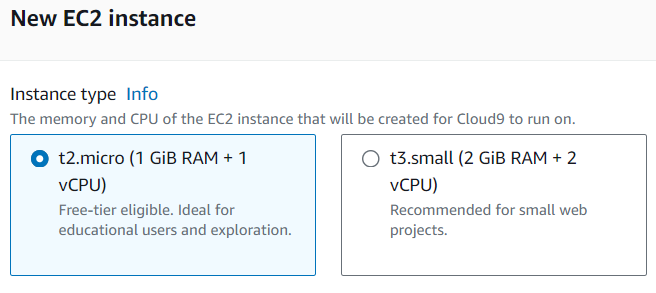
**Figure 3.6 Subscription Confirmed**

4. Set Up AWS Cloud9 Environment

* Search for Cloud9.
* Click on Create environment.
* Provide a name (e.g., LambdaEnv) and configure the instance as needed (e.g., select t2.micro as a default instance size).
* Click Next, then click Create environment.



**Figure 3.7 Create Environment (Cloud9)**



**Figure 3.8 Select the Instance Type (EC2 Instance)**

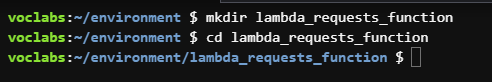
5. Create a Directory for Your Lambda Function

* Once your Cloud9 environment is set up, open a new terminal inside Cloud9.
* Create a directory for your Lambda function by running

**Command:**

mkdir lambda\_requests\_function

cd lambda\_requests\_function



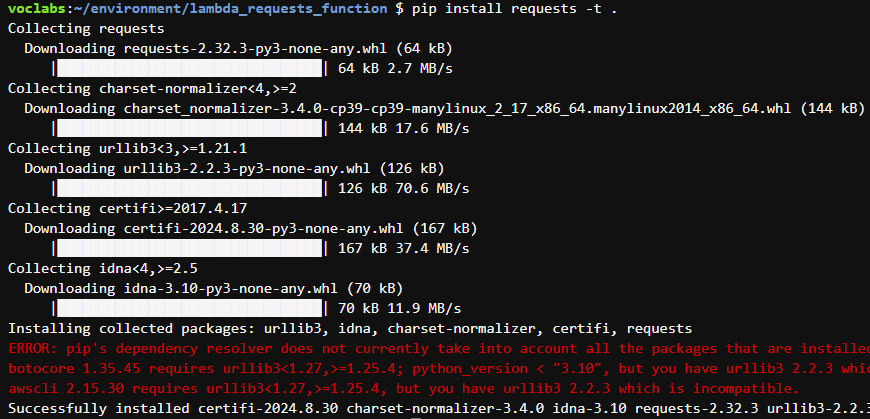
**Figure 3.9 Create a Directory**

6. Install the Requests Library

* Install the requests library in the current directory by executing

**Command:**

pip install requests -t .

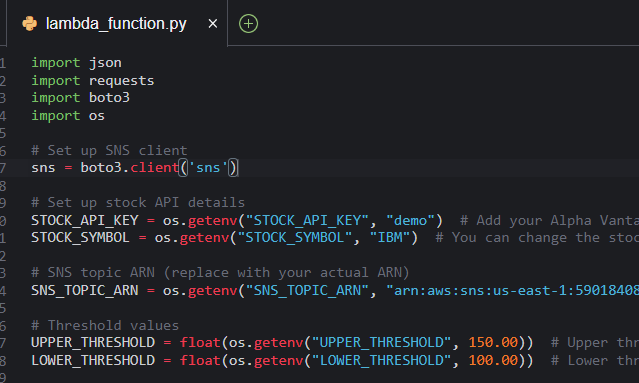


**Figure 3.10 Install requests Package**

* This command installs the requests module and its dependencies into the lambda\_requests\_function directory.

7. Add Your Lambda Function Code

* In the Cloud9 file explorer, create a new Python file named lambda\_function.py in the lambda\_requests\_function directory.
* Add your Lambda function code to this file.



**Figure 3.11 Create lambda\_function.py file**

**Program:**

import json

import requests

import boto3

import os

# Set up SNS client

sns = boto3.client('sns')

# Set up stock API details

STOCK\_API\_KEY = os.getenv("STOCK\_API\_KEY", "demo")

STOCK\_SYMBOL = os.getenv("STOCK\_SYMBOL", "IBM")

# SNS topic ARN (replace with your actual ARN)

SNS\_TOPIC\_ARN = os.getenv("SNS\_TOPIC\_ARN", "arn:aws:sns:us-

east-1:590184086939:StockPriceAlerts")

# Threshold values

UPPER\_THRESHOLD = float(os.getenv("UPPER\_THRESHOLD", 150.00))

LOWER\_THRESHOLD = float(os.getenv("LOWER\_THRESHOLD", 100.00))

def get\_stock\_price(symbol):

# Alpha Vantage API URL

url =

f"https://www.alphavantage.co/query?function=TIME\_SERIES\_INTRADAY&symbol={STOCK\_API\_KEY}&interval=1min&apikey={STOCK\_TOPIC\_ARN}"

response = requests.get(url)

data = response.json()

# Extract the latest stock price

time\_series = data.get("Time Series (1min)", {})

if not time\_series:

return None

latest\_time = sorted(time\_series.keys())[0]

latest\_price = float(time\_series[latest\_time]["1. open"])

return latest\_price

def lambda\_handler(event, context):

# Get the current stock price

stock\_price = get\_stock\_price(STOCK\_SYMBOL)

if stock\_price is None:

print("Could not fetch stock price")

return {"statusCode": 500, "body": json.dumps("Stock

price fetch failed")}

print(f"Current {STOCK\_SYMBOL} price: {stock\_price}")

# Check if the stock price exceeds the upper threshold

if stock\_price > UPPER\_THRESHOLD:

message = f"The stock price of {STOCK\_SYMBOL} is

${stock\_price}, which exceeds your upper

threshold of ${UPPER\_THRESHOLD}!"

# Send an SNS notification for upper threshold

sns.publish(

TopicArn=SNS\_TOPIC\_ARN,

Message=message,

Subject=f"{STOCK\_SYMBOL} Stock Price Alert: Above

Threshold"

)

print("Upper threshold alert sent!")

# Check if the stock price falls below the lower threshold

elif stock\_price < LOWER\_THRESHOLD:

message = f"The stock price of {STOCK\_SYMBOL} is

${stock\_price}, which is below your lower

threshold of ${LOWER\_THRESHOLD}!"

# Send an SNS notification for lower threshold

sns.publish(

TopicArn=SNS\_TOPIC\_ARN,

Message=message,

Subject=f"{STOCK\_SYMBOL} Stock Price Alert: Below

Threshold"

)

print("Lower threshold alert sent!")

else:

print(f"The stock price of {STOCK\_SYMBOL} is within the

threshold range.")

return {"statusCode": 200, "body": json.dumps(f"Checked

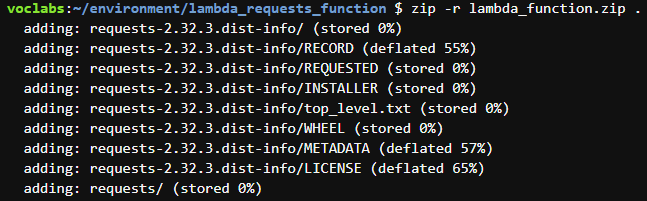
stock price: ${stock\_price}")}

8. Zip the Directory

* After adding your code, zip the contents of the directory by running

**Command:**

zip -r lambda\_function.zip .

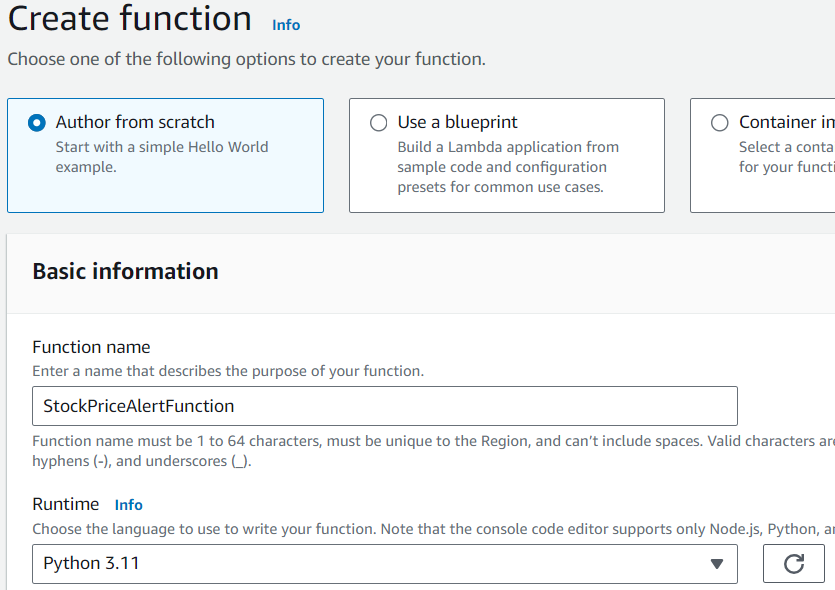


**Figure 3.12 Zip the directory files**

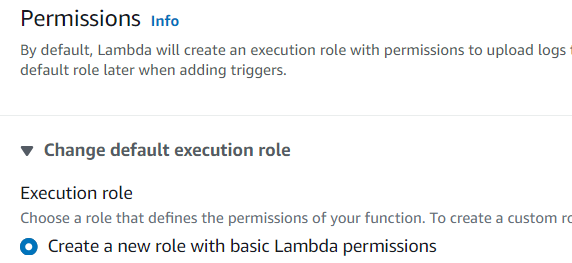
* This creates a file named lambda\_function.zip, containing both the requests library and your lambda\_function.py code.
* download the project file into the local system, extract the downloaded file since it is in the format of zip.

9. Access the AWS Lambda Console and Create a Function

* In to your AWS Management Console and search for Lambda to select the AWS Lambda Console.
* Click on Create Function, then choose Author from scratch.
* Fill in the following details:
  + Function name: StockPriceAlertFunction
  + Runtime: Python 3.x
  + Execution role: Select Create a new role with basic Lambda permissions.
* Click “Create function”.



**Figure 3.13 Create function**



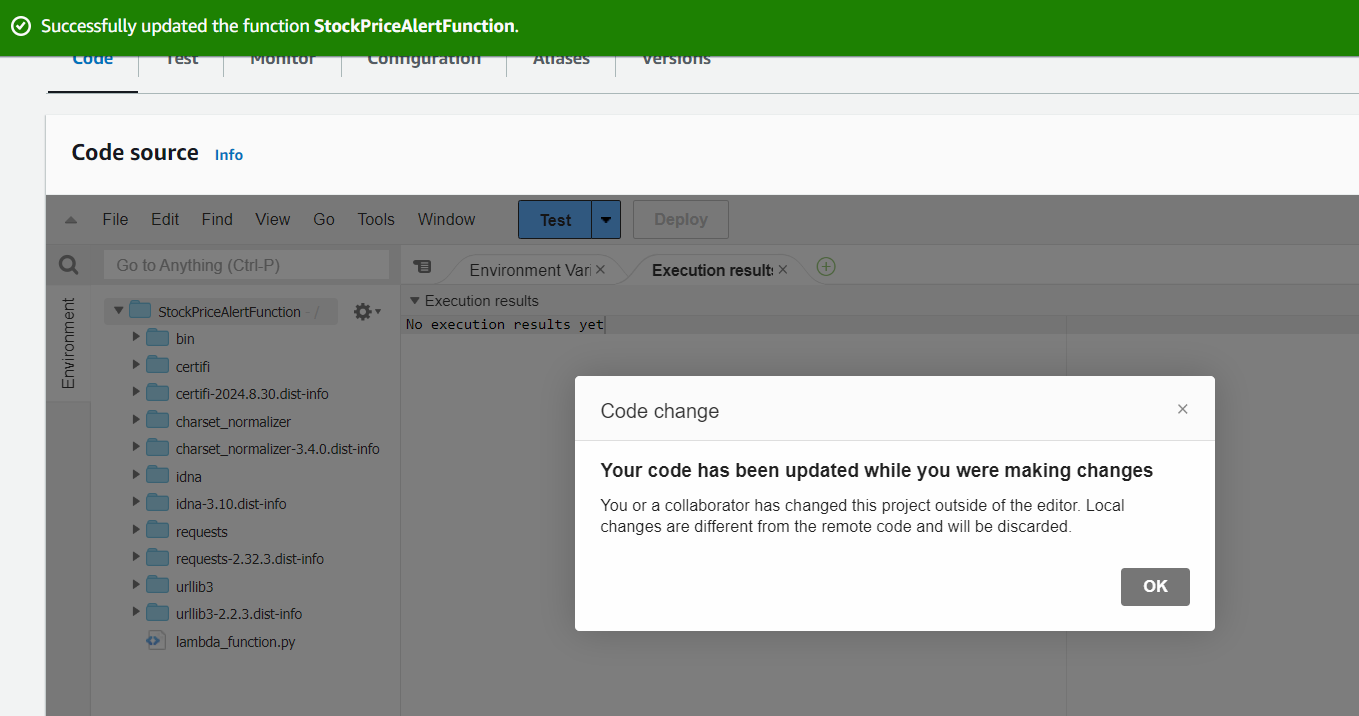
**Figure 3.14 Select Execution role**



**Figure 3.15 Successfully Created Function**

10. Upload the ZIP File to AWS Lambda

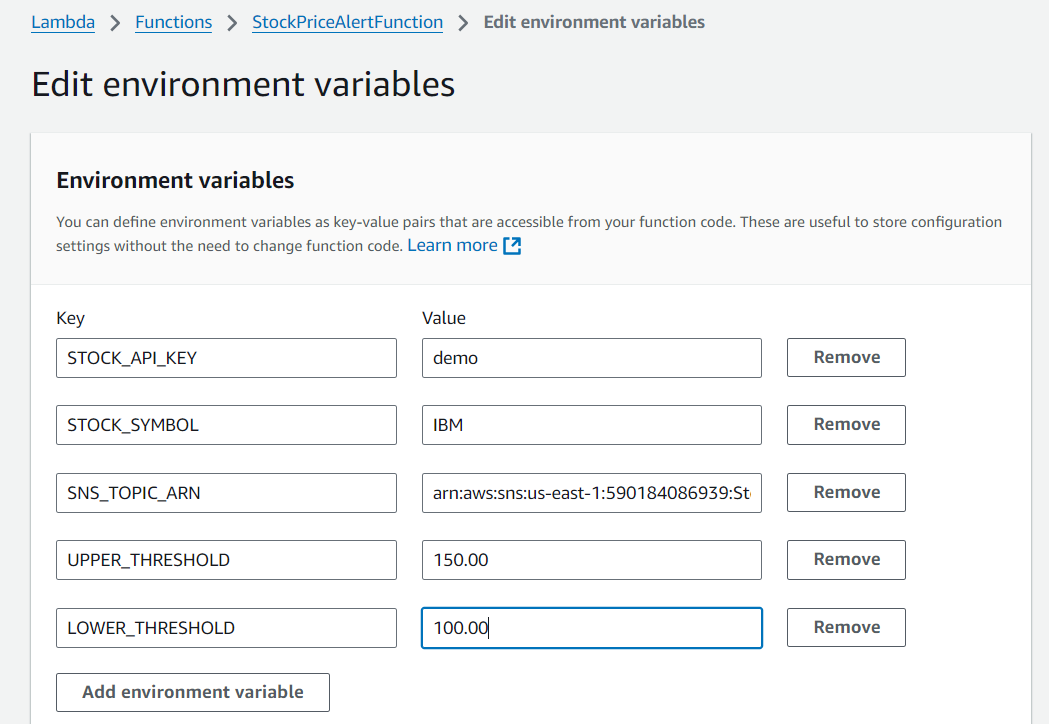
* Open the AWS Lambda Console in your browser.
* Go to your existing Lambda function.
* In the Code section, click Upload and choose the lambda\_function.zip file you created.
* Click Save to finalize the upload.



**Figure 3.16 Upload the zip file to Lambda Console**

11. Set Environment Variables in Lambda

* Scroll down to the **Environment Variables** section.
* Click **Edit** and add the following key-value pairs:
  + **STOCK\_API\_KEY:** Your Alpha Vantage API key.
  + **STOCK\_SYMBOL:** The stock symbol you want to monitor (e.g., AAPL).
  + **SNS\_TOPIC\_ARN:** The ARN of the SNS topic created earlier for notifications.
  + **PRICE\_THRESHOLD:** The stock price threshold that triggers alerts ( both UPPER\_THRESHOLD, LOWER\_THRESHOLD)
* Click **Save** to apply the changes.



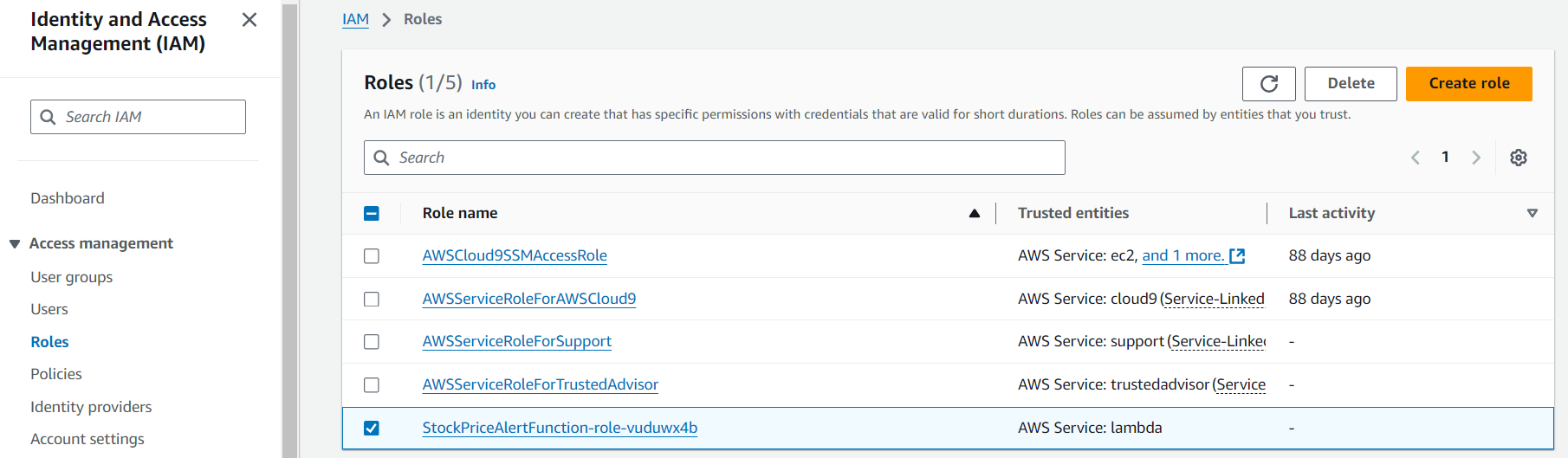
**Figure 3.17 Edit Environment Variables**



**Figure 3.18 Successfully updated the function**

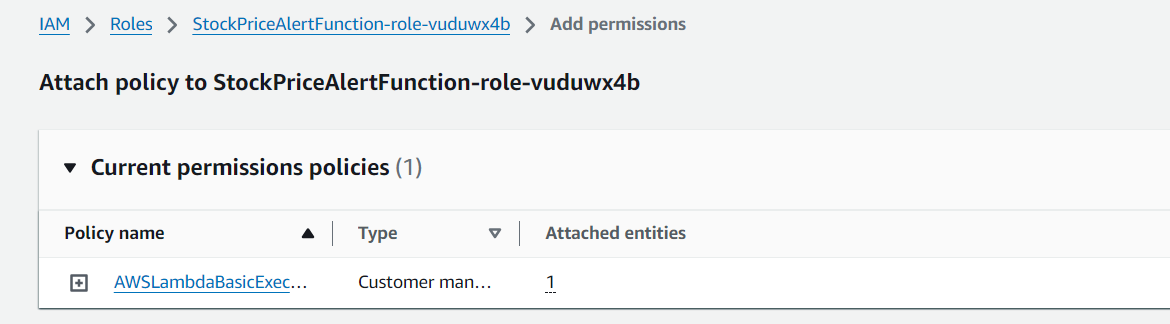
12. Set Lambda Permissions for SNS

* In the AWS IAM Console, go to Roles.



**Figure 3.19 IAM roles**

* Locate and select the execution role associated with your Lambda function.



**Figure 3.20 Lambda function Add Permissions**

* Click Attach Policies and create a custom inline policy with the following permissions to allow Lambda to publish to SNS

**Code:**

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "logs:CreateLogGroup",

"Resource":"arn:aws:logs:us-east-

1:590184086939:\*"

},

{

"Effect": "Allow",

"Action": [

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": [

"arn:aws:logs:us-east-1:590184086939:log-

group:/aws/lambda/StockPriceAlertFunction:\*"

]

},

{

"Effect": "Allow",

"Action": "sns:Publish",

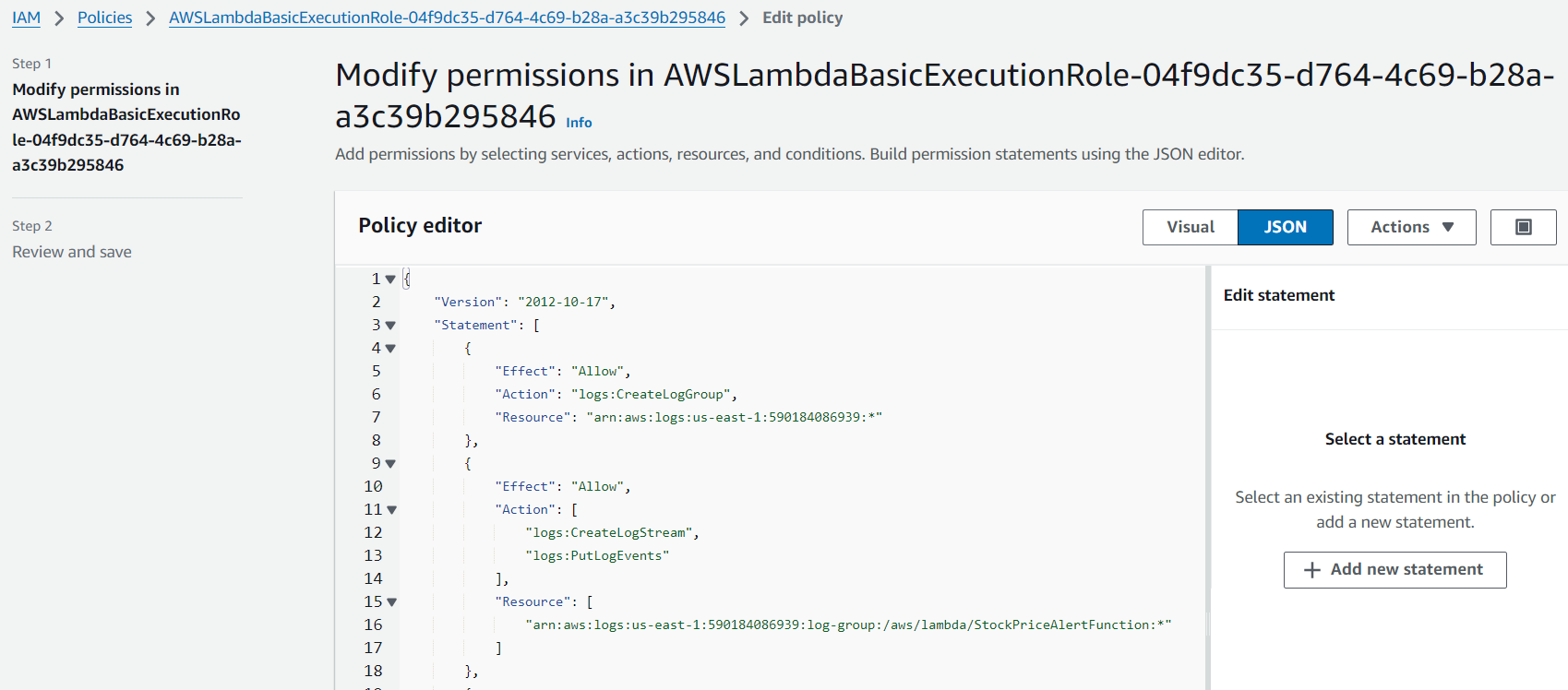
"Resource": "arn:aws:sns:us-east-

1:590184086939:StockPriceAlerts"

}

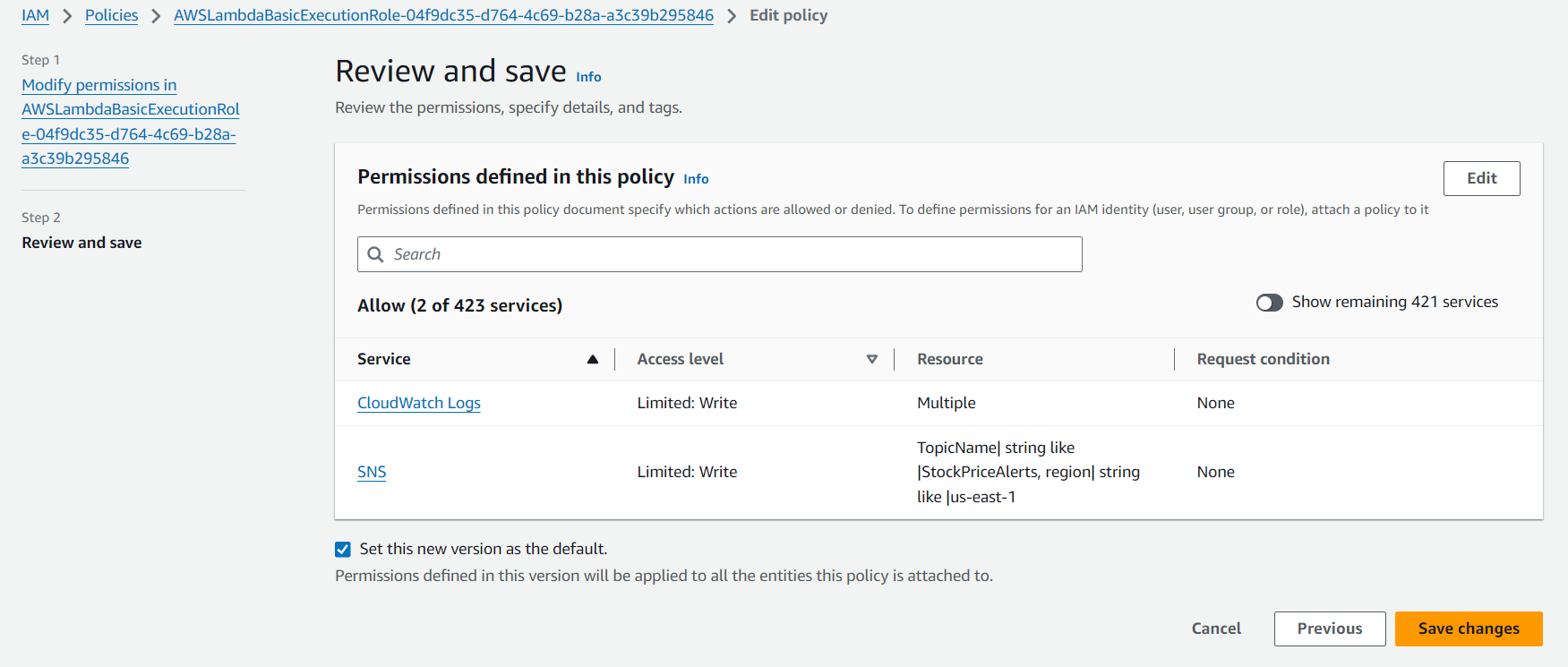
]

}



**Figure 3.21 Modify Permissions of Lambda Function**

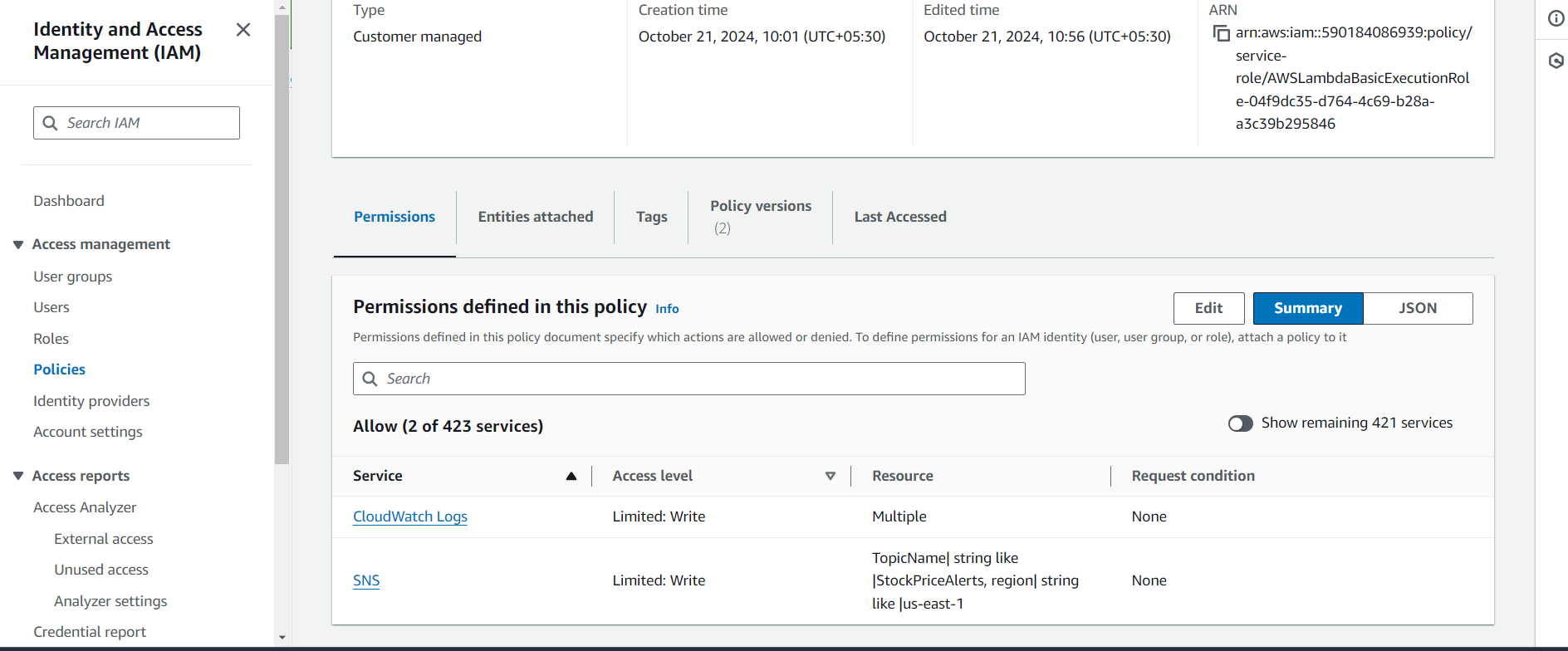
* Review and save the IAM role Policy



**Figure 3.22 Review and Save**



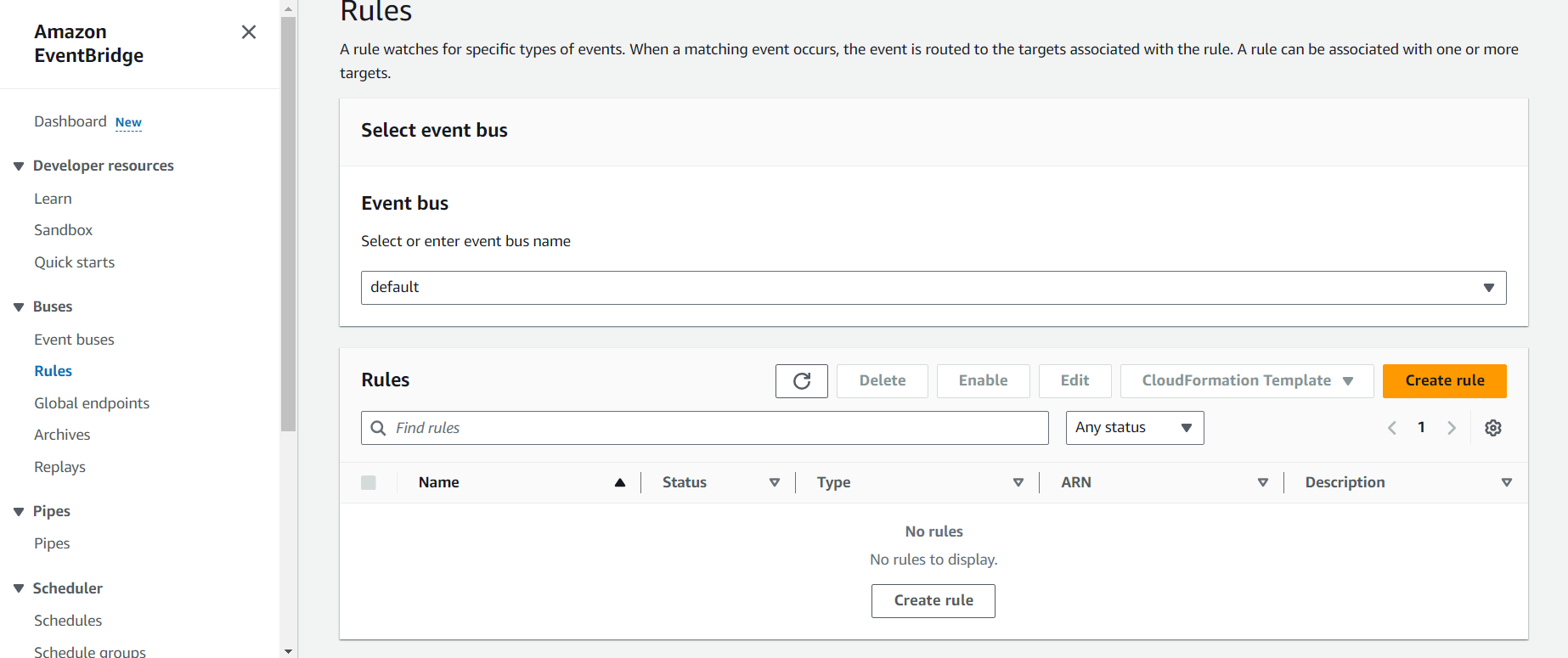
**Figure 3.23 Policy Updated**



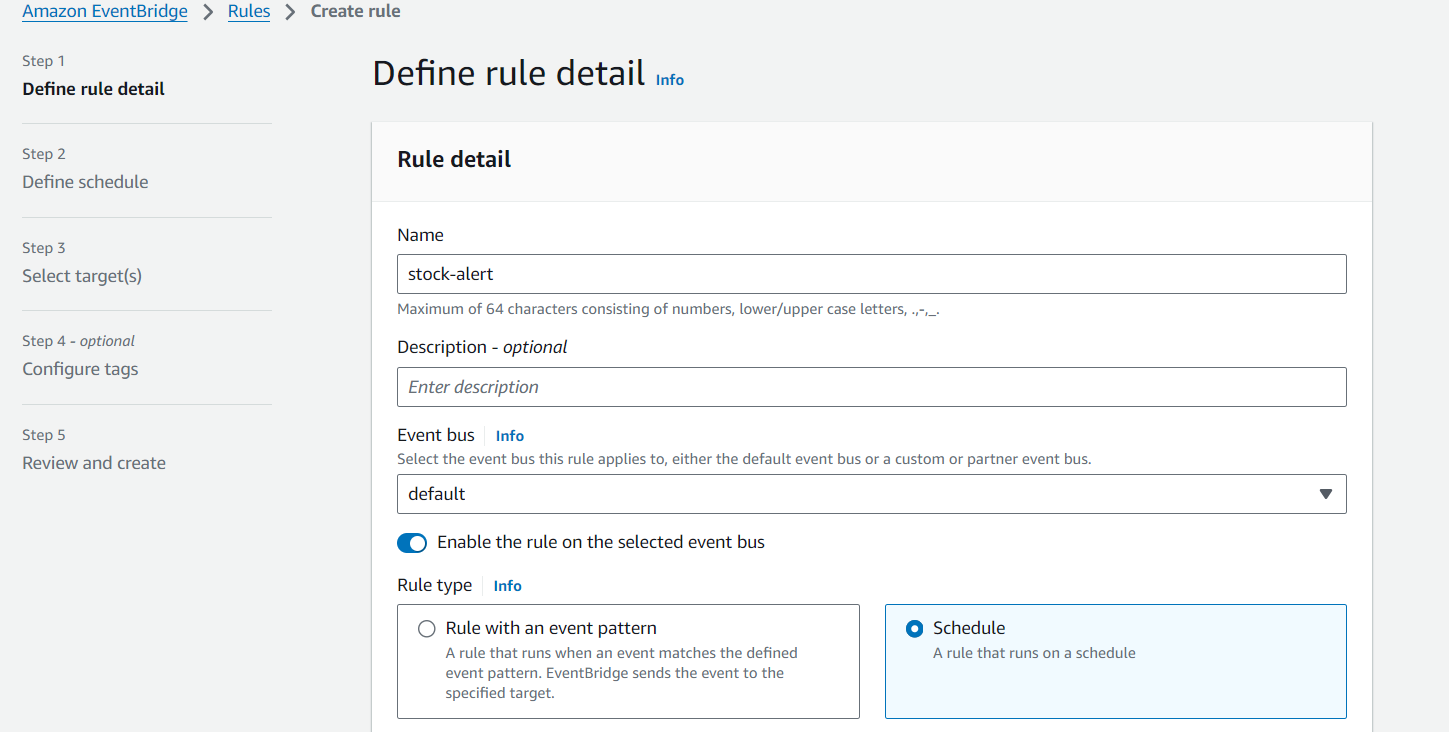
**Figure 3.24 Permissions defined in the Policy**

13. Schedule the Lambda Function with CloudWatch

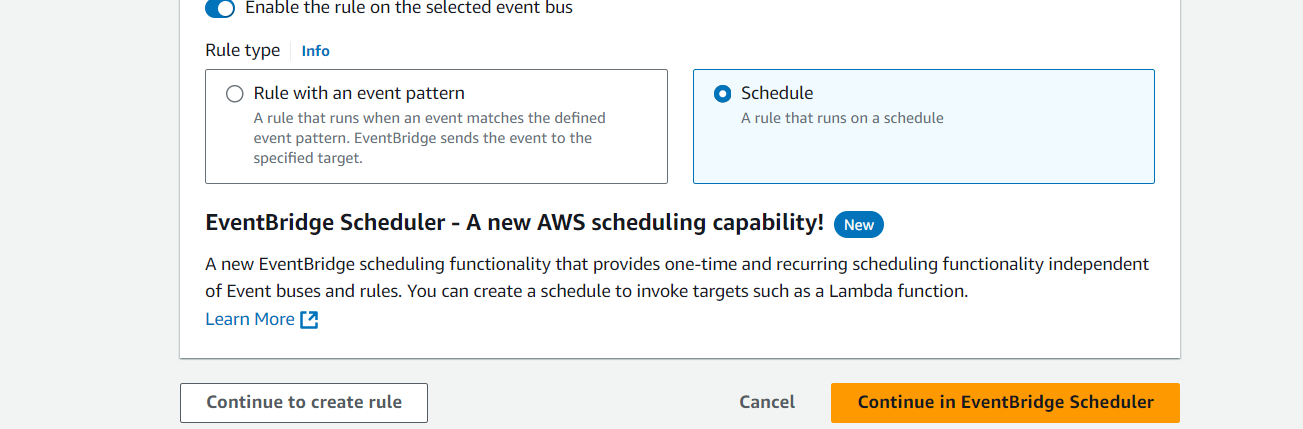
* Go to the Amazon EventBridge console and select "Create Rule."
* Name the rule, optionally add a description, and choose the default event bus.
* Select "Schedule" as the rule type for setting a recurring or one-time event.
* Set the schedule and choose the Lambda function as the target.
* Review the rule details and create it to activate the schedule.



**Figure 3.25 Create Rule**

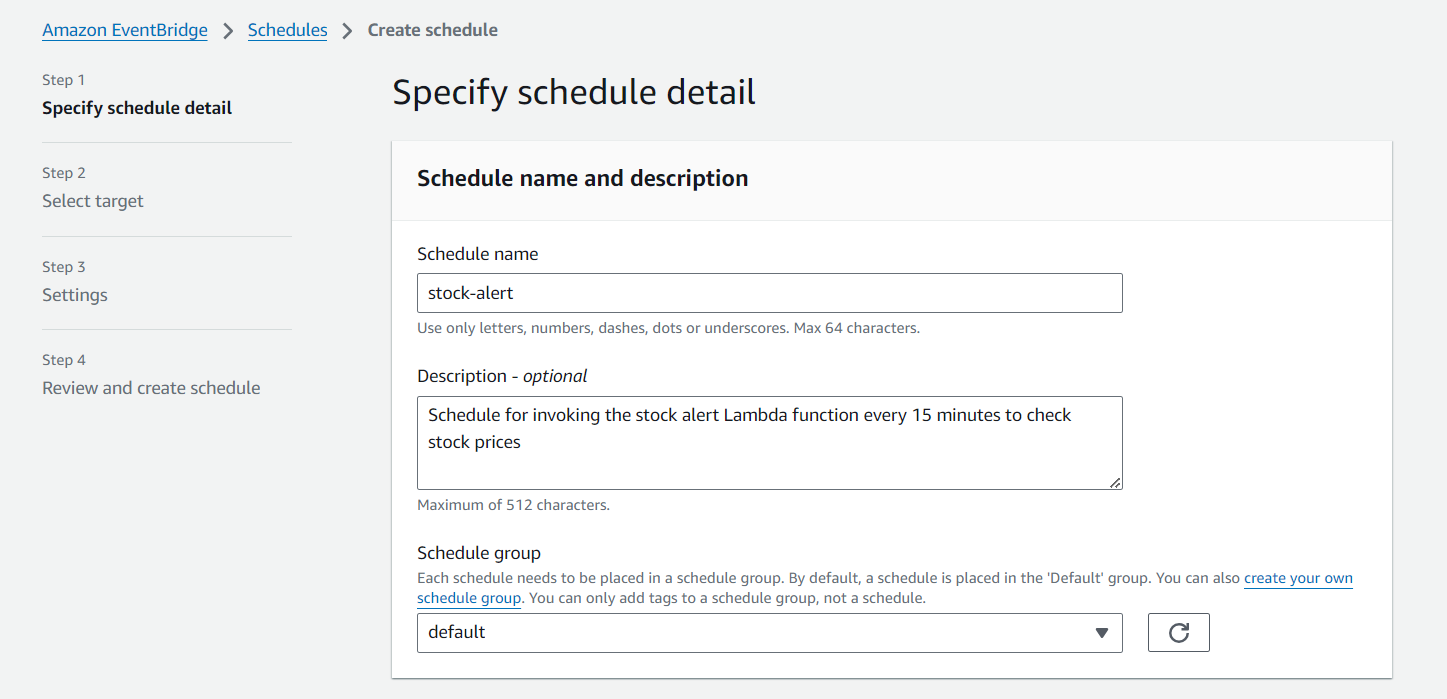


**Figure 3.26 Define rule Detail**

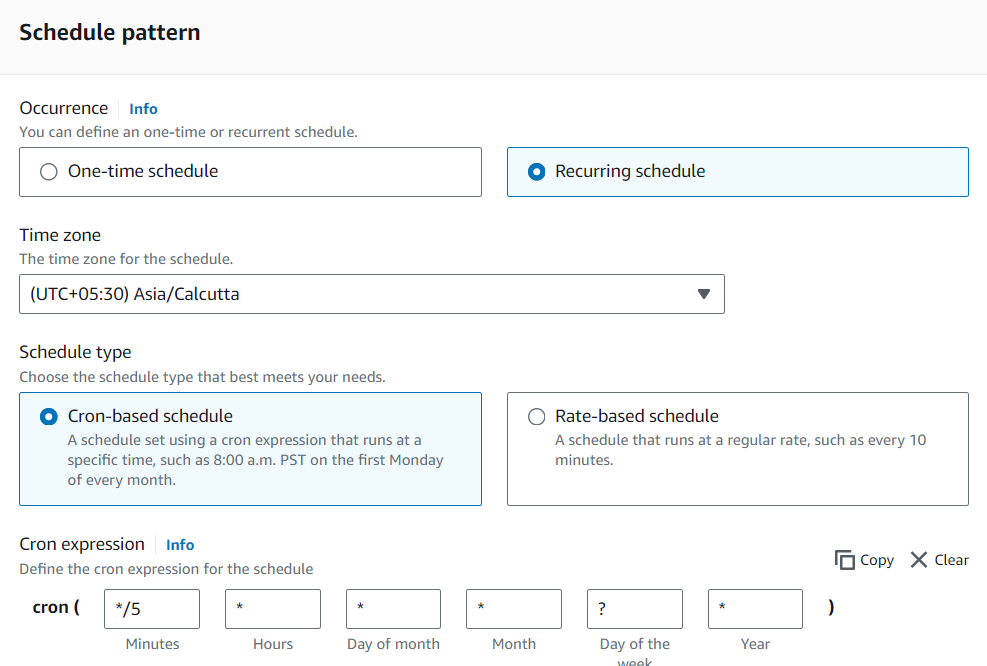


**Figure 3.27 Create Schedule**

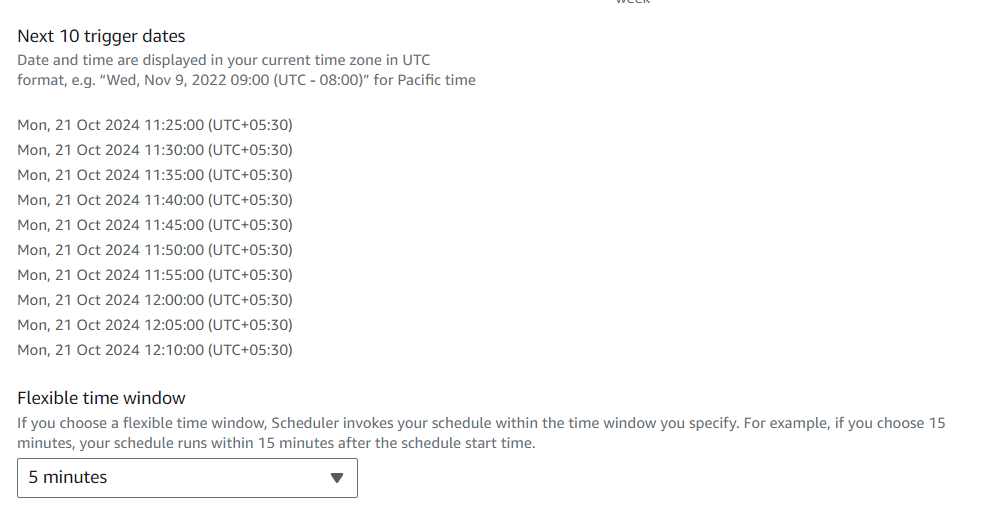
* Set the event source:
  + Event Source: Choose EventBridge (CloudWatch Events).
  + Rule Type: Select Schedule expression.
  + Schedule Expression: Enter rate(15 minutes) to trigger the function every 15 minutes.



**Figure 3.28 Specify Schedule detail**

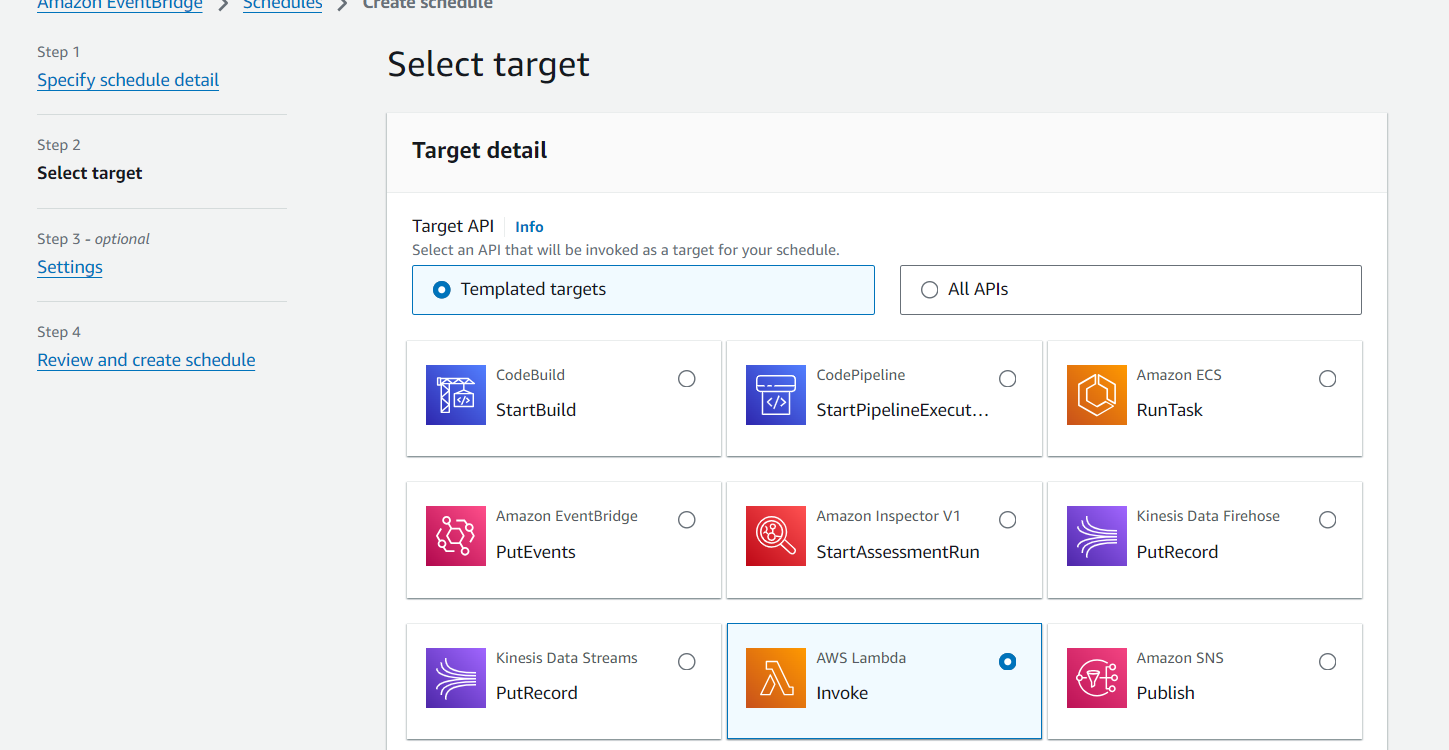


**Figure 3.29 Define Schedule Pattern**

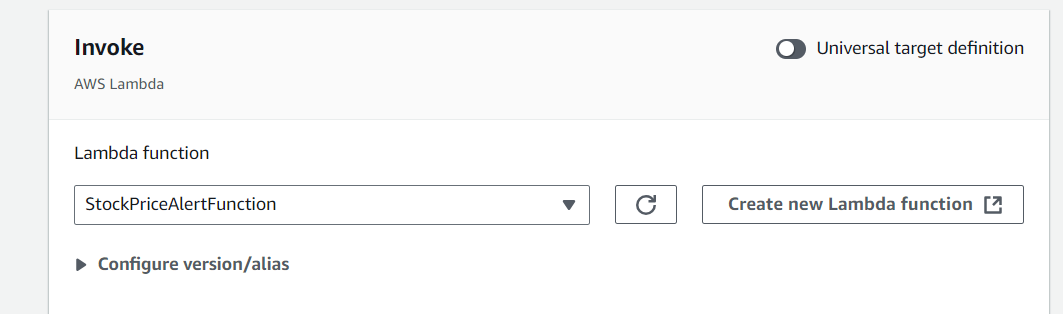


**Figure 3.30 Trigger Dates & Time window**

* Add the Lambda function as the target:
  + Under Target, choose Lambda Function.
  + Select your StockPriceAlertFunction from the dropdown.

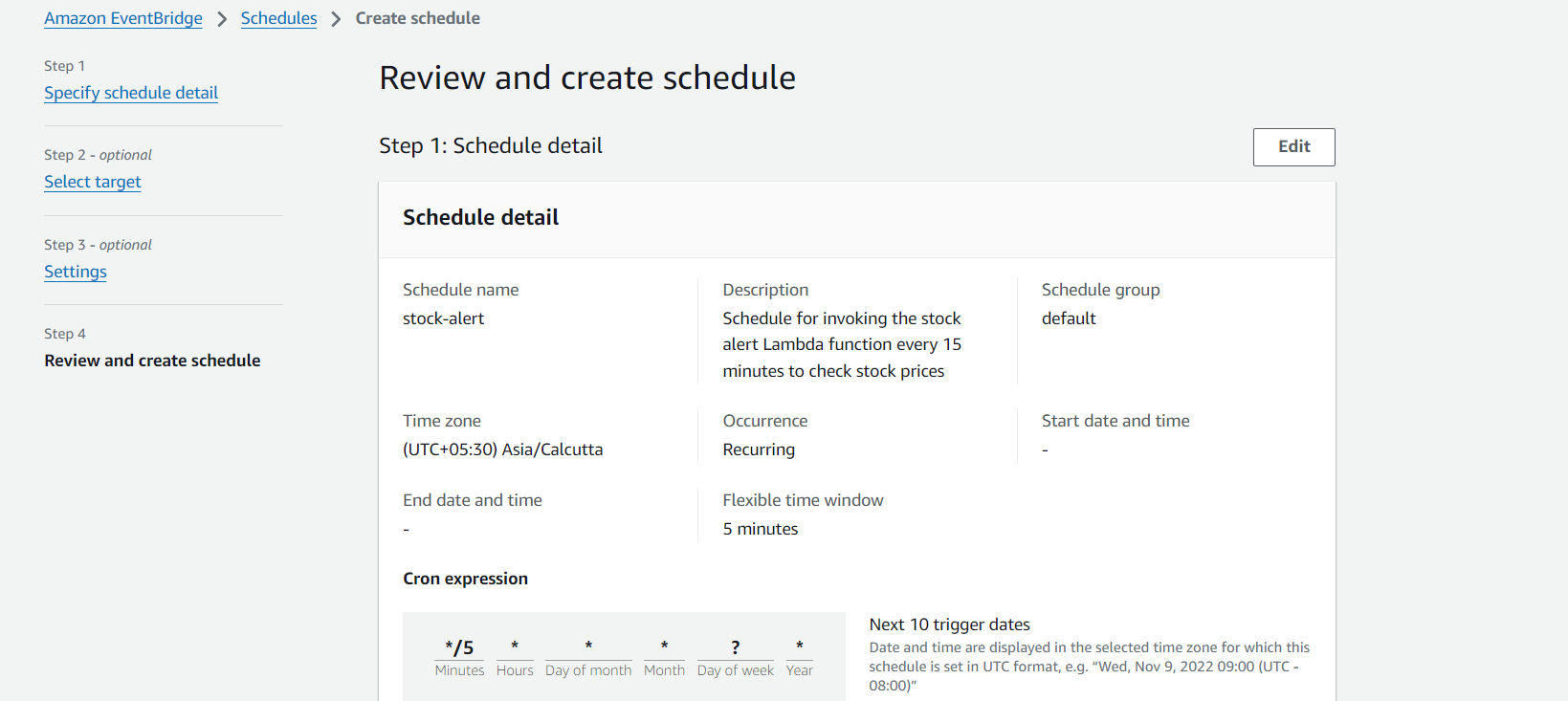


**Figure 3.31 Select Target**

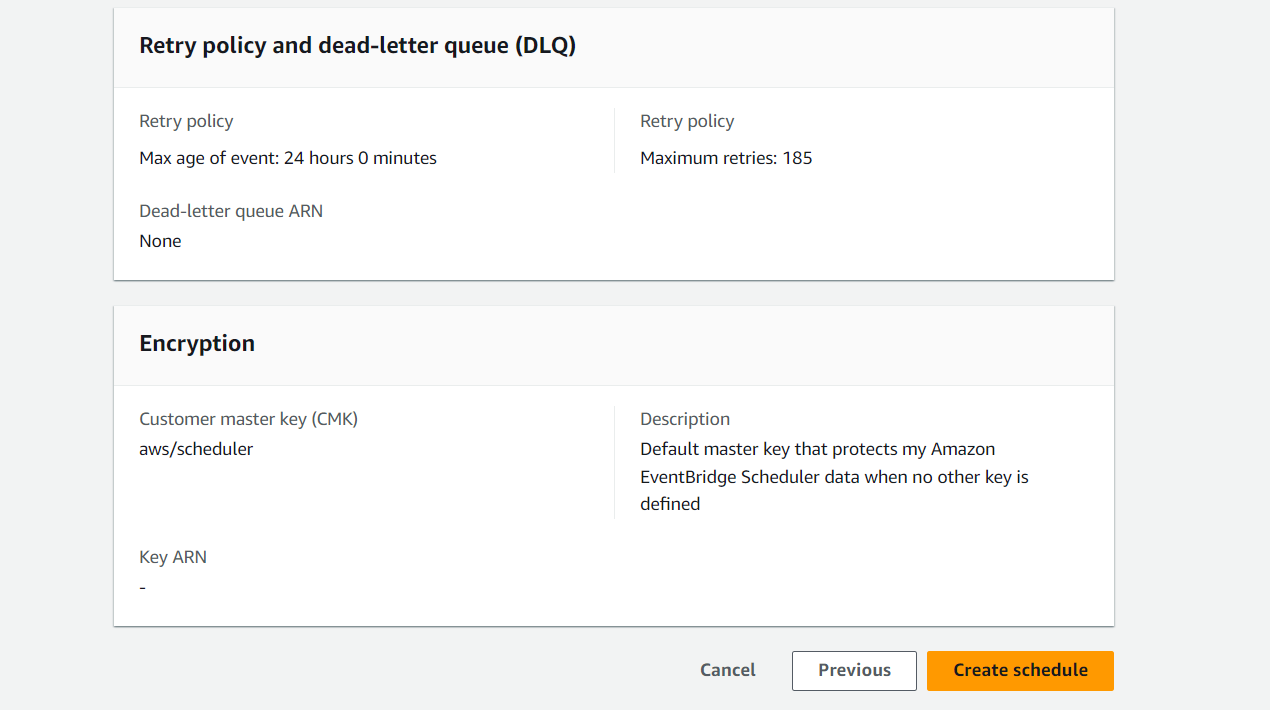


**Figure 3.32 Invoke (Lambda function)**

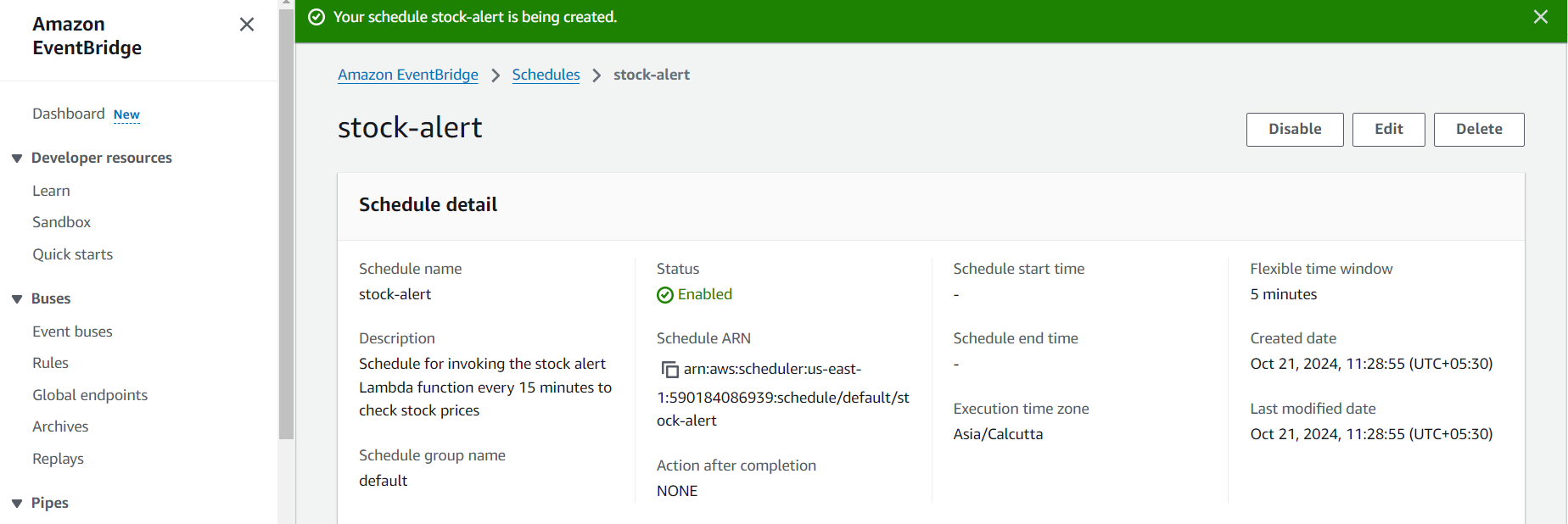
* Review and create the rule.



**Figure 3.33 Review Schedule**



**Figure 3.34 Create Schedule**



**Figure 3.35 Schedule Successfully Created**

14. Create a Test Event for Lambda Function

* In the AWS Lambda Console, navigate to your Lambda function and click on the Test button at the top right of the function's detail page.
* If this is your first test, you'll be prompted to Configure test events.
  + Select Create new test event.
  + Name your event, for example, TestEvent.
  + Use the following simple JSON structure for the test event

**Code:**

{

"STOCK\_API\_KEY": "demo",

"STOCK\_SYMBOLS": "IBM",

"SNS\_TOPIC\_ARN": "arn:aws:sns:us-east-

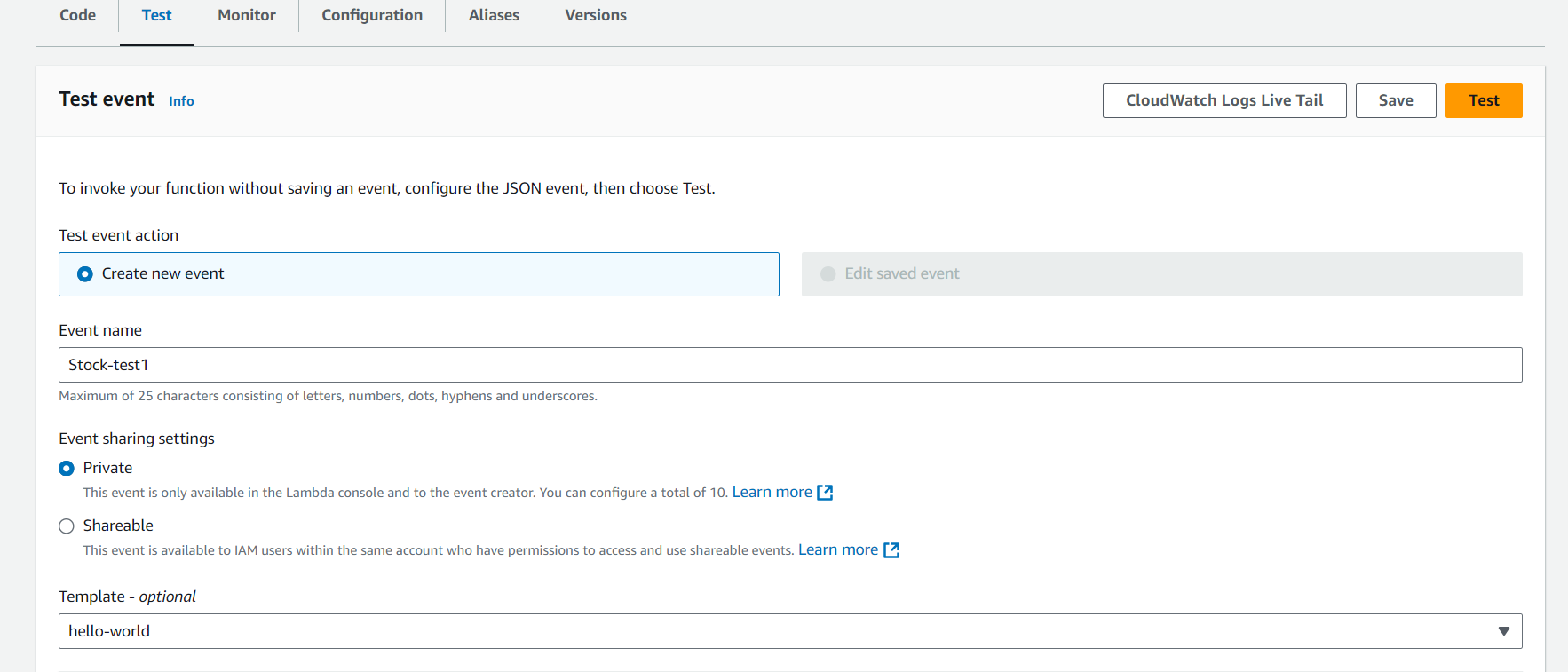
1:590184086939:StockPriceAlerts",

"UPPER\_THRESHOLD": 150.00,

"LOWER\_THRESHOLD": 100.00,

"message": "Check stock price for IBM Inc."

}



**Figure 3.36 Create Test Event**

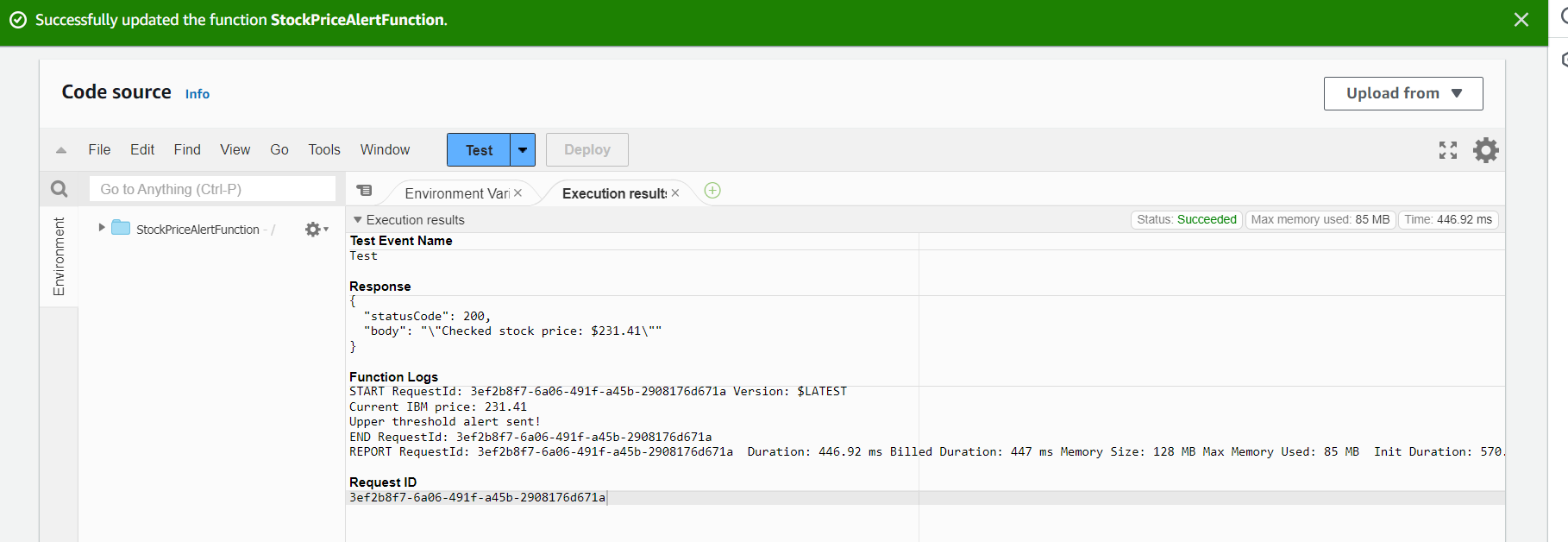


**Figure 3.37 Event JSON**

* Click Create and then Test to trigger the Lambda function using this event.

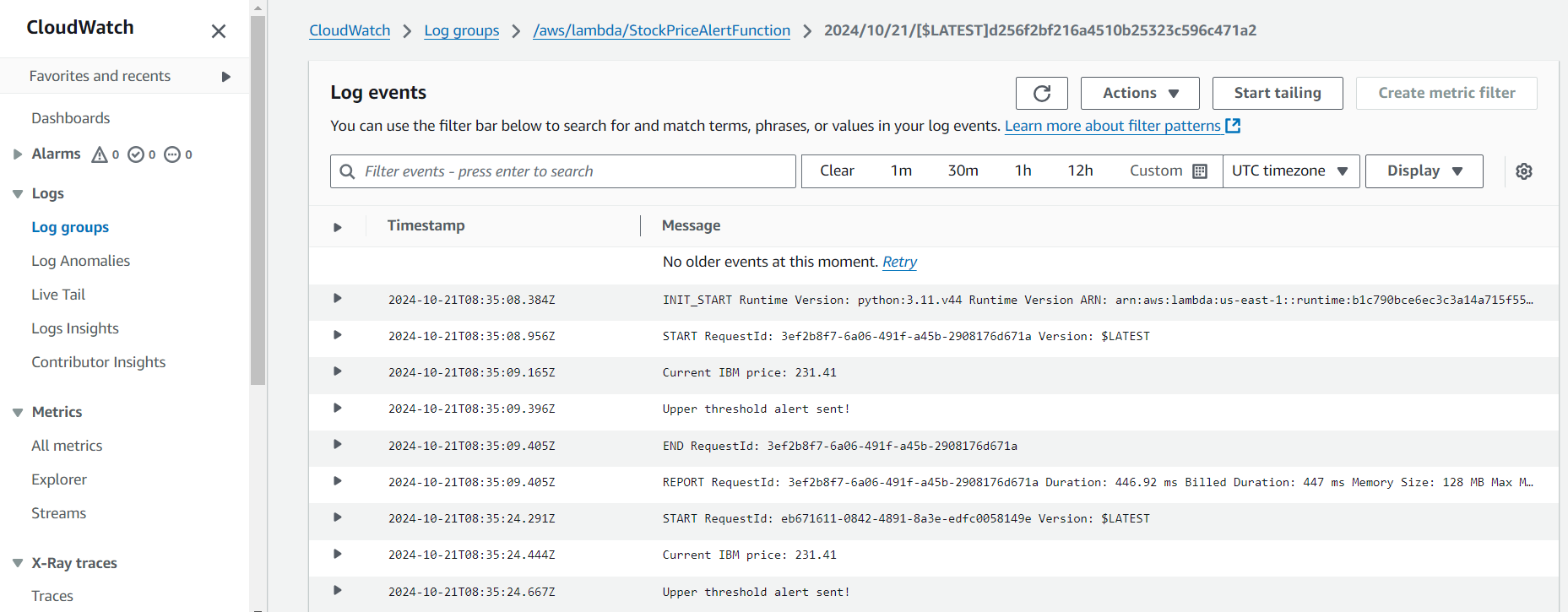
15. Test the System

* **Manual Test:** In the Lambda console, you can manually test your function by clicking the Test button after configuring a test event, as explained earlier.



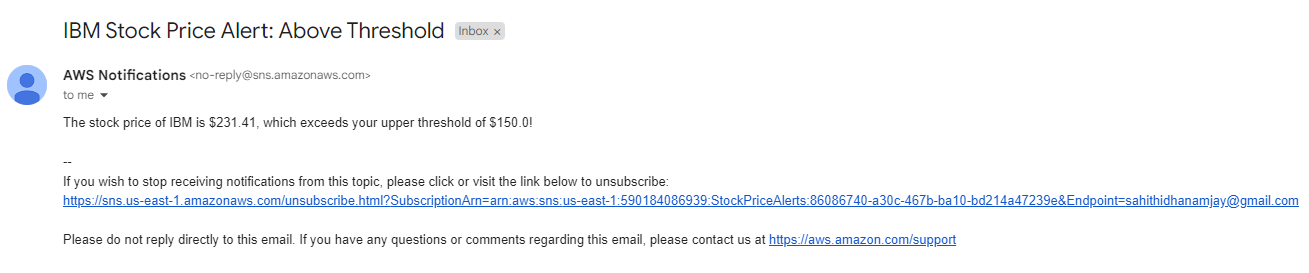
**Figure 3.38 Manual Test of Lambda Function**

* **Monitor Logs:** To track the function’s performance and troubleshoot any issues, you can access detailed logs in CloudWatch Logs. This will show each execution of the Lambda function, including any errors or successful triggers.



**Figure 3.39 Monitor the Log Events (CloudWatch)**

* **Scheduled Execution:** If the Lambda function is set to run on a schedule via CloudWatch, you can wait for the automatic trigger to execute it at the defined intervals (e.g., every 15 minutes).



**Figure 3.40 Alert Triggered to the Endpoint**

**CHAPTER 4**

**CHALLENGES FACED**

* **Stock API Rate Limits**:
* The Alpha Vantage API’s rate limits restricted the number of requests, making it difficult to track multiple stocks simultaneously. Optimizing API calls and caching data helped manage these limits effectively.
* **Threshold Accuracy**:
* Ensuring that alerts were triggered only for significant stock price changes required fine-tuning the price threshold logic, avoiding unnecessary or missed notifications.
* **Notification Latency**:
* Ensuring real-time delivery of alerts was crucial, but occasional delays occurred. Monitoring Lambda execution and SNS delivery helped minimize these delays.
* **Scalability**:
* As the system scaled, managing Lambda execution and API rate limits became critical. Lambda’s serverless model helped handle increased load without manual intervention.
* **Cost Management**:
* Balancing the costs of Lambda invocations and API requests required careful optimization of the function’s scheduling and efficient API usage to avoid unnecessary expenses.
* **Security and Permissions**:
* Configuring IAM roles to securely manage service interactions was essential. Implementing least-privilege permissions ensured that each service had only the necessary access.
* **API Failures and Timeouts**:
* Occasional API failures or timeouts interrupted stock data fetching. Implementing retries and error handling mechanisms ensured that the system remained reliable.
* **Cloud9 Package Installation**:
* Packaging the requests library for Lambda in Cloud9 required proper zipping of dependencies. Ensuring compatibility post-deployment resolved initial setup challenges.