TITLE: Building a Real-Time Stock Price Alert System on AWS

Chapter-1: INTRODUCTION

In an increasingly volatile financial market, timely access to stock price information is critical for investors and traders. The Stock Price Alerting System is designed to empower users by providing real-time notifications about significant stock price changes. This system enables users to monitor their investments efficiently and make informed decisions promptly.

Purpose of the System: The primary purpose of the Stock Price Alerting System is to automate the process of tracking stock prices and alerting users when specific price thresholds are reached. By leveraging cloud technologies, this system minimizes the need for constant manual monitoring, allowing users to focus on strategy and decision-making rather than real-time price tracking.

Key Features

- Real-Time Monitoring: The system continuously fetches stock prices from reliable data sources to provide up-to-the-minute information.
- Customizable Alerts: Users can define specific price points for their chosen stocks, receiving notifications via email or SMS when those thresholds are met.
- Automated Notification Delivery: Leveraging services like Amazon SNS (Simple Notification Service), the system automates the delivery of alerts, ensuring that users are informed immediately of price changes.
- Scalable and Cost-Effective: Built on a serverless architecture using AWS services like Lambda, this system can efficiently handle varying loads and minimize operational costs.
- Development Using AWS Cloud9: The system's setup is simplified using AWS Cloud9, a cloud-based Integrated Development Environment (IDE) that enables users to create, edit, and manage their Lambda function code without the need for local installations or configurations.

System Components

1. **AWS Lambda:** A serverless compute service that runs the monitoring and alerting logic without requiring dedicated server infrastructure.
2. **Amazon SNS:** A messaging service that facilitates the delivery of alerts via email and SMS.
3. **Amazon CloudWatch:** A monitoring service that schedules the execution of Lambda functions and provides logging capabilities.
4. **AWS Cloud9:** A cloud IDE used to develop and manage the Lambda function, making it easy for users to create and test their code in a collaborative environment.
5. **Third-Party API (e.g., Alpha Vantage):** A data source for fetching real-time stock price information, essential for triggering alerts based on user-defined criteria.

Chapter-2:EXECUTIVE SUMMARY

The **Stock Price Alerting System** is a cloud-based solution designed to automate stock price monitoring and provide real-time alerts to users. This system enhances decision-making for investors and traders by ensuring timely notifications of significant price movements, eliminating the need for constant market surveillance.

System Overview

The architecture of the Stock Price Alerting System leverages multiple Amazon Web Services (AWS):

- **AWS Lambda**: Executes the core application logic without server management, retrieving stock prices at scheduled intervals and checking against user-defined alert criteria.
- **Amazon SNS (Simple Notification Service)**: Delivers alerts via email or SMS, ensuring users are promptly informed when stock prices reach specified thresholds.
- **Amazon CloudWatch**: Monitors Lambda function performance and execution, allowing for the scheduling of regular price checks and ensuring reliable system operation.
- **AWS Cloud9**: A cloud-based Integrated Development Environment (IDE) that simplifies the development, testing, and management of Lambda functions, enabling users to code without local infrastructure.
- **Third-Party API (e.g., Alpha Vantage)**: Integrates with APIs to fetch real-time stock price data, facilitating effective monitoring of market fluctuations for selected stocks.

Key Benefits

The Stock Price Alerting System offers several advantages:

- **Real-Time Updates**: Immediate alerts regarding stock price changes enable prompt responses to market movements.
- **Customizable Notification Settings**: Users can tailor alert preferences based on specific price thresholds, ensuring relevant notifications.
- **Automation and Efficiency**: Automating the monitoring process saves time and reduces the risk of missing critical trading opportunities.
- **Scalability**: The serverless architecture allows seamless handling of varying loads, accommodating the needs of both individual investors and larger trading teams.
- **Cost-Effectiveness**: Utilizing AWS services incurs costs only when in use, making it an economical choice for users seeking robust monitoring solutions.

Target Audience

The system is designed for individual investors, day traders, financial analysts, and anyone who needs timely stock price information to make informed trading decisions. It is suitable for both beginners who are new to stock trading and experienced professionals who want advanced monitoring featur

Chapter 3: IMPLEMENTATION

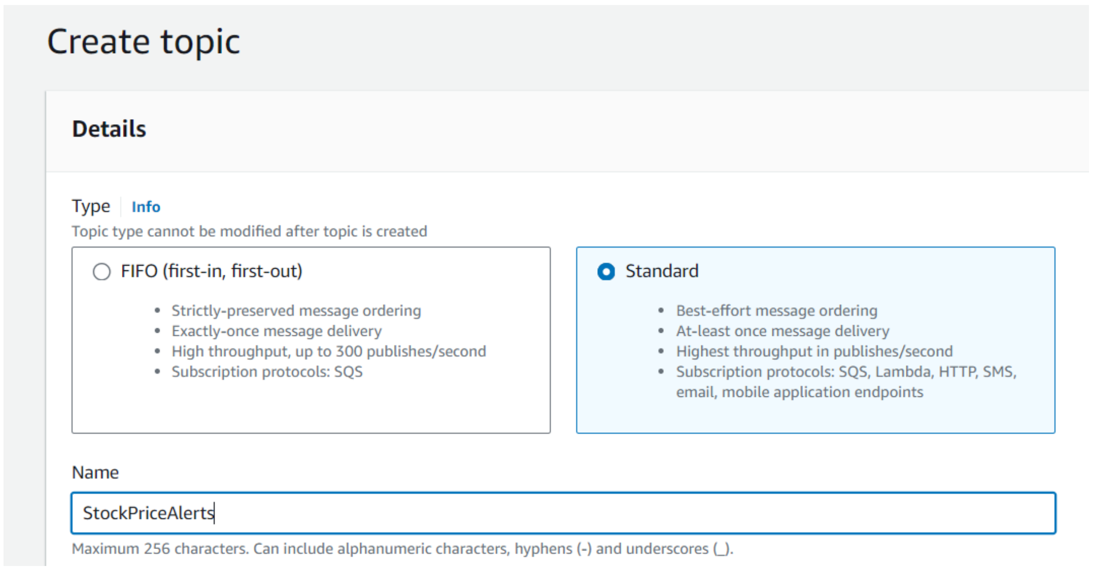
1. Create an SNS Topic for Notifications

Step 1: Navigate to Amazon SNS

- In the services menu, search for and select **SNS**.

Step 2: Create a New Topic

- Click on the **Topics** option in the left sidebar.
- Click on the **Create topic** button.
- Select **Standard** as the type of topic.



Create topic

Details

Type Info
Topic type cannot be modified after topic is created

( ) FIFO (first-in, first-out)
- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

[✓ Standard]
- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

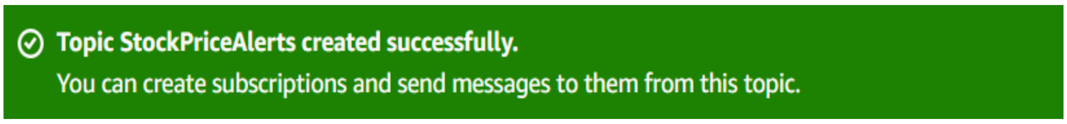
Name
StockPriceAlerts
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Step 3: Configure Topic Details

- Fill in the following details:
  - **Name**: Enter a unique name for your topic (e.g., StockPriceAlerts).

Step 4: Create the Topic

- Click on the **Create topic** button. You will be taken to the topic details page.

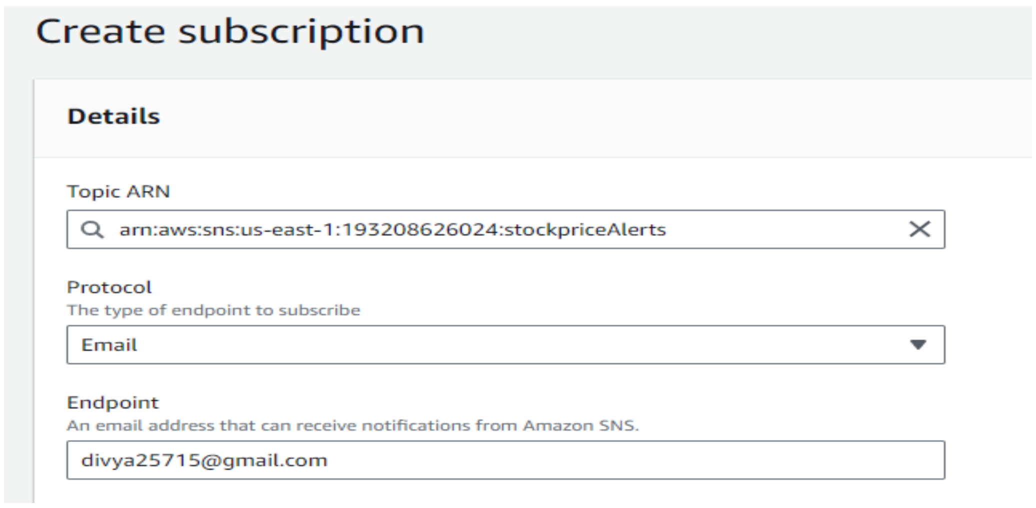


Topic StockPriceAlerts created successfully.
You can create subscriptions and send messages to them from this topic.

4.2 Subscribing to the SNS Topic

Step 5: Create a Subscription

- On the topic details page, click on the **Create subscription** button.



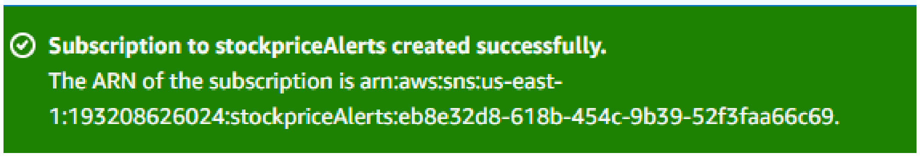
Create subscription

Details

Topic ARN
arn:aws:sns:us-east-1:193208626024:stockpriceAlerts

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
divya25715@gmail.com



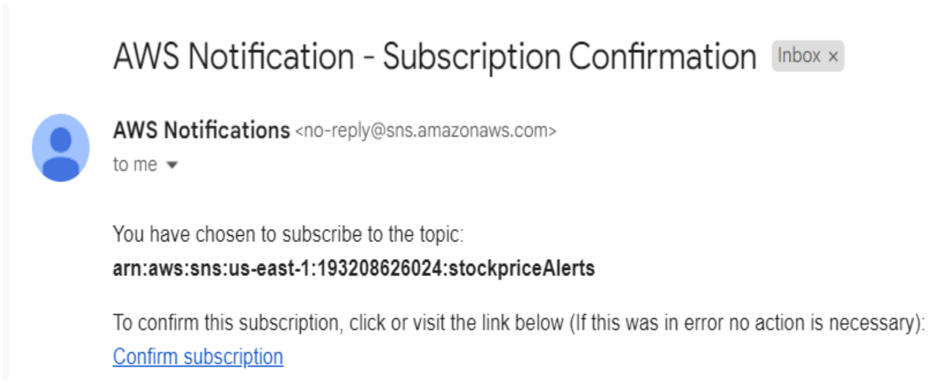
Subscription to stockpriceAlerts created successfully.
The ARN of the subscription is arn:aws:sns:us-east-1:193208626024:stockpriceAlerts:eb8e32d8-618b-454c-9b39-52f3faa66c69.

Step 6: Configure Subscription Details

- Fill in the following details:
    - **Protocol**: Choose the protocol for notifications (e.g., **Email** or **SMS**).
    - **Endpoint**: Enter the email address or phone number that will receive notifications.

Step 7: Confirm Subscription

- If you selected **Email**, check your email for a subscription confirmation message and click the confirmation link.



AWS Notification - Subscription Confirmation Inbox ×

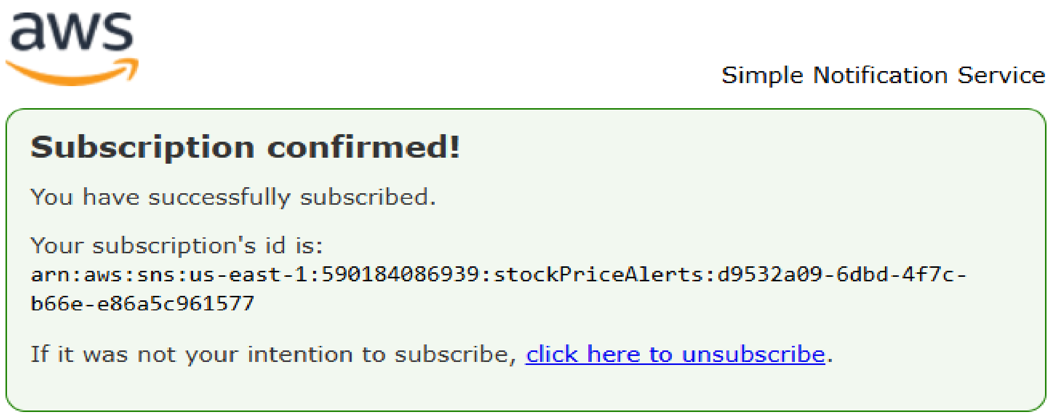
AWS Notifications <no-reply@sns.amazonaws.com>
to me

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:193208626024:stockpriceAlerts

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Step 8: Verify Subscription:

- After confirming, the subscription status should change to **Confirmed** in the SNS console.


Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
`arn:aws:sns:us-east-1:590184086939:stockPriceAlerts:d9532a09-6dbd-4f7c-b66e-e86a5c961577`

If it was not your intention to subscribe, click here to unsubscribe.

2.Setting Up AWS Cloud9

Step 1: Access AWS Cloud9

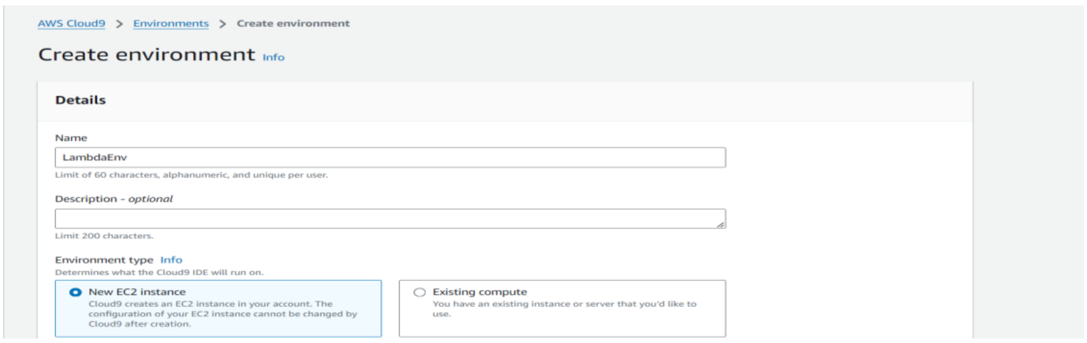
1. **Navigate to Cloud9**:
   - In the services menu, type **Cloud9** and select it.

Step 2: Create a New Cloud9 Environment

1. Click on "Create environment".
2. **Enter Environment Name**:
   - Name it LambdaEnv.
3. **Click "Next"**.

Step 3: Configure the Environment

1. **Environment Settings**:
   - **Instance type**: Select t2.micro or similar.
   - **Platform**: Ensure **Amazon Linux** is selected.
   - **Cost-saving setting**: Configure as preferred.
2. **Click "Next"**.
3. **Review settings** and click **"Create environment"**.


AWS Cloud9 > Environments > Create environment

Create environment Info

Details

Name
LambdaEnv
Limit of 60 characters, alphanumeric, and unique per user.

Description - *optional*

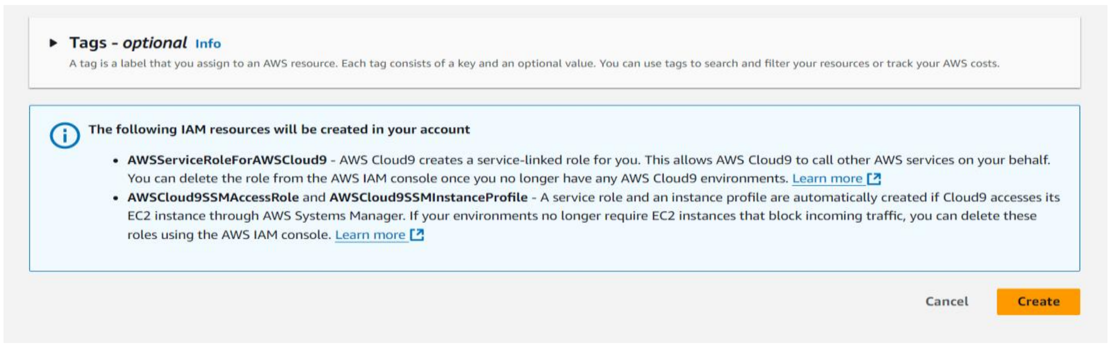
Limit 200 characters.

Environment type Info
Determines what the Cloud9 IDE will run on.

[✓ New EC2 instance] Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

Existing compute — You have an existing instance or server that you'd like to use.

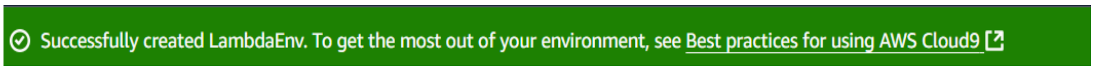
4. **Wait for the environment to be provisioned**.


Tags - *optional* Info

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

The following IAM resources will be created in your account

- **AWSServiceRoleForAWSCloud9** - AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. Learn more
- **AWSCloud9SSMAccessRole** and **AWSCloud9SSMInstanceProfile** - A service role and an instance profile are automatically created if Cloud9 accesses its EC2 instance through AWS Systems Manager. If your environments no longer require EC2 instances that block incoming traffic, you can delete these roles using the AWS IAM console. Learn more


Successfully created LambdaEnv. To get the most out of your environment, see Best practices for using AWS Cloud9

Step 4: Set Up the Development Environment

1. **Open Terminal**:
   - The terminal is located at the bottom of the IDE.
2. **Create a Directory**:
   - Run the following commands:

     # mkdir lambda_requests_function

     #cd lambda_requests_function
3. **Install Required Libraries**:
   - Install the requests library:

     #pip install requests -t .

➢ Go to a folder lambda_requests_function inside it run the below code as save it as lambda_function:

```
 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_SYMBOL}&interval=1min&apikey={STOCK_API_KEY}"

    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}")}
```

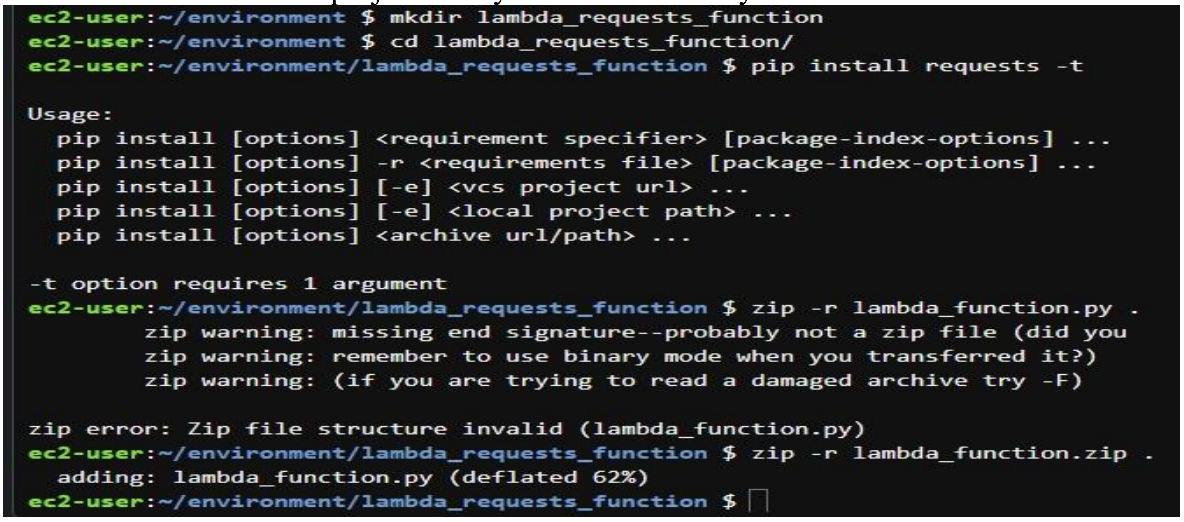
Step 6: Zip the Project for Lambda Deployment

1. **Zip the Directory Contents**:
   - Run the following command:

# zip -r lambda_function.zip .

This will create a file named `lambda_function.zip` that contains both the `requests` library and your `lambda_function.py` code

➢ Now download the whole project in to your local files in system



```
ec2-user:~/environment $ mkdir lambda_requests_function
ec2-user:~/environment $ cd lambda_requests_function/
ec2-user:~/environment/lambda_requests_function $ pip install requests -t

Usage:
  pip install [options] <requirement specifier> [package-index-options] ...
  pip install [options] -r <requirements file> [package-index-options] ...
  pip install [options] [-e] <vcs project url> ...
  pip install [options] [-e] <local project path> ...
  pip install [options] <archive url/path> ...

-t option requires 1 argument
ec2-user:~/environment/lambda_requests_function $ zip -r lambda_function.py .
        zip warning: missing end signature--probably not a zip file (did you
        zip warning: remember to use binary mode when you transferred it?)
        zip warning: (if you are trying to read a damaged archive try -F)

zip error: Zip file structure invalid (lambda_function.py)
ec2-user:~/environment/lambda_requests_function $ zip -r lambda_function.zip .
  adding: lambda_function.py (deflated 62%)
ec2-user:~/environment/lambda_requests_function $ 
```

3.Upload the ZIP File to AWS Lambda

Creating the Lambda Function

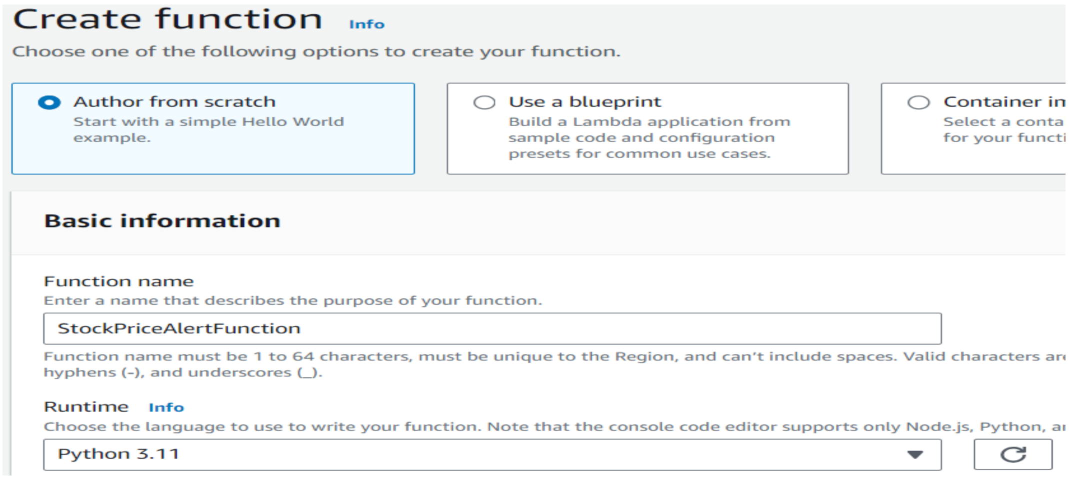
Step 1: Log in to the AWS Management Console

Step 2: Access AWS Lambda

- In the services menu, search for and select **Lambda**.

Step 3: Create a New Function

1. Click on **Create function**.
2. Choose **Author from scratch**.
   - **Function name:** Enter StockPriceAlertFunction.
   - **Runtime:** Select **Python 3.x**.



Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

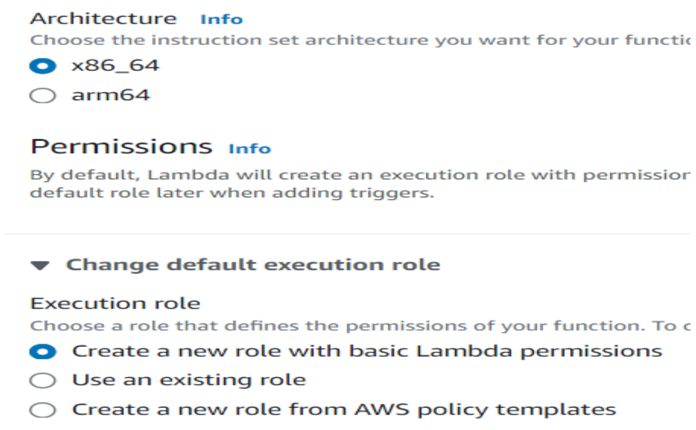
Container in
Select a conta
for your functi

Basic information

Function name
Enter a name that describes the purpose of your function.
StockPriceAlertFunction
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters ar
hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, a
Python 3.11

   - **Execution role:** Choose Create a new role with basic Lambda permissions.


Architecture Info

Choose the instruction set architecture you want for your functi

[✓ x86_64]

arm64

Permissions Info

By default, Lambda will create an execution role with permissior default role later when adding triggers.

Change default execution role

Execution role

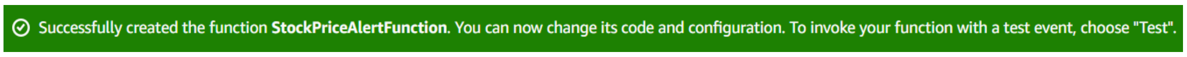
Choose a role that defines the permissions of your function. To c

[✓ Create a new role with basic Lambda permissions]

Use an existing role

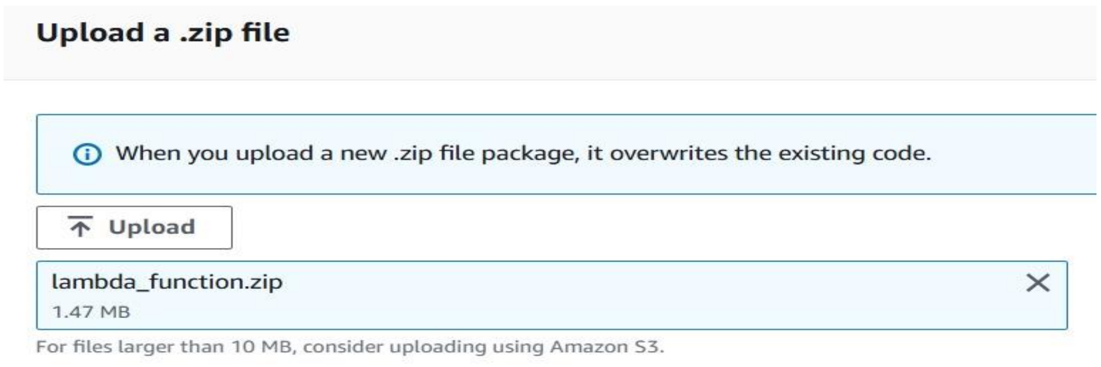
Create a new role from AWS policy templates

3. Click **Create function**.


Successfully created the function **StockPriceAlertFunction**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Step 4: Upload the Zip File to AWS Lambda

1. **Navigate to the Code Section**:
   - In the Lambda function page, scroll down to the **Function code** section.
2. **Upload the Zip File**:
   - Click on the **Upload** button and select **.zip file** from the dropdown menu.
   - Choose the lambda_function.zip file that you created earlier.


Upload a .zip file

When you upload a new .zip file package, it overwrites the existing code.

lambda_function.zip

1.47 MB

For files larger than 10 MB, consider uploading using Amazon S3.

3. **Save Your Changes**:
   - After the zip file is uploaded, click the **Save** button to apply the changes.

4. Set Lambda Permissions for SNS

Step 1: Open the IAM Console

- **Log in**: Go to the AWS Management Console and sign in with your AWS credentials.
- **Search for IAM**: In the search bar at the top, type **IAM** and select **IAM** from the dropdown list to open the IAM Console.
  Step 2: Navigate to Roles

- In the IAM Console, look at the left sidebar and click on **Roles** to view a list of IAM roles in your account.
  Step 3: Find the Lambda Execution Role
- In the list of roles, find the role that was created for your Lambda function. The role name often includes the name of the Lambda function for easier identification.
- Click on the role name to open the role's details page.
  Step 4: Attach the SNS Publish Policy
- On the role details page, navigate to the **Permissions** tab.
- Click on the **Add permissions** button and select **Attach policies** from the dropdown.
  Step 5: Create a Custom Policy (if necessary)
- If a suitable policy already exists, you can attach it directly. Otherwise, you will need to create a new policy:
  - Click on **Create policy**.
  - Go to the **JSON** tab.

Step 6: Add the Policy JSON

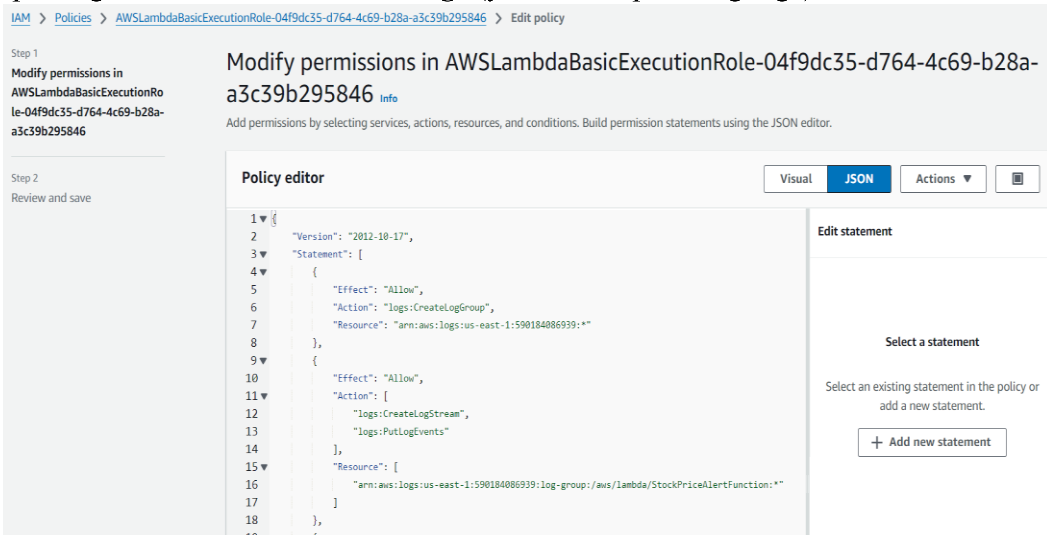
- In the JSON editor, paste the following policy, modifying the placeholders accordingly:
  Json

```
{
   "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"
        }
    ]
```

- **Replace the placeholders**:
    - your-region: The AWS region where your SNS topic is located (e.g., us-east-1).
    - your-account-id: Your 12-digit AWS account ID.
    - StockPriceAlerts: The name of your SNS topic.

Step 7: Review and Create the Policy

- After pasting the JSON, click **Next: Tags** (you can skip adding tags).



IAM > Policies > AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846 > Edit policy

Step 1
Modify permissions in AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846

Step 2
Review and save

Modify permissions in AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846 Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

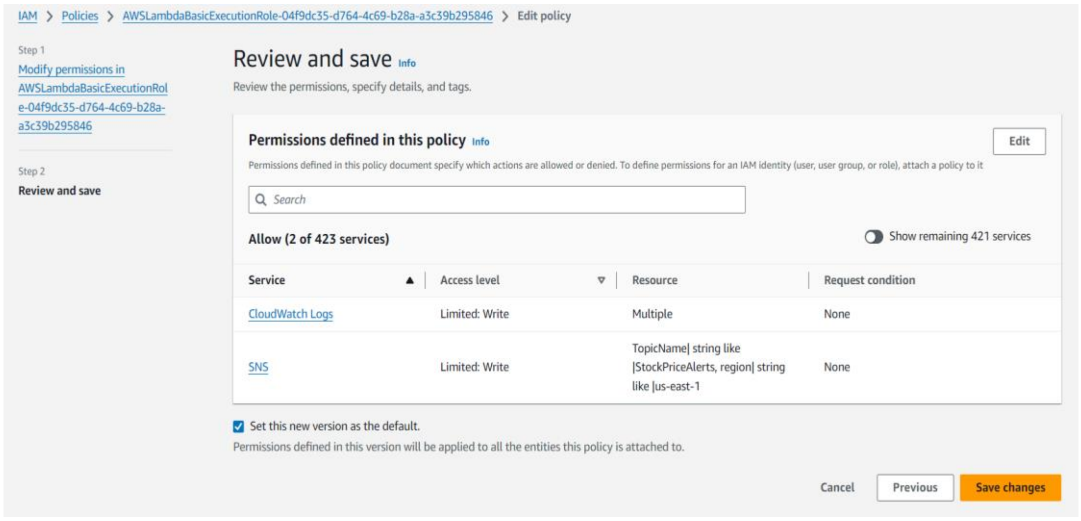
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-1:590184086939:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:590184086939:log-group:/aws/lambda/StockPriceAlertFunction:*"
            ]
        },
```

Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

- Click **Next: Review**.



IAM > Policies > AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846 > Edit policy

Step 1
Modify permissions in AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846

Step 2
Review and save

Review and save Info

Review the permissions, specify details, and tags.

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

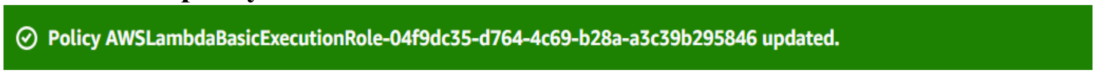
Allow (2 of 423 services)

Show remaining 421 services

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	Multiple	None
SNS	Limited: Write	TopicName| string like |StockPriceAlerts, region| string like |us-east-1	None

✓ Set this new version as the default.
Permissions defined in this version will be applied to all the entities this policy is attached to.

- Give the policy a name, such as LambdaSNSPublishPolicy.
- Click **Create policy**.

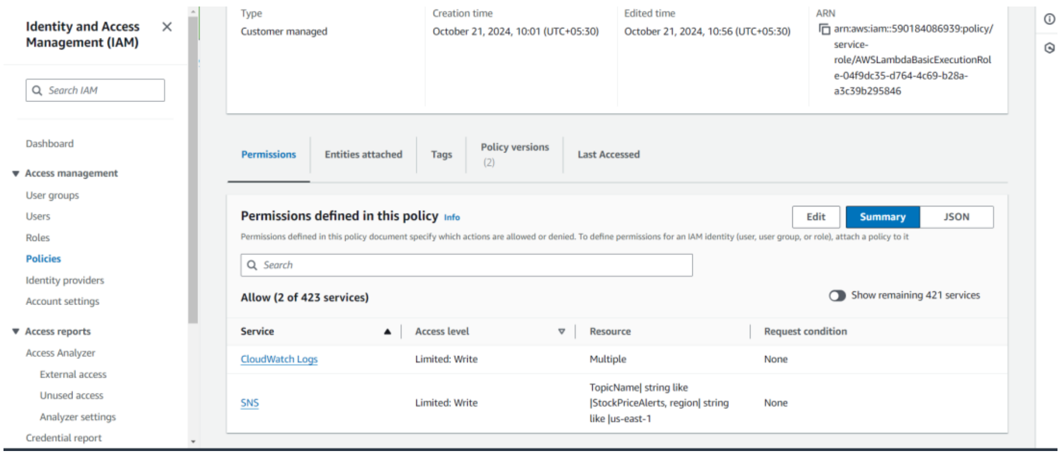


Policy AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846 updated.

Step 8: Attach the Policy to the Lambda Role

- After creating the policy, return to the **Roles** section in IAM.
- Select the Lambda execution role you identified earlier.
- Click on **Attach policies**.

- Search for the newly created policy (LambdaSNSPublishPolicy), select it, and click **Attach policy**.
  Step 9: Confirm Permissions
- Ensure the new policy appears in the list of permissions for the Lambda execution role.



Type: Customer managed
Creation time: October 21, 2024, 10:01 (UTC+05:30)
Edited time: October 21, 2024, 10:56 (UTC+05:30)
ARN: arn:aws:iam::590184086939:policy/service-role/AWSLambdaBasicExecutionRole-04f9dc35-d764-4c69-b28a-a3c39b295846

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (2 of 423 services)

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	Multiple	None
SNS	Limited: Write	TopicName| string like |StockPriceAlerts, region| string like |us-east-1	None

5. Schedule the Lambda Function with CloudWatch

Step 1: Open the CloudWatch Console

- **Log in** to the [AWS Management Console]() using your AWS account credentials.

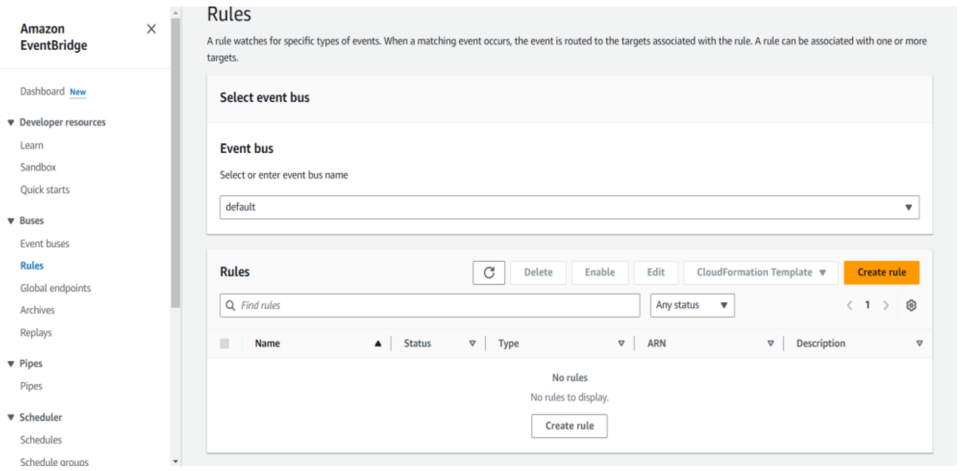
- In the **search bar** at the top, type **CloudWatch** and select **CloudWatch** from the dropdown list.

Step 2: Navigate to Rules

- In the CloudWatch Console, look at the left sidebar and click on **Rules** under the **Events** section.

Step 3: Create a New Rule

- Click the **Create rule** button at the top right of the Rules page



Rules

A rule watches for specific types of events. When a matching event occurs, the event is routed to the targets associated with the rule. A rule can be associated with one or more targets.

Select event bus

Event bus

Select or enter event bus name: default

Rules

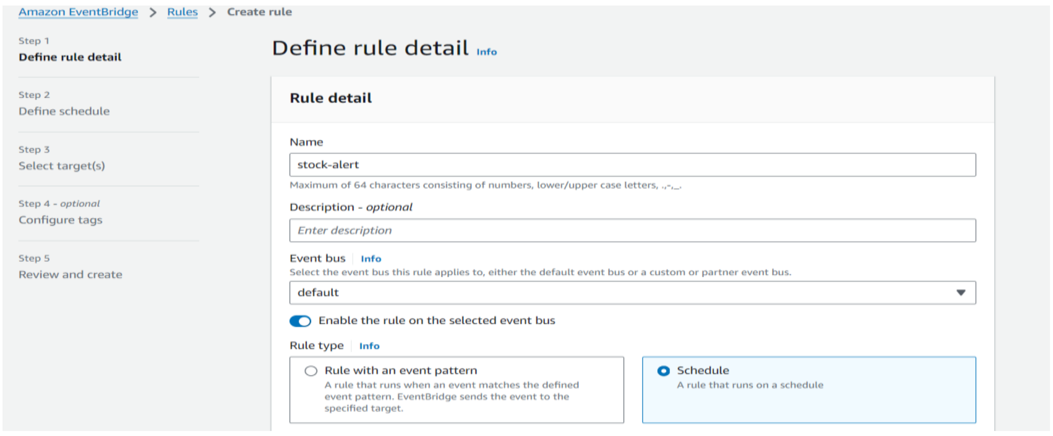
Name	Status	Type	ARN	Description

No rules

No rules to display.

Step 4: Configure the Event Source

- **Event Source**: Under the **Event Source** section, select **EventBridge (CloudWatch Events)**.
- **Rule Type**: Choose **Schedule**.



Amazon EventBridge > Rules > Create rule

Step 1
Define rule detail

Step 2
Define schedule

Step 3
Select target(s)

Step 4 - *optional*
Configure tags

Step 5
Review and create

Define rule detail Info

Rule detail

Name: stock-alert
Maximum of 64 characters consisting of numbers, lower/upper case letters, .,-,_.

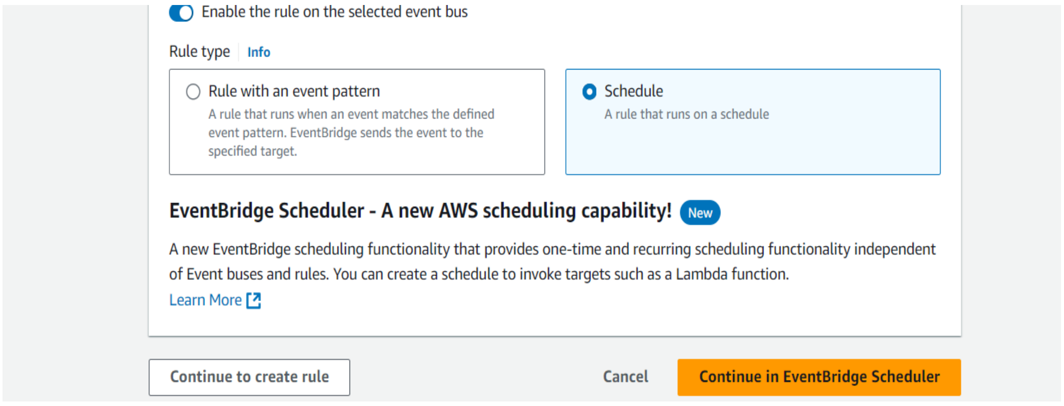
Description - *optional*: *Enter description*

Event bus Info
Select the event bus this rule applies to, either the default event bus or a custom or partner event bus.
default

Enable the rule on the selected event bus: on

Rule type Info
- Rule with an event pattern — A rule that runs when an event matches the defined event pattern. EventBridge sends the event to the specified target.
- [✓ Schedule] — A rule that runs on a schedule

- **Schedule Expression**: In the **Schedule expression** field, enter rate(15 minutes) to specify that the Lambda function should run every 15 minutes.

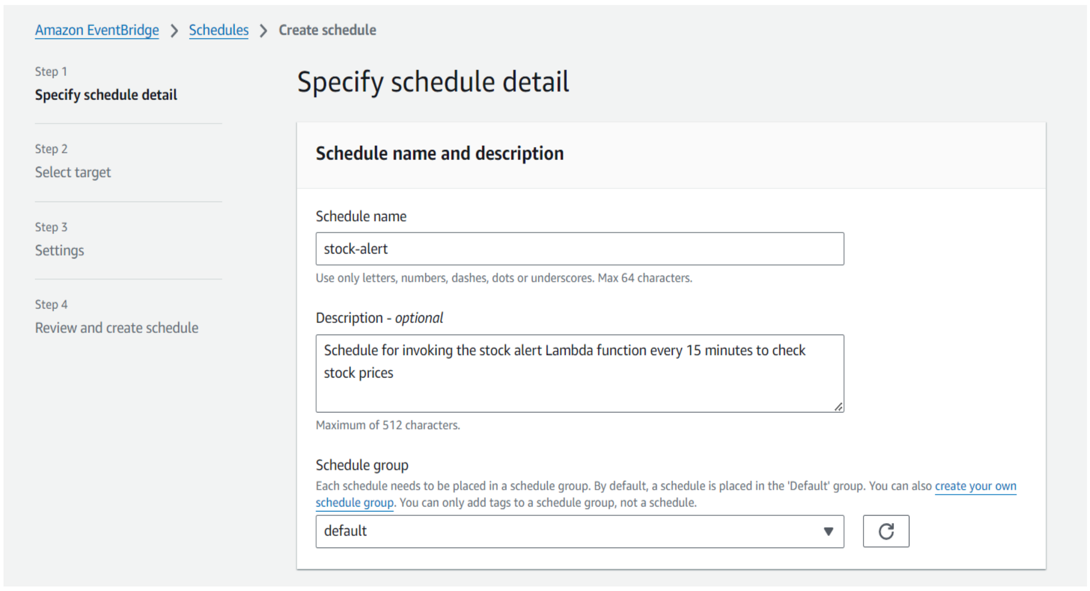


Enable the rule on the selected event bus: on

Rule type Info
- Rule with an event pattern — A rule that runs when an event matches the defined event pattern. EventBridge sends the event to the specified target.
- [✓ Schedule] — A rule that runs on a schedule

EventBridge Scheduler - A new AWS scheduling capability! New

A new EventBridge scheduling functionality that provides one-time and recurring scheduling functionality independent of Event buses and rules. You can create a schedule to invoke targets such as a Lambda function.
Learn More



Amazon EventBridge > Schedules > Create schedule

Step 1
Specify schedule detail

Step 2
Select target

Step 3
Settings

Step 4
Review and create schedule

Specify schedule detail

Schedule name and description

Schedule name: stock-alert
Use only letters, numbers, dashes, dots or underscores. Max 64 characters.

Description - *optional*: Schedule for invoking the stock alert Lambda function every 15 minutes to check stock prices
Maximum of 512 characters.

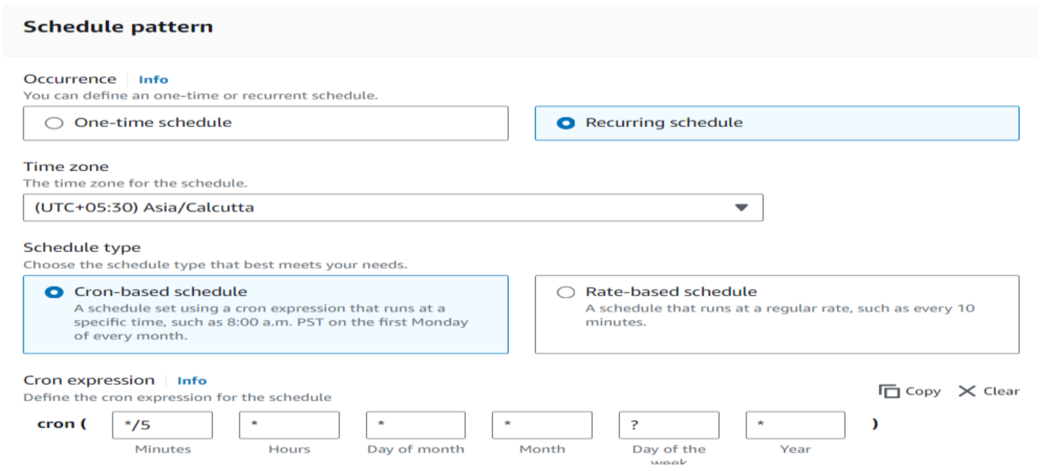
Schedule group
Each schedule needs to be placed in a schedule group. By default, a schedule is placed in the 'Default' group. You can also create your own schedule group. You can only add tags to a schedule group, not a schedule.
default

To clarify:

- ***/5**: Every 5 minutes

- *** (for hours, day of month, month, etc.): Every hour, day, month, etc.

- **?**: No specific day of the week (because you are using * for day of the month)



Schedule pattern

Occurrence Info
You can define an one-time or recurrent schedule.
One-time schedule [✓ Recurring schedule]

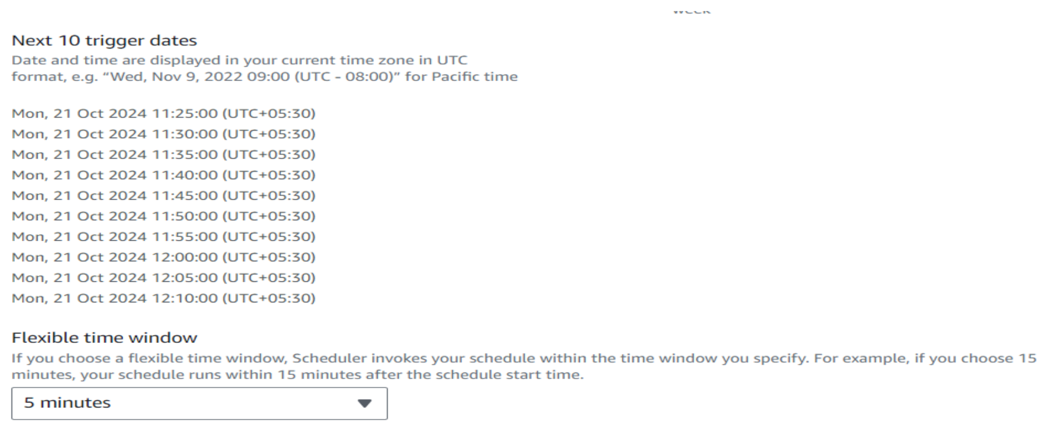
Time zone
The time zone for the schedule.
(UTC+05:30) Asia/Calcutta

Schedule type
Choose the schedule type that best meets your needs.
[✓ Cron-based schedule] A schedule set using a cron expression that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.
Rate-based schedule A schedule that runs at a regular rate, such as every 10 minutes.

Cron expression Info
Define the cron expression for the schedule

	Minutes	Hours	Day of month	Month	Day of the week	Year	
cron (	*/5	*	*	*	?	*	)

This will run the Lambda function every 5 minutes as scheduled. If you're fine with this, you can stick with the 5-minute schedule!



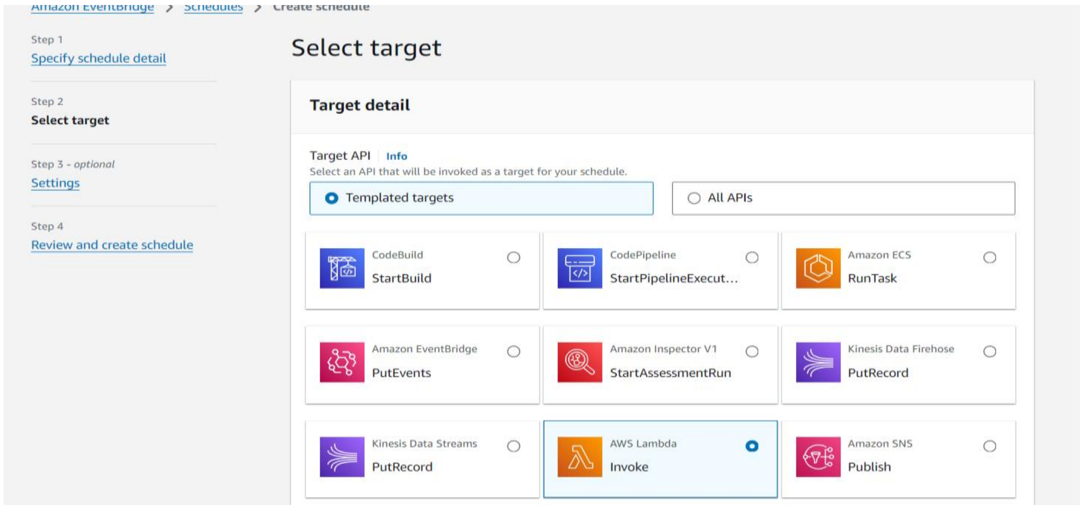
Next 10 trigger dates
Date and time are displayed in your current time zone in UTC format, e.g. "Wed, Nov 9, 2022 09:00 (UTC - 08:00)" for Pacific time

```
Mon, 21 Oct 2024 11:25:00 (UTC+05:30)
Mon, 21 Oct 2024 11:30:00 (UTC+05:30)
Mon, 21 Oct 2024 11:35:00 (UTC+05:30)
Mon, 21 Oct 2024 11:40:00 (UTC+05:30)
Mon, 21 Oct 2024 11:45:00 (UTC+05:30)
Mon, 21 Oct 2024 11:50:00 (UTC+05:30)
Mon, 21 Oct 2024 11:55:00 (UTC+05:30)
Mon, 21 Oct 2024 12:00:00 (UTC+05:30)
Mon, 21 Oct 2024 12:05:00 (UTC+05:30)
Mon, 21 Oct 2024 12:10:00 (UTC+05:30)
```

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

Step 5: Add the Lambda Function as the Target

- Scroll down to the **Targets** section.
- Click on **Add target**.
- In the **Target type** dropdown, select **Lambda function**.


Amazon EventBridge > Schedules > Create schedule

Step 1
Specify schedule detail

Step 2
Select target

Step 3 - *optional*
Settings

Step 4
Review and create schedule

Select target

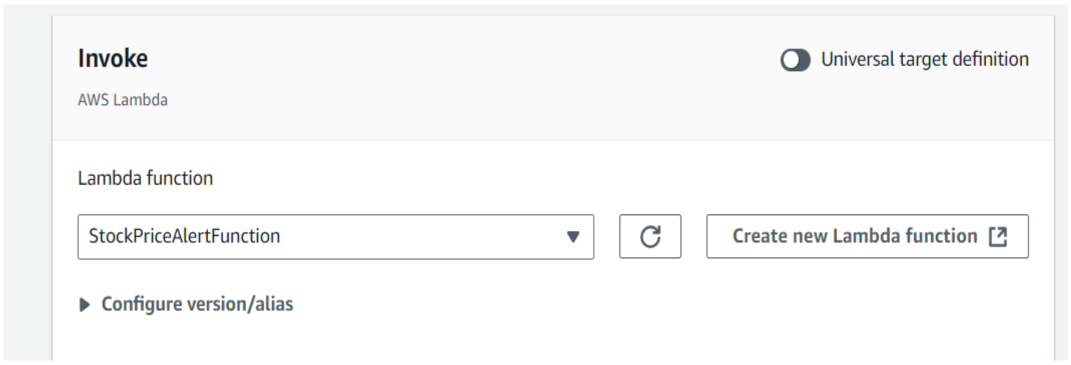
Target detail

Target API Info
Select an API that will be invoked as a target for your schedule.

[✓ Templated targets] All APIs

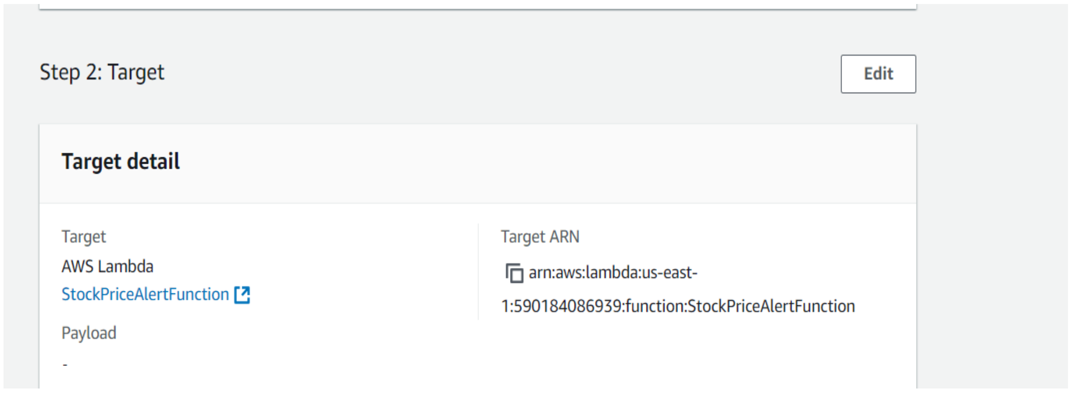
CodeBuild StartBuild	CodePipeline StartPipelineExecut...	Amazon ECS RunTask
Amazon EventBridge PutEvents	Amazon Inspector V1 StartAssessmentRun	Kinesis Data Firehose PutRecord
Kinesis Data Streams PutRecord	[✓ AWS Lambda Invoke]	Amazon SNS Publish

- In the **Function** dropdown, choose your Lambda function (e.g., StockPriceAlertFunction).


Invoke
AWS Lambda

Universal target definition

Lambda function
StockPriceAlertFunction

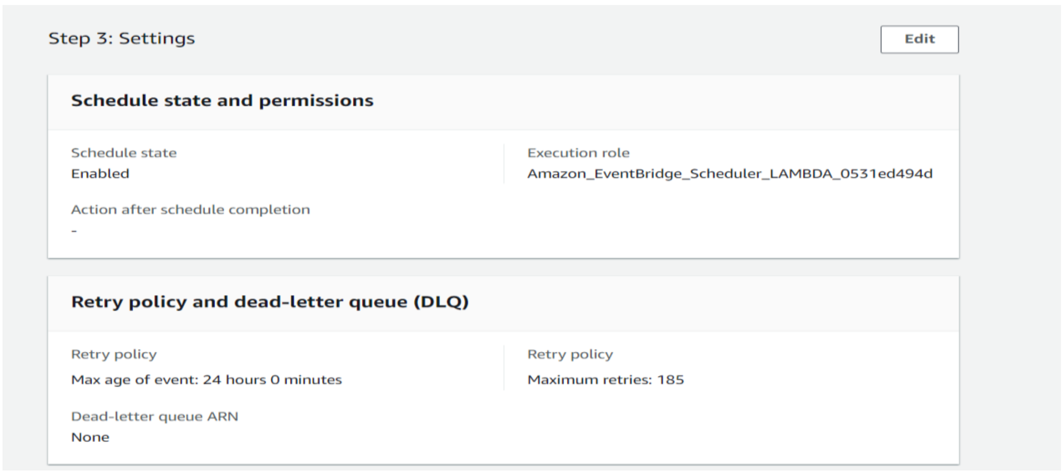

Step 2: Target

Target detail

Target
AWS Lambda
StockPriceAlertFunction

Payload
-

Target ARN
arn:aws:lambda:us-east-1:590184086939:function:StockPriceAlertFunction


Step 3: Settings

Schedule state and permissions

Schedule state
Enabled

Action after schedule completion
-

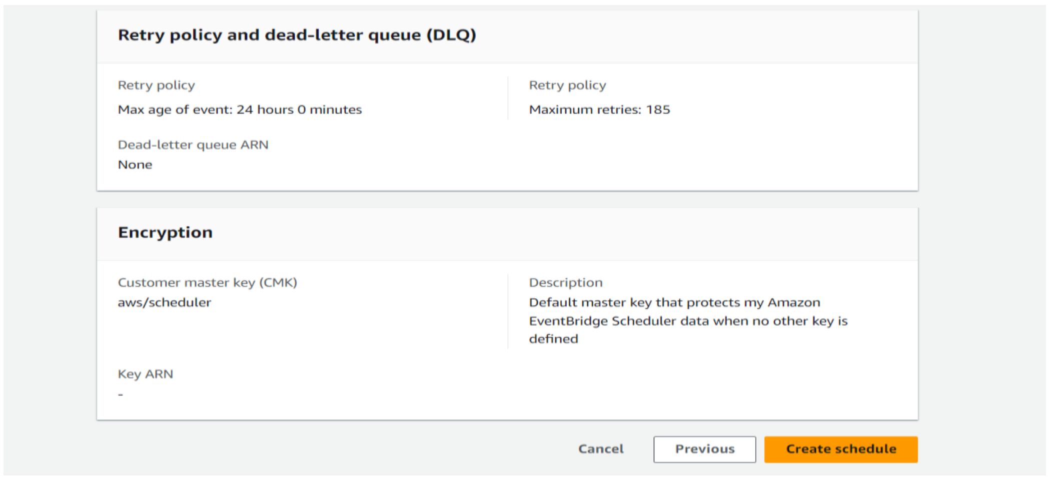
Execution role
Amazon_EventBridge_Scheduler_LAMBDA_0531ed494d

Retry policy and dead-letter queue (DLQ)

Retry policy
Max age of event: 24 hours 0 minutes

Dead-letter queue ARN
None

Retry policy
Maximum retries: 185



Retry policy and dead-letter queue (DLQ)

Retry policy
Max age of event: 24 hours 0 minutes

Retry policy
Maximum retries: 185

Dead-letter queue ARN
None

Encryption

Customer master key (CMK)
aws/scheduler

Description
Default master key that protects my Amazon EventBridge Scheduler data when no other key is defined

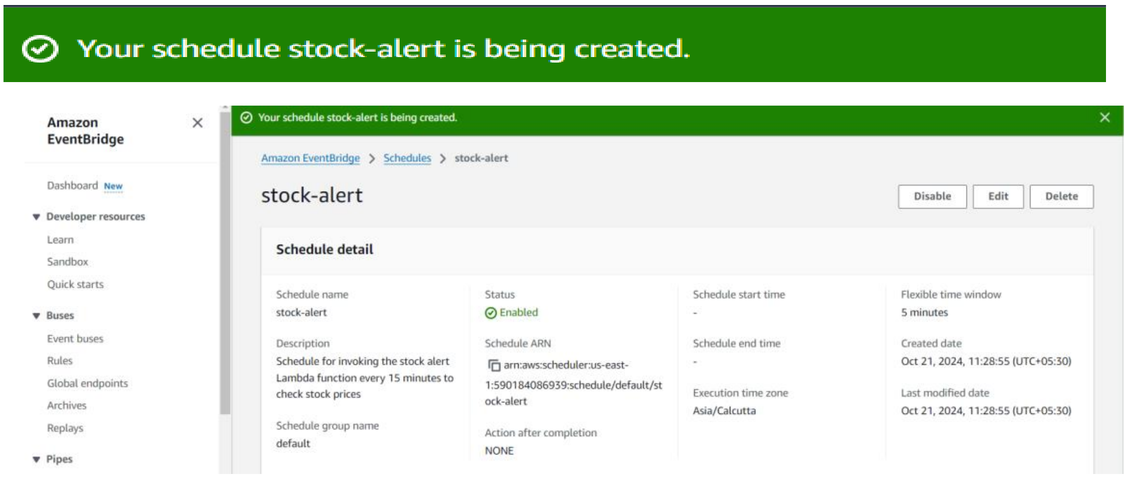
Key ARN
-

Step 6: Name and Create the Rule

- Scroll to the **Configure details** section.
- Enter a **Name** for your rule (e.g., StockPriceAlertSchedule).
- Ensure that the **State** is set to **Enabled**.
- Click the **Create rule** button at the bottom.

Step 7: Confirm the Rule

- Once created, your new rule should appear in the list of rules. Confirm that it's listed and enabled.



Your schedule stock-alert is being created.

stock-alert

Schedule detail

Schedule name
stock-alert

Description
Schedule for invoking the stock alert Lambda function every 15 minutes to check stock prices

Schedule group name
default

Status
Enabled

Schedule ARN
arn:aws:scheduler:us-east-1:590184086939:schedule/default/stock-alert

Action after completion
NONE

Schedule start time
-

Schedule end time
-

Execution time zone
Asia/Calcutta

Flexible time window
5 minutes

Created date
Oct 21, 2024, 11:28:55 (UTC+05:30)

Last modified date
Oct 21, 2024, 11:28:55 (UTC+05:30)

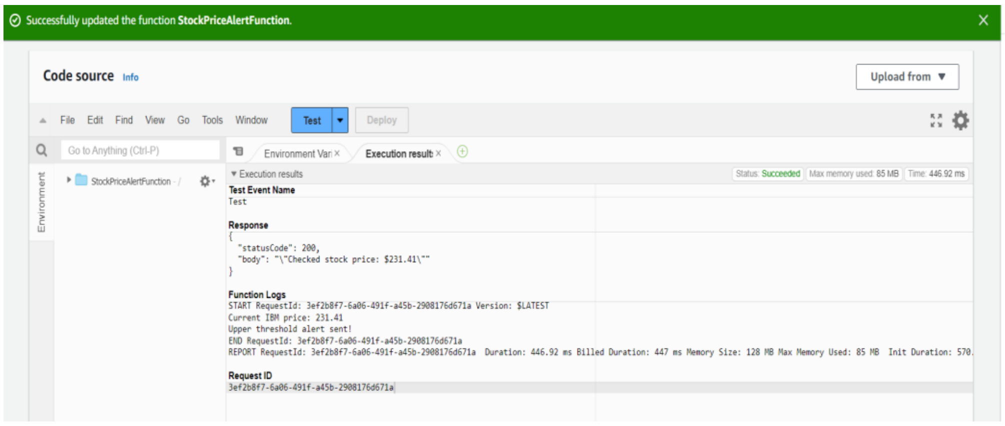
6.Test Your Lambda Function

Once the function is updated, you can test it by clicking the Test button in the Lambda console.

Manual Test

- You can manually test your Lambda function by clicking the **Test** button in the Lambda console.
  - Navigate to your Lambda function in the AWS Management Console.

- Click on the **Test** button.
- Create a test event if you haven't already, or select an existing test event.
- Click **Test** to execute the function and observe the output.



Successfully updated the function **StockPriceAlertFunction**.

Execution results — Status: Succeeded | Max memory used: 85 MB | Time: 446.92 ms

Test Event Name
```
Test
```

Response
```
{
  "statusCode": 200,
  "body": "\"Checked stock price: $231.41\""
}
```

Function Logs
```
START RequestId: 3ef2b8f7-6a06-491f-a45b-2908176d671a Version: $LATEST
Current IBM price: 231.41
Upper threshold alert sent!
END RequestId: 3ef2b8f7-6a06-491f-a45b-2908176d671a
REPORT RequestId: 3ef2b8f7-6a06-491f-a45b-2908176d671a  Duration: 446.92 ms Billed Duration: 447 ms Memory Size: 128 MB Max Memory Used: 85 MB  Init Duration: 570.
```

Request ID
```
3ef2b8f7-6a06-491f-a45b-2908176d671a
```

Wait for Trigger

- If your Lambda function is set up with a scheduled trigger (e.g., using Amazon CloudWatch Events or EventBridge), wait for the scheduled time to see it execute automatically.
  - Ensure that your rule is enabled to trigger the Lambda function at the specified interval (e.g., every 15 minutes).
  - Check the time and ensure it aligns with the scheduled interval.

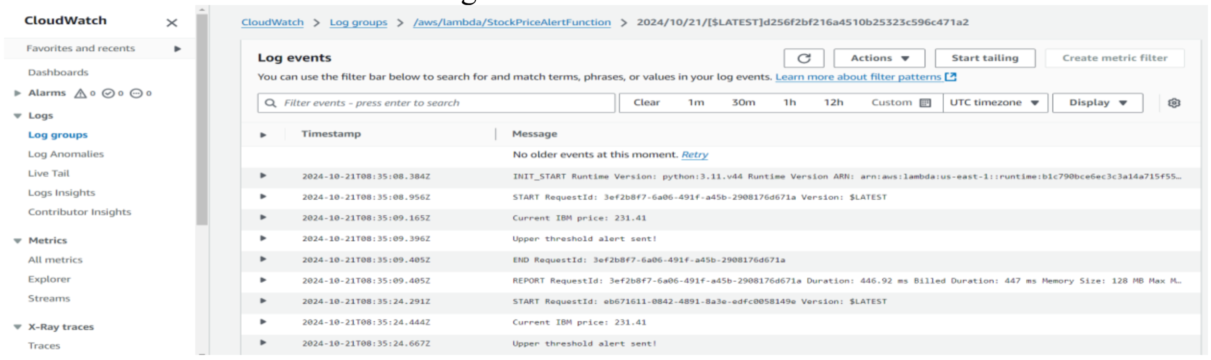
Monitor Logs

- You can view detailed logs of each Lambda execution in **CloudWatch Logs**.
  - Go to the **CloudWatch Console**.
  - In the left sidebar, click on **Logs** and then **Log Groups**.
  - Look for a log group named /aws/lambda/<YourFunctionName>, where <YourFunctionName> is the name of your Lambda function.
  - Click on the log group to view the log streams for each invocation of your Lambda function.
  - Select a log stream to see detailed information about the execution, including any errors or output generated by the function.

In CloudWatch you will get the monitoring details as

After 5 minutes:

You will get the notification to the email.

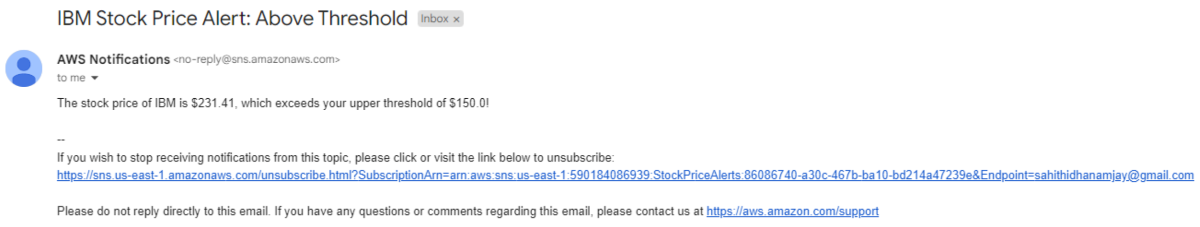


CloudWatch > Log groups > /aws/lambda/StockPriceAlertFunction > 2024/10/21/[$LATEST]d256f2bf216a4510b25323c596c471a2

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

```
Timestamp                   Message
                            No older events at this moment. Retry
2024-10-21T08:35:08.384Z    INIT_START Runtime Version: python:3.11.v44 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:b1c790bce6ec3c3a14a715f55…
2024-10-21T08:35:08.956Z    START RequestId: 3ef2b8f7-6a06-491f-a45b-2908176d671a Version: $LATEST
2024-10-21T08:35:09.165Z    Current IBM price: 231.41
2024-10-21T08:35:09.396Z    Upper threshold alert sent!
2024-10-21T08:35:09.405Z    END RequestId: 3ef2b8f7-6a06-491f-a45b-2908176d671a
2024-10-21T08:35:09.405Z    REPORT RequestId: 3ef2b8f7-6a06-491f-a45b-2908176d671a Duration: 446.92 ms Billed Duration: 447 ms Memory Size: 128 MB Max M…
2024-10-21T08:35:24.291Z    START RequestId: eb671611-0842-4891-8a3e-edfc0058149e Version: $LATEST
2024-10-21T08:35:24.444Z    Current IBM price: 231.41
2024-10-21T08:35:24.667Z    Upper threshold alert sent!
```



IBM Stock Price Alert: Above Threshold Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com>
to me

The stock price of IBM is $231.41, which exceeds your upper threshold of $150.0!

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:590184086939:StockPriceAlerts:86086740-a30c-467b-ba10-bd214a47239e&Endpoint=sahithidhanamjay@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Conclusion:

The stock price alert system built on AWS successfully addresses the need for real-time notifications, allowing users to stay informed about market movements. By leveraging various AWS services, the system is designed for scalability, reliability, and efficient user management. Users benefit from customizable alerts, enabling them to make informed trading decisions quickly.