

1. Designing an Automatic Data Collection and Storage System with AWS Lambda and Slack Integration for Server Availability Monitoring and Slack Notification

Overview

Goal:

To create an AWS Lambda function that will periodically fetch data from an API and store it in an Amazon RDS instance. The function should be triggered by an Amazon CloudWatch Event that occurs every 15 seconds.

Technologies used:

AWS Lambda, Amazon DynamoDB, Amazon CloudWatch, Amazon EventBridge, Slack API

Steps:

1. Set up an AWS Lambda function: Create an AWS Lambda function that will periodically check the availability of servers.
2. Configure Lambda to store data in Amazon DynamoDB.
3. Configure a CloudWatch event trigger to schedule the Lambda function to run periodically.
4. Integrating a Slack bot and incoming webhook for your Slack workspace.
5. Modify Lambda function to send data to Slack: Modify the Lambda function to send a notification to Slack when a server is unavailable. You can use the Slack API to send a message to your Slack workspace via the incoming webhook.
6. Test and Deploy: Test the Lambda function and Slack integration to ensure that server availability data is being collected and stored in S3, and notifications are being sent to Slack when a server is unavailable.
7. Monitor and Maintain: Monitor the system to ensure it is running smoothly and make any necessary adjustments as needed

Codes Used:

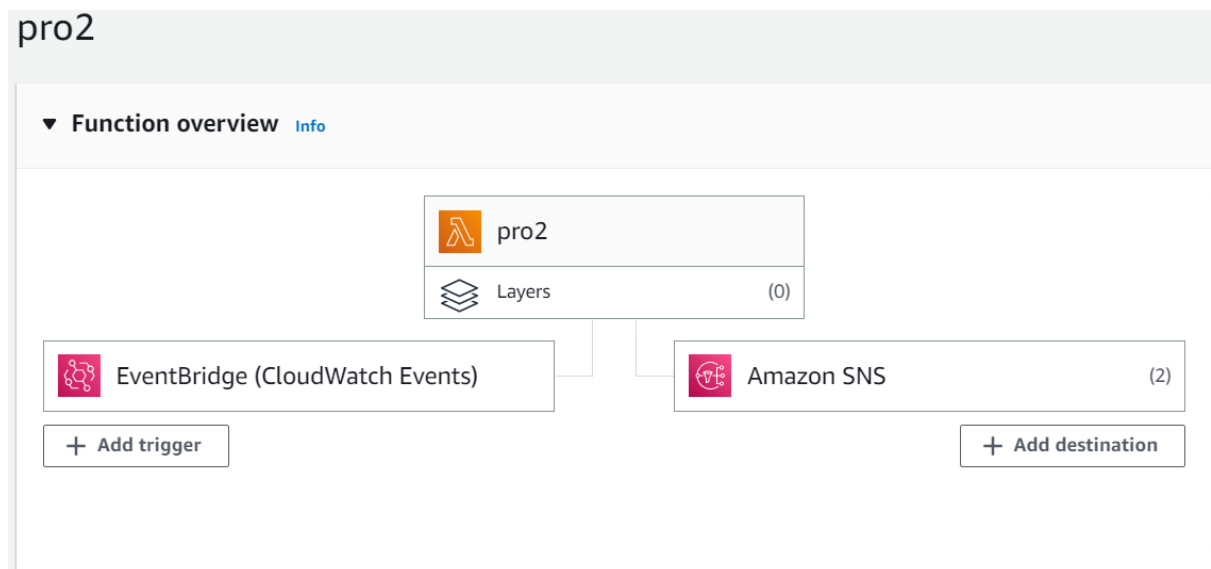
Using the lambda function in AWS we can deploy the required programme to fetch the API details for every 1 min.

While creating the Lambda function, we have to create a database to store the data fetched from the API. The database we used here was Amazon DynamoDB.

```
lambda_function × Execution results × (+)
1 import json
2 import requests
3 import boto3
4 from datetime import datetime
5
6 def put_doc(dict1,table):
7     dynamodb_response = table.put_item(
8         Item = {
9             'timestamp': str(dict1['timestamp']),
10            'latitude':dict1['iss_position']['latitude'],
11            'longitude':dict1['iss_position']['longitude'],
12            'message':dict1['message'],
13            'date_time':datetime.now().strftime("%Y-%m-%d, %H:%M:%S")
14        })
15
16
17 def post_slack_notification(slack_message):
18     # Define the Slack webhook URL
19     slack_webhook_url = "https://hooks.slack.com/services/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"
20
21     # Send a POST request to the Slack webhook URL
22     response = requests.post(slack_webhook_url, json=slack_message)
23
24
25 def lambda_handler(event, context):
26
27     dynamodb = boto3.resource('dynamodb')
28     table = dynamodb.Table('project2')
29
30     url = "http://api.open-notify.org/iss-now.json"
31     response = requests.get(url)
32
33
34     if response.status_code != 200:
35         slack_message = {
36             "text": "ALERT: Server {} is not available. Status code:{}".format(url,response.status_code)
37         }
38         if response.status_code != 200:
39             slack_message = {
40                 "text": "ALERT: Server {} is not available. Status code:{}".format(url,response.status_code)
41             }
42             post_slack_notification(slack_message)
43
44     else:
45         dict1 = json.loads(response.content)
46         put_doc(dict1,table)
47
48     print(response.content)
49     print(datetime.now().strftime("%Y-%m-%d, %H:%M:%S"))
50     return {
51         'statusCode': 200,
52         'body': json.dumps('Hello from Lambda!')}
53 }
```

The above code was used to fetch the details from API and to integrate Slack chatbot with the request.

Schema of Lambda function, trigger and destination



Trigger:

Triggers (1) [Info](#)

Find triggers

☐ Trigger

EventBridge (CloudWatch Events): pro2
arn:aws:events:us-east-1:026270796322:rule/pro2
Rule state: **ENABLED**

▼ Details

Description: **automated trigger for min**

Event bus: **default**

name: **pro2**

Schedule expression: **rate(1 minute)**

Service principal: **events.amazonaws.com**

Statement ID: **AWSEvents_pro2_Id231883036199**

url: **events/home#/rules/pro2**

Here, EventBridge service is utilised to setup the trigger. Inside this service we can find the Rules section in which we specify the event and cron job duration for automatic data collection and storage. We can enable the rule for starting the data collection and disable it accordingly.

Amazon EventBridge > Rules > pro2

pro2

Edit Enable Delete CloudFormation Template ▼

Rule details Info

| | | | |
|--|--|---|------------------|
| Rule name pro2 | Status ⊗ Disabled | Event bus name default | Type Standard |
| Description automated trigger for min | Rule ARN arn:aws:events:us-east-1:02627079:6322:rule/pro2 | Event bus ARN arn:aws:events:us-east-1:02627079:6322:event-bus/default | |

Event schedule Targets Monitoring Tags

Event schedule Info Edit

Fixed rate of
1 minute

We can use Amazon EventBridge Scheduler function to schedule the same instead of using the Rules option.

Schedules (1) Refresh Disable Edit Delete Create schedule

Search loaded schedules All states All groups < 1 > ⚙

| <input type="checkbox"/> | Schedule name ▼ | Schedule group ▼ | Status ▲ | Target ▼ | Target type ▼ | Last modified ▼ |
|--------------------------|-----------------|------------------|-----------|----------|---------------|------------------------------------|
| <input type="checkbox"/> | pro2_schedule | default | ✔ Enabled | pro2 | LAMBDA_Invoke | Mar 21, 2023, 14:00:44 (UTC+05:30) |

This too will fetch the details from the API and store it inside the database.

Destination:

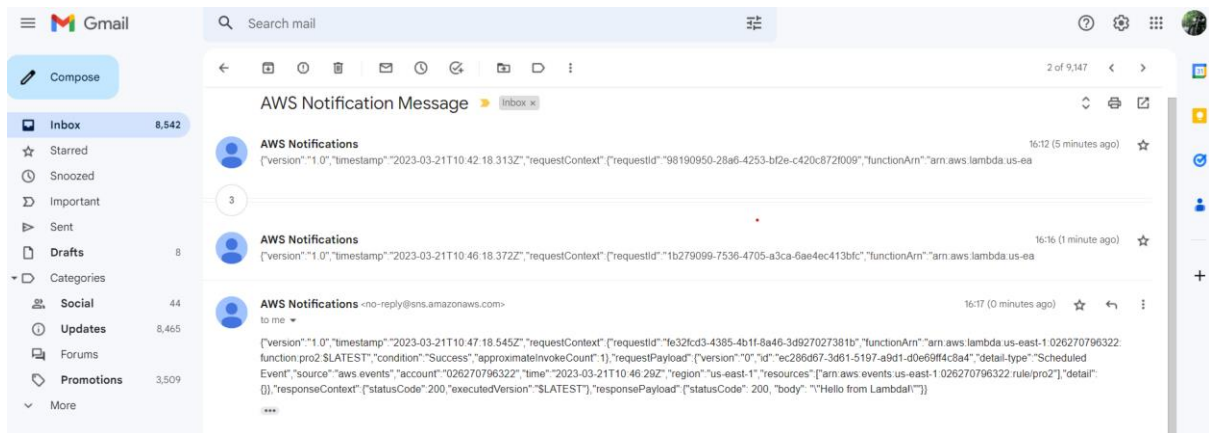
Destinations Info Remove Edit Add destination

Find destinations

| | Source | Stream | Condition | Destination |
|-----------------------|-------------------------|--------|------------|--|
| <input type="radio"/> | Asynchronous invocation | - | On success | arn:aws:sns:us-east-1:026270796322:new_alarm |
| <input type="radio"/> | Asynchronous invocation | - | On failure | arn:aws:sns:us-east-1:026270796322:new_alarm |

Here we can specify the destination (email) via Amazon SNS. Error notification will only be issued if the API provided throws back any error. For testing purpose, we created a success condition for notifying the data collection via the given email.

Hence, if the data collection is carried out according to the given lambda function, we will receive a notification message from AWS.



Data collected:

The following data are fetched automatically and stored inside DynamoDB upon enabling the scheduler or Rules in EventBridge for every minute.

Items returned (47)

| <input type="checkbox"/> | timestamp | ▼ | date_time | ▼ | latitude | ▼ | longitude | ▼ | message |
|--------------------------|------------|---|--------------|---|----------|---|-----------|---|---------|
| <input type="checkbox"/> | 1679394818 | | 2023-03-2... | | 16.9812 | | 80.1527 | | success |
| <input type="checkbox"/> | 1679394737 | | 2023-03-2... | | 12.9653 | | 77.0066 | | success |
| <input type="checkbox"/> | 1679394119 | | 2023-03-2... | | -18.0355 | | 54.1737 | | success |
| <input type="checkbox"/> | 1679393999 | | 2023-03-2... | | -23.7985 | | 49.1812 | | success |
| <input type="checkbox"/> | 1679393778 | | 2023-03-2... | | -33.8498 | | 38.4788 | | success |
| <input type="checkbox"/> | 1679394018 | | 2023-03-2... | | -22.9423 | | 49.9613 | | success |
| <input type="checkbox"/> | 1679393838 | | 2023-03-2... | | -31.2310 | | 41.6095 | | success |
| <input type="checkbox"/> | 1679395577 | | 2023-03-2... | | 47.8320 | | 123.9172 | | success |
| <input type="checkbox"/> | 1679395697 | | 2023-03-2... | | 50.2577 | | 134.7043 | | success |
| <input type="checkbox"/> | 1679394435 | | 2023-03-2... | | -2.2770 | | 65.9769 | | success |

The following are attributes of a single data “1679394818” collected

DynamoDB > Items: project2 > Edit item

Edit item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Form JSON view

| Attribute name | Value | Type | |
|---------------------------|----------------------|--------|--------|
| timestamp - Partition key | 1679394818 | String | |
| date_time | 2023-03-21, 10:33:38 | String | Remove |
| latitude | 16.9812 | String | Remove |
| longitude | 80.1527 | String | Remove |
| message | success | String | Remove |

Attributes View DynamoDB JSON

```
1 {
2   "timestamp": {
3     "s": "1679394818"
4   },
5   "date_time": {
6     "s": "2023-03-21, 10:33:38"
7   },
8   "latitude": {
9     "s": "16.9812"
10  },
11  "longitude": {
12    "s": "80.1527"
13  },
14  "message": {
15    "s": "success"
16  }
17 }
```

Slack integration:

For integrating slack, the following steps were followed

1. Create a new slack App
2. Add basic functionality and install the App
3. Copy the webhook URL and integrate it with the lambda function
4. Your setup is done and the slack Api will launch your project with specific channel for automated message receipt

5. For this project we have setup the slack Api as to receive error messages if the given URL is not accessible. For testing we used a 403 forbidden error fetching URL <https://www.sec.gov/Archives/edgar/dailyindex/xbrl/companyfacts.zip>



Q Search

- Start learning
- Authentication
- Messaging
- Metadata
- Surfaces
- Block Kit
- Interactivity
- APIs
- Workflows
- Enterprise
- Apps for Admins
- Gov Slack

Your Apps

Create New App

Filter apps by name or workspace

| App Name | Workspace | Distribution Status |
|----------|-----------|---------------------|
| mypro2 | project_2 | Not distributed |

Your App Configuration Tokens

Generate Token

[Learn about tokens](#)

Building Apps for Slack

Create an app that's just for your workspace (or build one that can be used by any workspace) by following the steps below.

Add features and functionality



Choose and configure the tools you'll need to create your app (or review all [our documentation](#)).

Building an internal app locally or behind a firewall?

To receive your app's payloads over a WebSockets connection, enable [Socket Mode](#) for your app.

Incoming Webhooks

Post messages from external sources into Slack.

Interactive Components

Add components like buttons and select menus to your app's interface, and create an interactive experience for users.

Slash Commands

Allow users to perform app actions by typing commands in Slack.

Event Subscriptions

Make it easy for your app to respond to activity in Slack.

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your `message` in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}'  
https://hooks.slack.com/services/T04TE1M5A77/B04TGRTQY11/6D9h6yJkgqxn6Y1mTa0VAj9  
L
```

Copy

| Webhook URL | Channel | Added By |
|--|-----------|---------------------------------------|
| <div>https://hooks.slack.com/services/</div> <div>Copy</div> | #project2 | harikrishnancherukara Mar 11, 2023 |

project_2

Slack Connect

Browse Slack

Channels

general

project2

random

Add channels

Direct messages

harikrishnancherukara you

Add coworkers

Apps

mypro2

Add apps

project2

This is the very beginning of the # project2 channel

This channel is for everything #project2. Hold meetings, share docs, and make decisions together with your team. [Edit description](#)

[Add people](#)

Saturday, March 11th

harikrishnancherukara 6:10 PM

joined #project2.

harikrishnancherukara 6:18 PM

added an integration to this channel: [mypro2](#)

mypro2 APP 6:30 PM

ALERT: Server is not available. Status code:

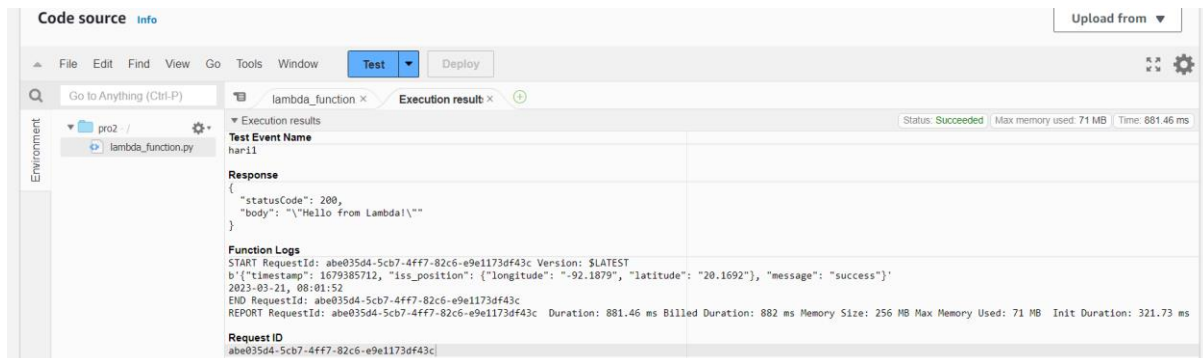
ALERT: Server <http://api.open-notify.org/iss-now.json> is not available. Status code:200

ALERT: Server <https://www.sec.gov/Archives/edgar/daily-index/xbrl/companyfacts.zip> is not available. Status code:403

[illegible]

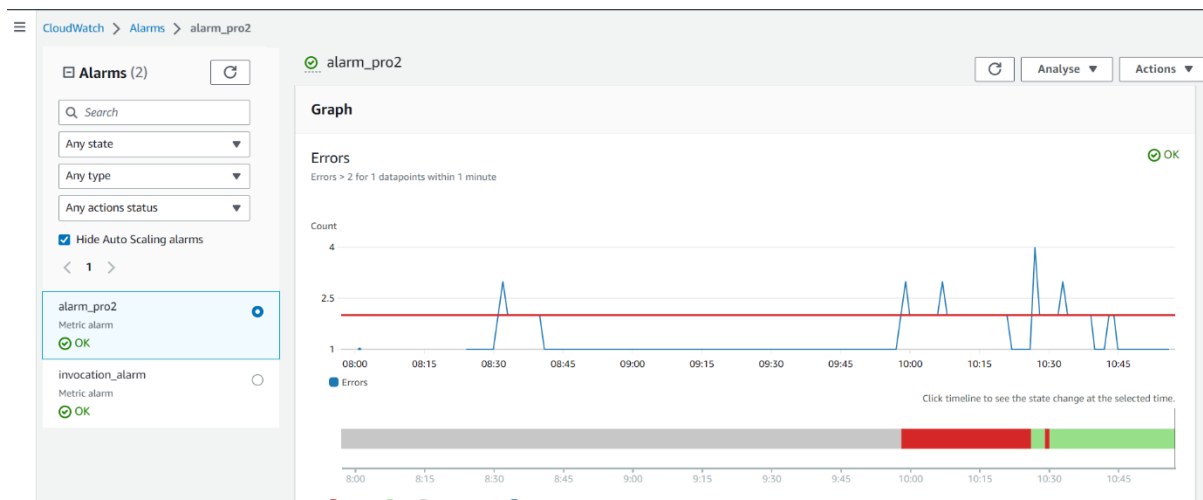
Whenever the request doesn't parse through, an alert will be automatically posted in the slack chat box as shown above. The test URL above mentioned was used to verify the recurring automated messages

Result:



The lambda function was executed without any errors and deployed which helped to fetch the data from the given Api and load it into Amazon DynamoDB database.

The CloudWatch logs for the same is given below,



For testing purpose, two alarms were created for error and invocation basing the lambda function. In this case, the alarm is monitoring the "errors" metric and has been configured to trigger when the metric exceeds a threshold of 2 for 1 datapoint within a 1-minute period.

This means that if there are more than 2 errors within a 1-minute period, the alarm will be triggered and a notification will be sent to the slack Api chat box.

Since it is highly unlikely for the given Api link to fail another alarm for invocation was set. If there are more than 2 invocations of the monitored service or function within a 1-minute period, the alarm will be triggered and a notification dispatched to the given SNS id.