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.

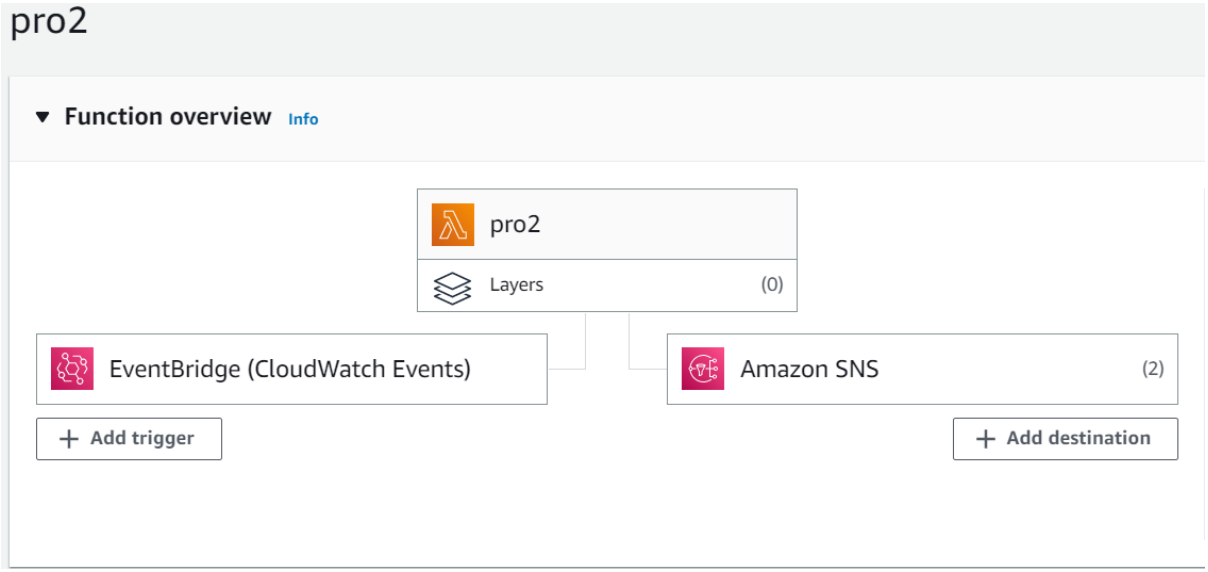
While creating the Lambda function, we have to create a database to store the data fetched from the API. The database we used here was Amazon DynamoDB.

lambda_function × Execution results ×

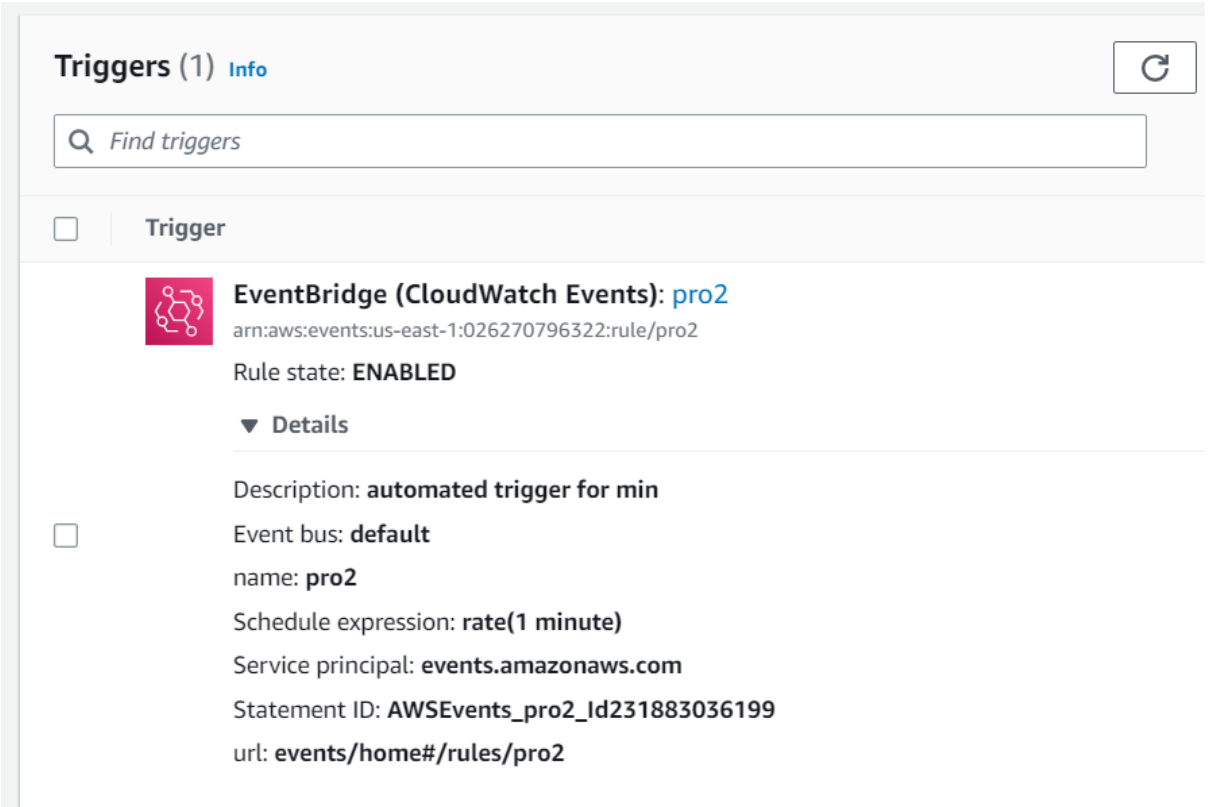
```python
import json
import requests
import boto3
from datetime import datetime

def put_doc(dict1,table):
    dynamodb_response = table.put_item(
        Item = {
            'timestamp': str(dict1['timestamp']),
            'latitude':dict1['iss_position']['latitude'],
            'longitude':dict1['iss_position']['longitude'],
            'message':dict1['message'],
            'date_time':datetime.now().strftime("%Y-%m-%d, %H:%M:%S")
        })


def post_slack_notification(slack_message):
    # Define the Slack webhook URL
    slack_webhook_url = "https://hooks.slack.com/services/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"

    # Send a POST request to the Slack webhook URL
    response = requests.post(slack_webhook_url, json=slack_message)


def lambda_handler(event, context):

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('project2')

    url = "http://api.open-notify.org/iss-now.json"
    response = requests.get(url)


    if response.status_code != 200:
        slack_message = {
            "text": "ALERT: Server {} is not available. Status code:{}".format(url,response.status_code)
     if response.status_code != 200:
         slack_message = {
             "text": "ALERT: Server {} is not available. Status code:{}".format(url,response.status_code)
         }
         post_slack_notification(slack_message)

     else:
         dict1 = json.loads(response.content)
         put_doc(dict1,table)

     print(response.content)
     print(datetime.now().strftime("%Y-%m-%d, %H:%M:%S"))
     return {
         'statusCode': 200,
         'body': json.dumps('Hello from Lambda!')
     }
```

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

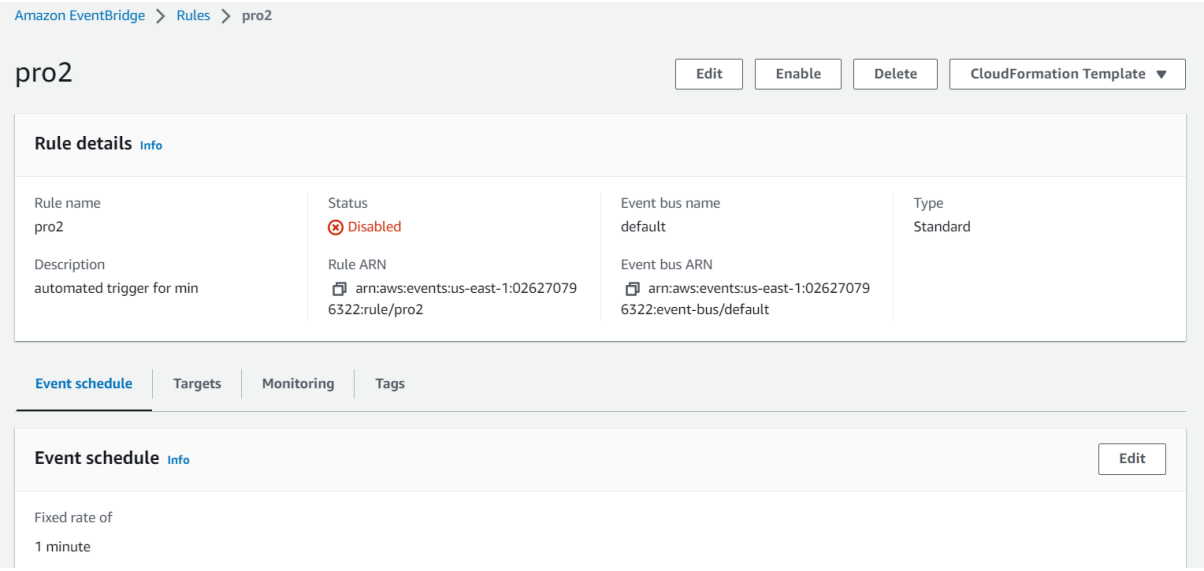
Schema of Lambda function, trigger and destination



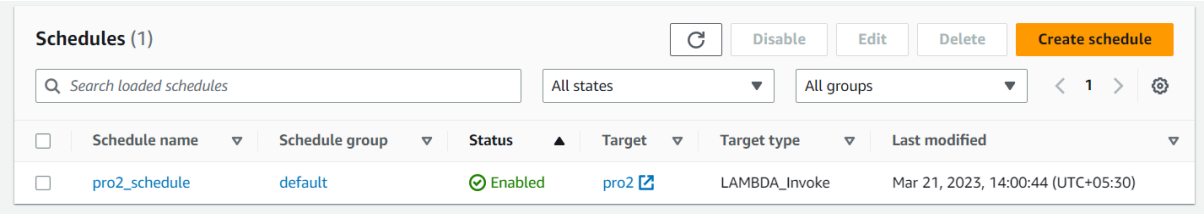
Trigger:



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.

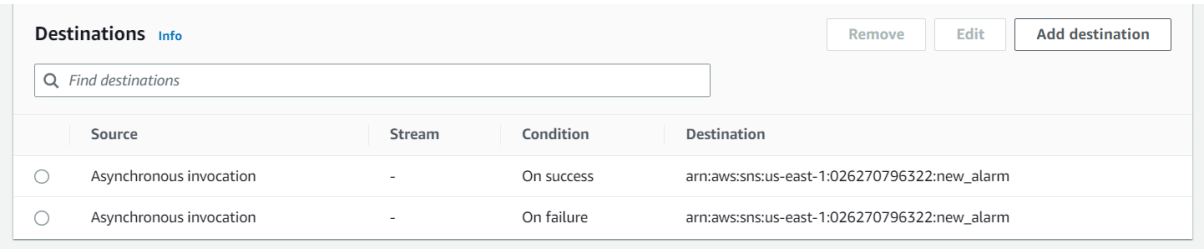


We can use Amazon EventBridge Scheduler function to schedule the same instead of using the Rules option.



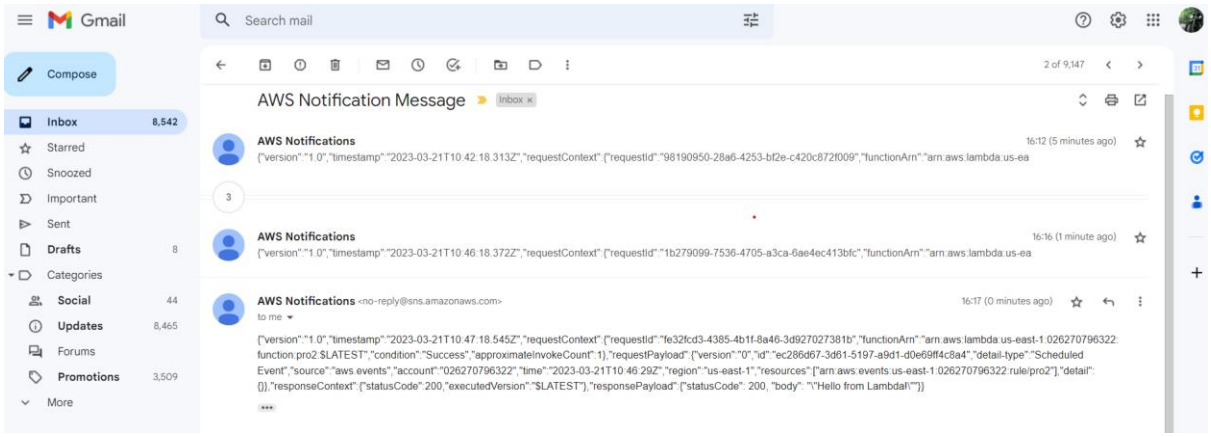
This too will fetch the details from the API and store it inside the database.

Destination:



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.



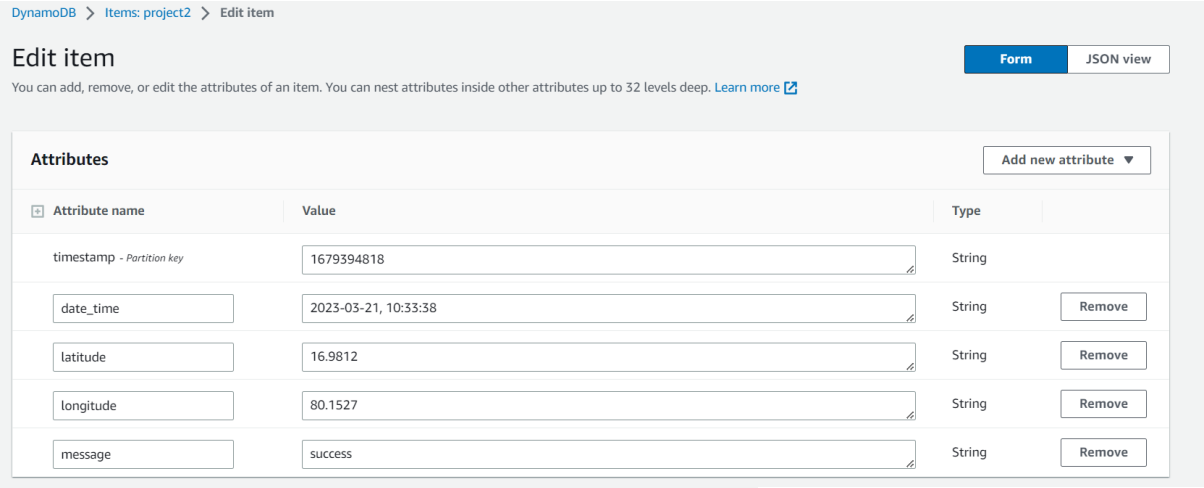
Data collected:

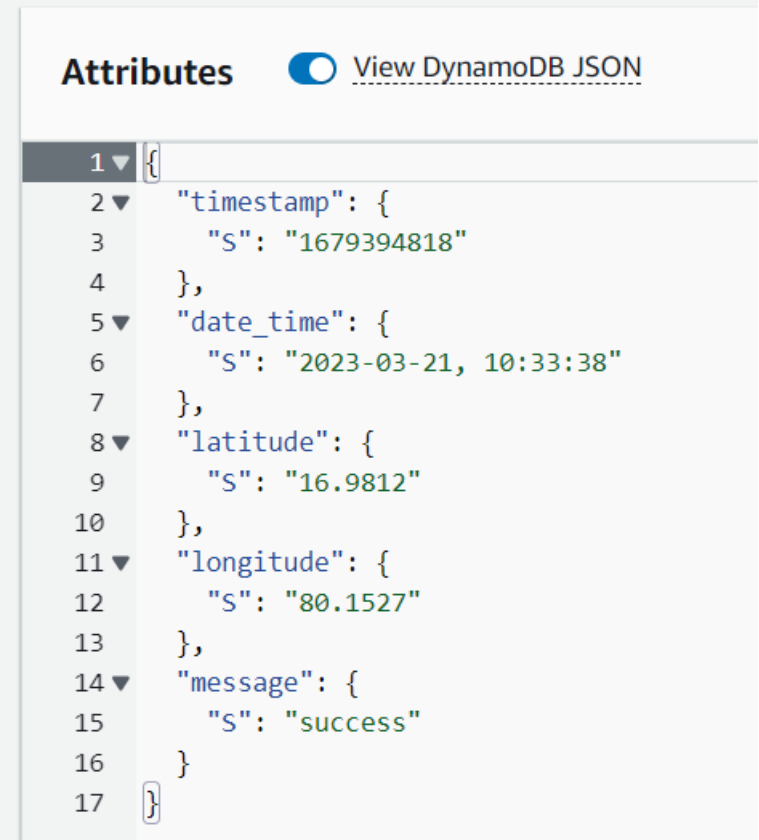
The following data are fetched automatically and stored inside DynamoDB upon enabling the scheduler or Rules in EventBridge for every minute.

Items returned (47)

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

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



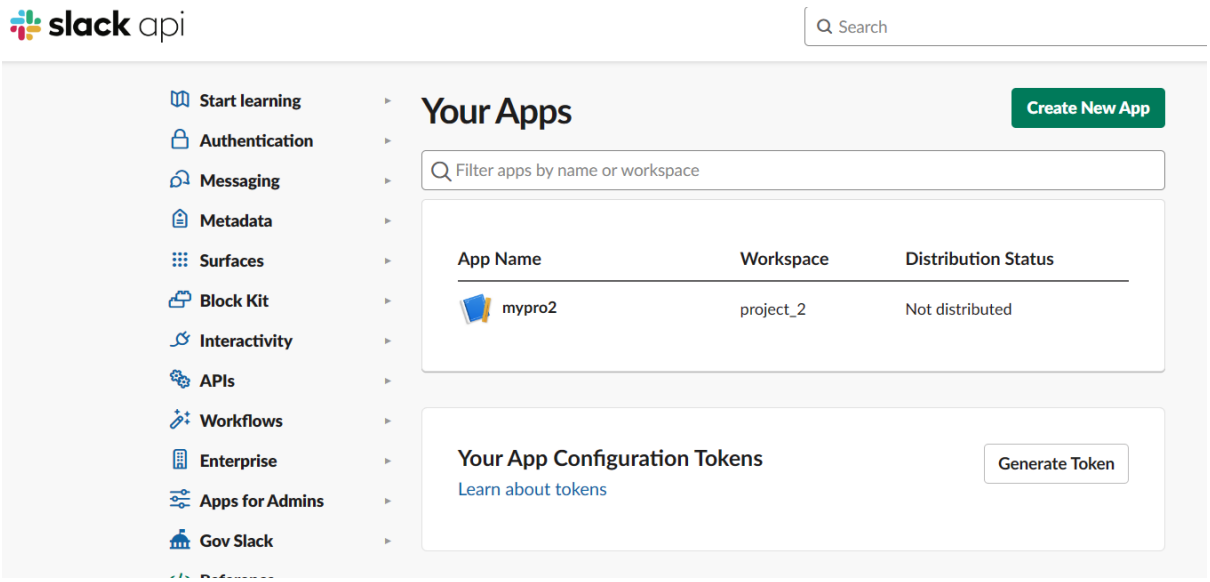


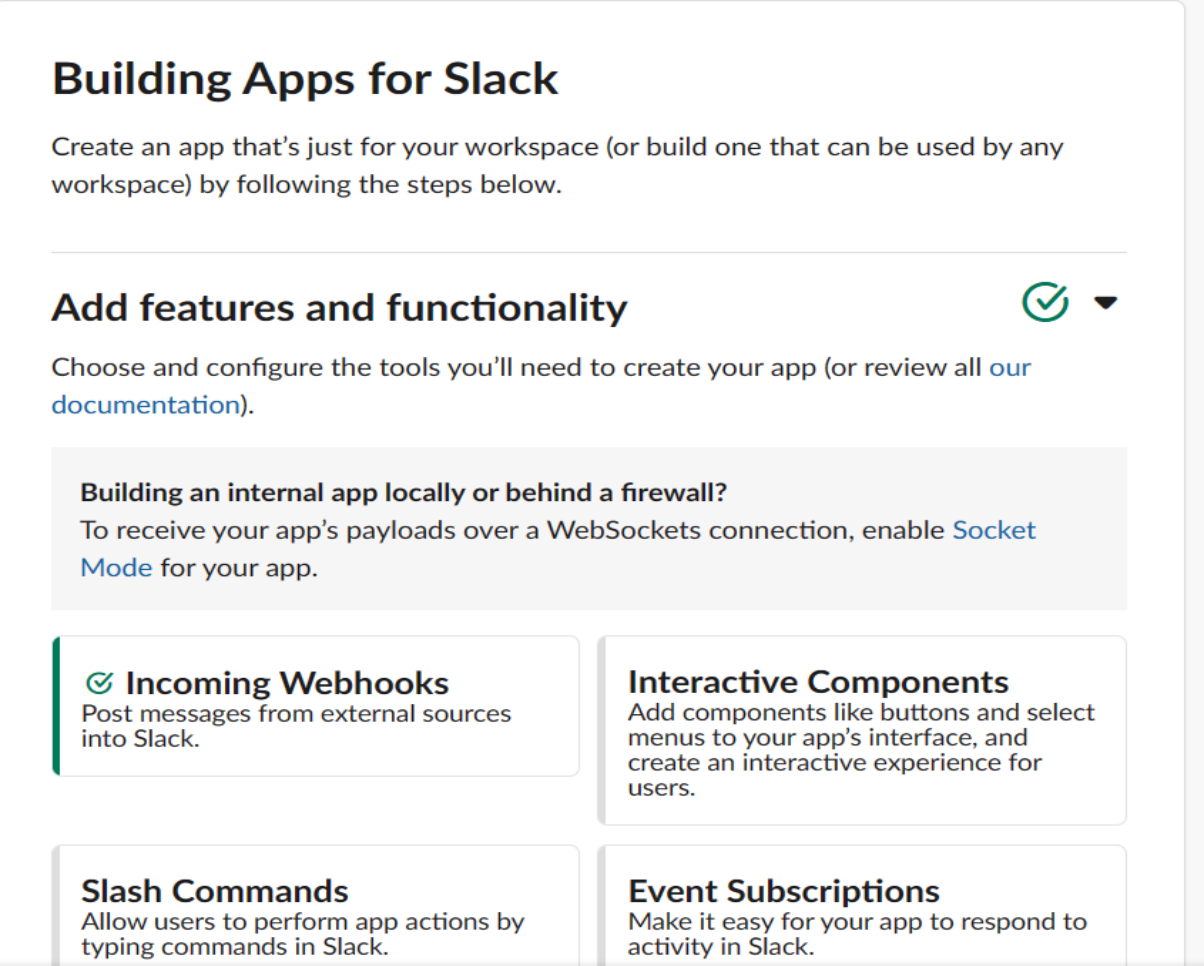
Slack integration:

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





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.

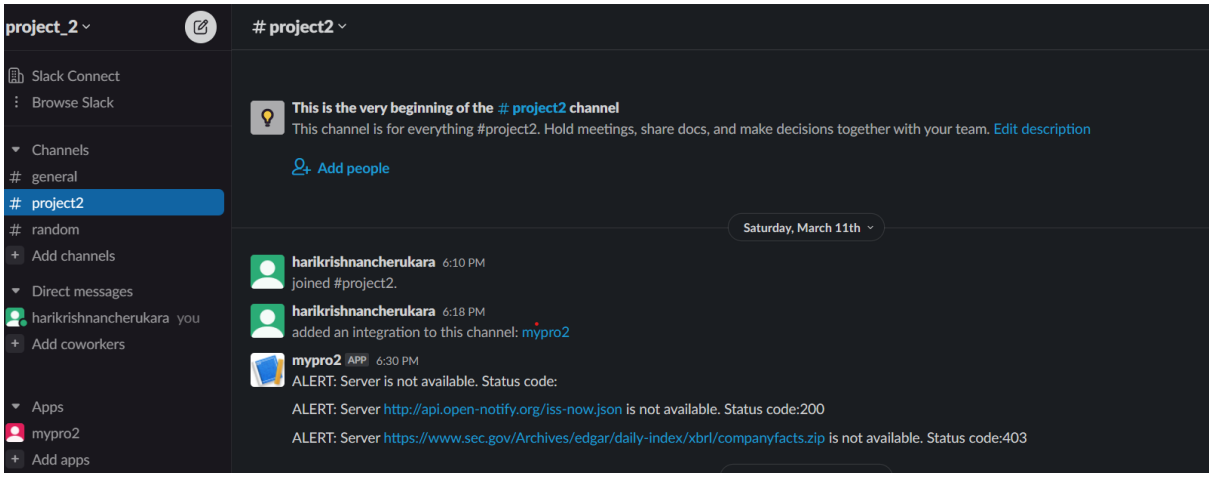
Add this webhook to your workspace below to activate this curl example.

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/B04TGRTQY11/6D9h6yJkgqxn6YlmTa0VAj9
L
```

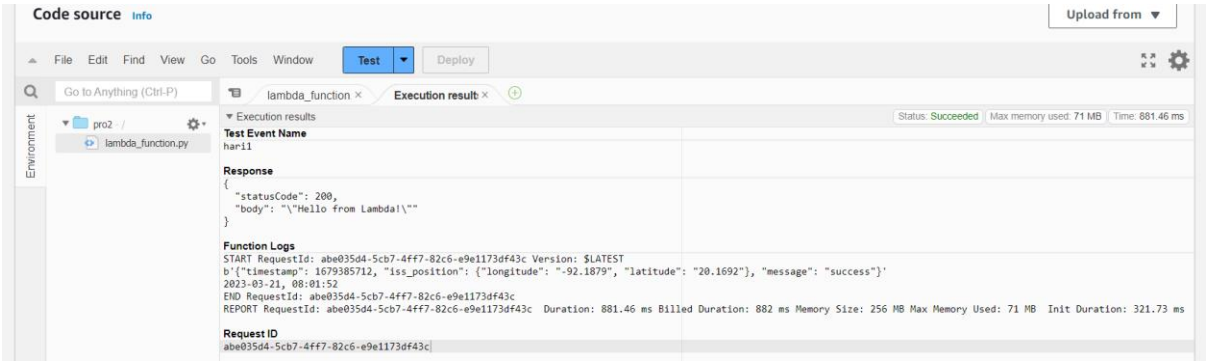
Copy

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



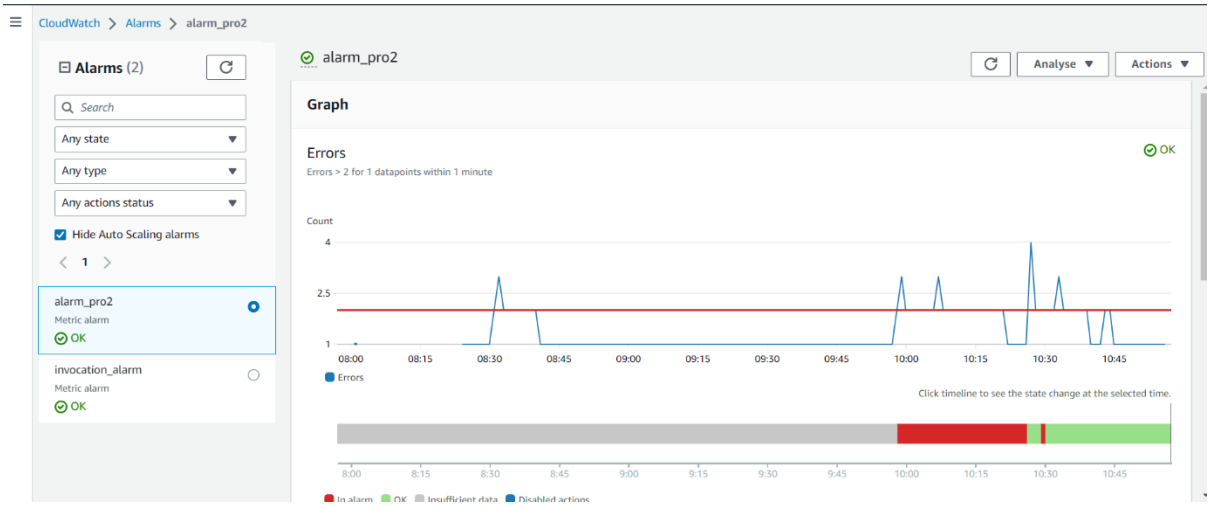
Whenever the request doesn't parse through, an alert will be automatically posted in the slack chat box as shown above. The test URL above mentioned was used to verify the recurring automated messages

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.