A PROJECT REPORT

on

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

Submitted to

GUVI GEEK NETWORK

In partial fulfilment of the requirements for the award CCM

IN

MASTER DATA ENGINEERING COURSE

By

PALUKURI SRINIVAS

E5 – Data Engineering batch

Under the esteemed guidance of
DE mentors,

"GUVI"

GIVEN PROJECT STATEMENT:

Project 2:

Project Title	**Designing an Automatic Data Collection and Storage System with AWS Lambda and Slack Integration for Server Availability Monitoring and Slack Notification**
Technologies	**AWS Lambda, Amazon RDS, Cloud Watch, Slack API**

Problem Statement:

You are tasked with creating 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 Cloud Watch Event that occurs every 15 seconds.

To fetch the data from the API, the function should use the requests library (or a similar library) to make a GET request to the API. The function should then use a library such as psycopg2 to connect to the Amazon RDS instance and store the data in the database.

In addition to fetching and storing the data, the function should also use Amazon Cloud Watch to monitor the server and send an alert to a Slack community if the server goes down. This can be done using the Slack API. Overall, the function should be able to run indefinitely and continue to fetch and store the data on a regular basis.

Approach:

1. Create an AWS Lambda function and configure it to be triggered by an Amazon Cloud Watch Event that occurs every 15 seconds.
2. In the function's code, use the requests library to make a GET request to the API to fetch the data.
3. Use a library such as psycopg2 to connect to the Amazon RDS instance and store the data in the database.
4. Use Amazon Cloud Watch to set up a monitoring alarm that will trigger when the server is unavailable.
5. Use the Slack API to send a message to your Slack community when the alarm is triggered.
6. Test the function to ensure that it is able to fetch and store the data correctly, and that the monitoring and alerting functionality is working as expected.
7. Deploy the function to run indefinitely, continuing to fetch and store the data on a regular basis.

Results:

The result of the above approach would be an AWS Lambda function that is continuously running and performing the following tasks:

- Fetching data from an API on a regular basis (every 15 seconds).

- Storing the fetched data in an Amazon RDS database.

- Monitoring the server's availability using a Cloud Watch Alarm.

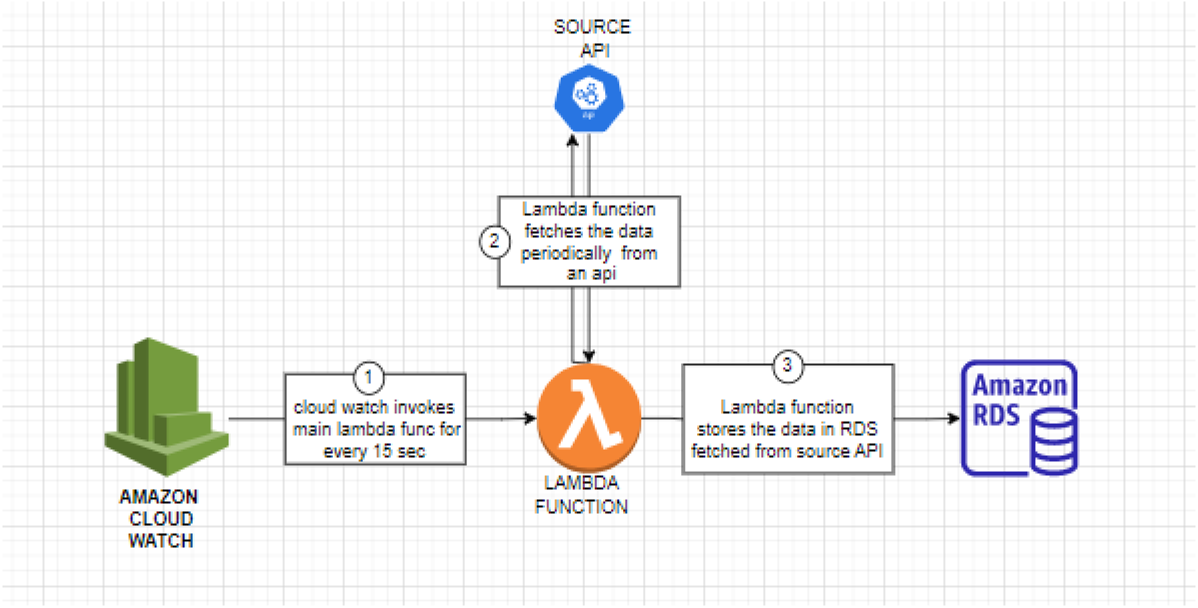
- Sending a notification to a Slack channel if the server becomes unavailable.

The function will continue to run and perform these tasks until it is stopped or modified. The Amazon RDS database will contain the data fetched from the API, and the Cloud Watch Alarm will be triggered if the server becomes unavailable. The Slack notification will alert users that the server is unavailable, and provide details on the status of the server. The function and the database can be monitored to ensure that they are running and storing data correctly.

PROJECT DESCRIPTION:

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

Project explained in below diagram:

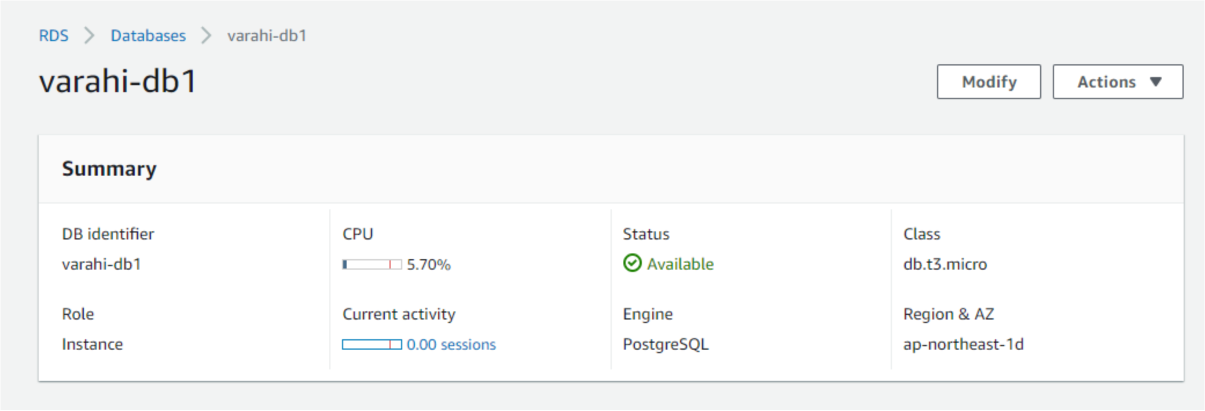

SOURCE API
Lambda function fetches the data periodically from an api
2
1
cloud watch invokes main lambda func for every 15 sec
3
Lambda function stores the data in RDS fetched from source API
Amazon RDS
AMAZON CLOUD WATCH
LAMBDA FUNCTION

Project divided into parts

1. Creation of AWS RDS with POSTGRES SQL
2. Creation of lambda function to get the data from source and storing the data in RDS
3. Creation of cloud watch event to trigger the main lambda function.
4. Creation of cloud watch alarm for source server availability

1.Creation of RDS with Postgres:

- Created varahi-db1



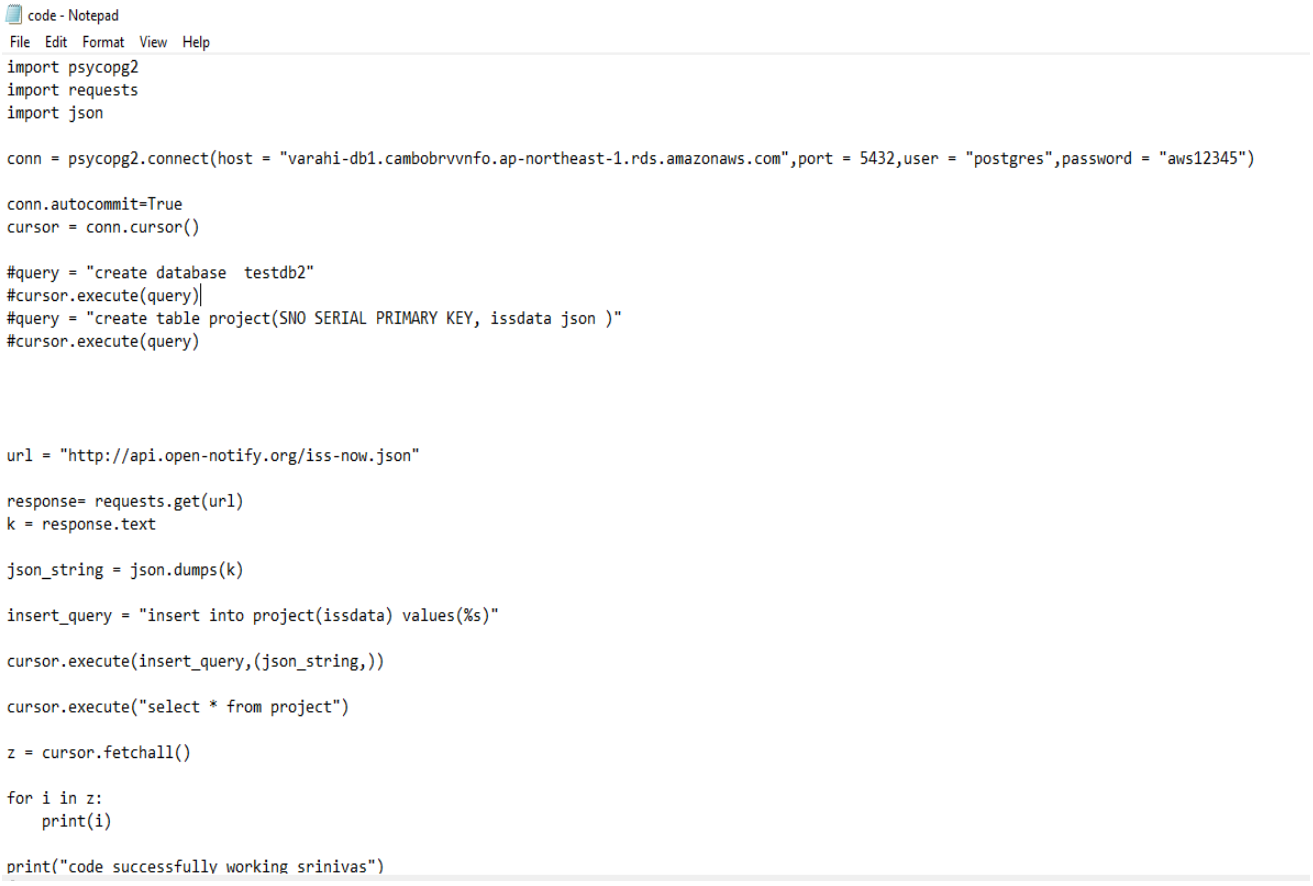
varahi-db1

Summary

DB identifier	CPU	Status	Class
varahi-db1	5.70%	✓ Available	db.t3.micro
Role	Current activity	Engine	Region & AZ
Instance	0.00 sessions	PostgreSQL	ap-northeast-1d

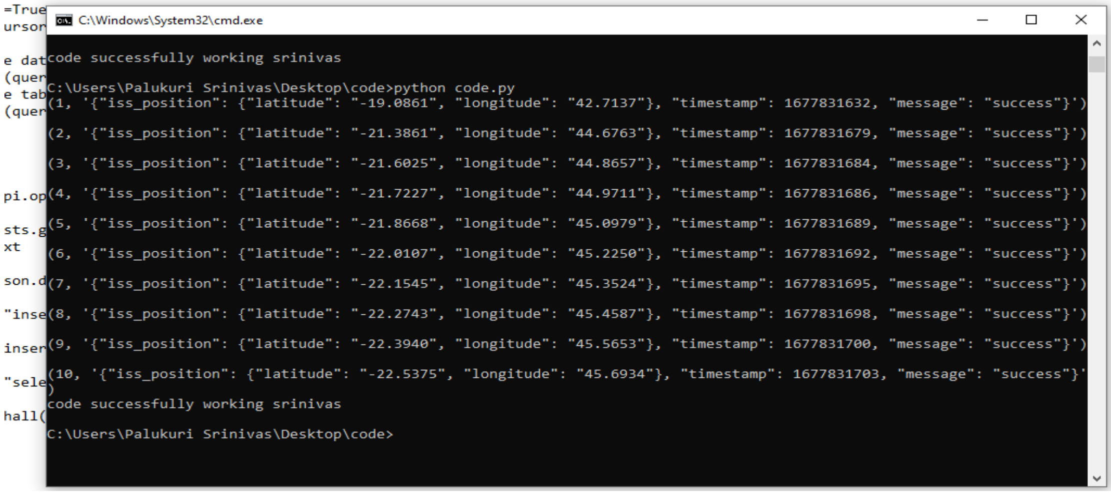
2. Creation of lambda function:

- Source code developed in local. Code snippet attached below.



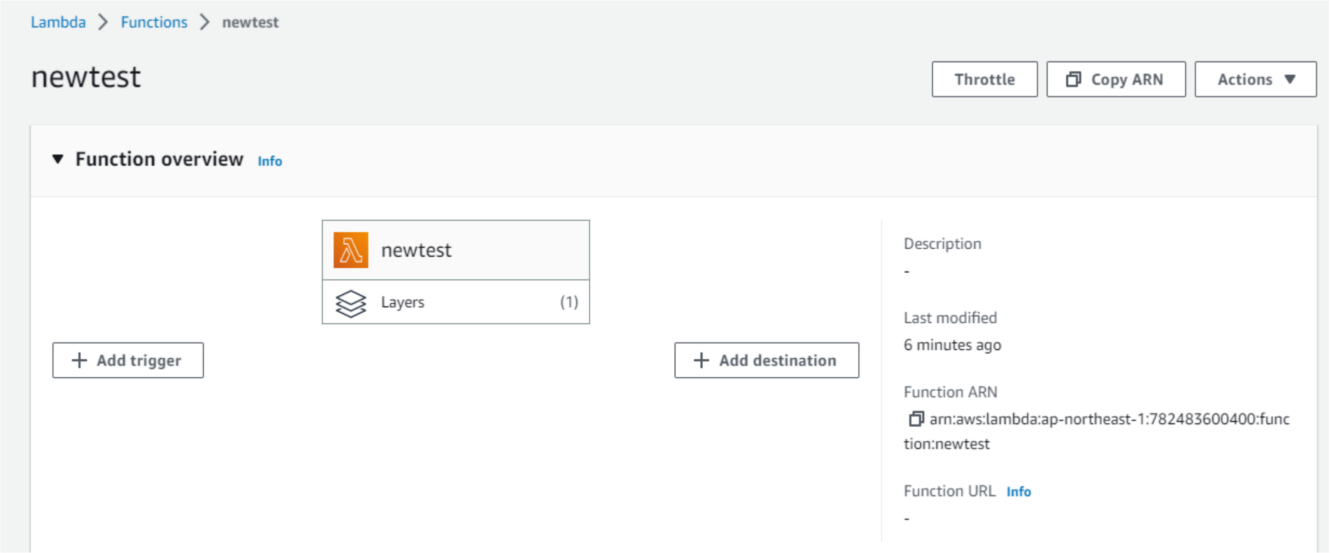
```
import psycopg2
import requests
import json

conn = psycopg2.connect(host = "varahi-db1.cambobrvvnfo.ap-northeast-1.rds.amazonaws.com",port = 5432,user = "postgres",password = "aws12345")

conn.autocommit=True
cursor = conn.cursor()

#query = "create database  testdb2"
#cursor.execute(query)
#query = "create table project(SNO SERIAL PRIMARY KEY, issdata json )"
#cursor.execute(query)



url = "http://api.open-notify.org/iss-now.json"

response= requests.get(url)
k = response.text

json_string = json.dumps(k)

insert_query = "insert into project(issdata) values(%s)"

cursor.execute(insert_query,(json_string,))

cursor.execute("select * from project")

z = cursor.fetchall()

for i in z:
    print(i)

print("code successfully working srinivas")
```

- The Source code execution found successful in local system. Snippet attached below.



```
code successfully working srinivas

C:\Users\Palukuri Srinivas\Desktop\code>python code.py
(1, '{"iss_position": {"latitude": "-19.0861", "longitude": "42.7137"}, "timestamp": 1677831632, "message": "success"}')

(2, '{"iss_position": {"latitude": "-21.3861", "longitude": "44.6763"}, "timestamp": 1677831679, "message": "success"}')

(3, '{"iss_position": {"latitude": "-21.6025", "longitude": "44.8657"}, "timestamp": 1677831684, "message": "success"}')

(4, '{"iss_position": {"latitude": "-21.7227", "longitude": "44.9711"}, "timestamp": 1677831686, "message": "success"}')

(5, '{"iss_position": {"latitude": "-21.8668", "longitude": "45.0979"}, "timestamp": 1677831689, "message": "success"}')

(6, '{"iss_position": {"latitude": "-22.0107", "longitude": "45.2250"}, "timestamp": 1677831692, "message": "success"}')

(7, '{"iss_position": {"latitude": "-22.1545", "longitude": "45.3524"}, "timestamp": 1677831695, "message": "success"}')

(8, '{"iss_position": {"latitude": "-22.2743", "longitude": "45.4587"}, "timestamp": 1677831698, "message": "success"}')

(9, '{"iss_position": {"latitude": "-22.3940", "longitude": "45.5653"}, "timestamp": 1677831700, "message": "success"}')

(10, '{"iss_position": {"latitude": "-22.5375", "longitude": "45.6934"}, "timestamp": 1677831703, "message": "success"}'
)
code successfully working srinivas

C:\Users\Palukuri Srinivas\Desktop\code>
```

- Created AWS lambda function - newtest



newtest

▼ Function overview

newtest

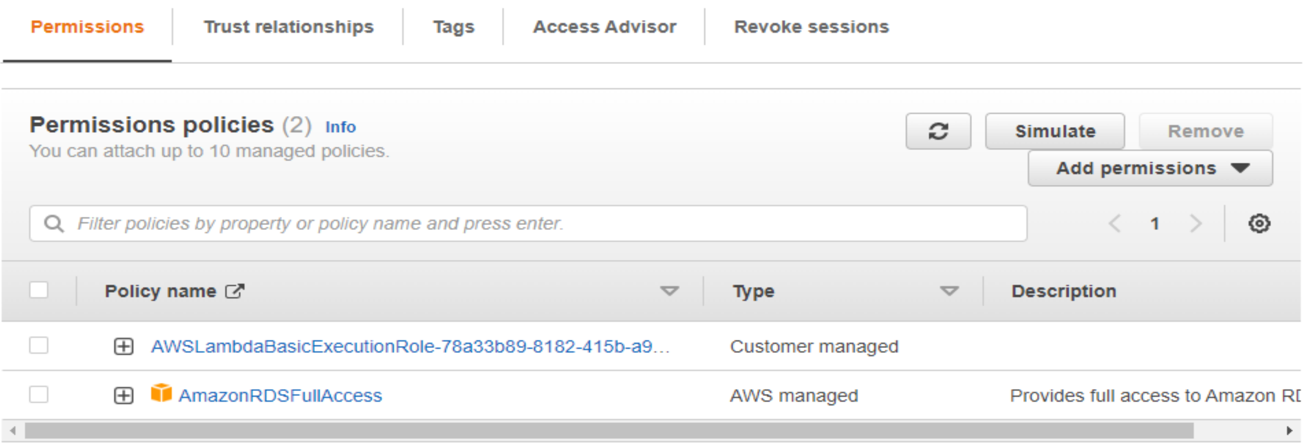
Description
-

Last modified
6 minutes ago

Function ARN
arn:aws:lambda:ap-northeast-1:782483600400:function:newtest

Function URL
-

Permissions given for newtest lambda function:

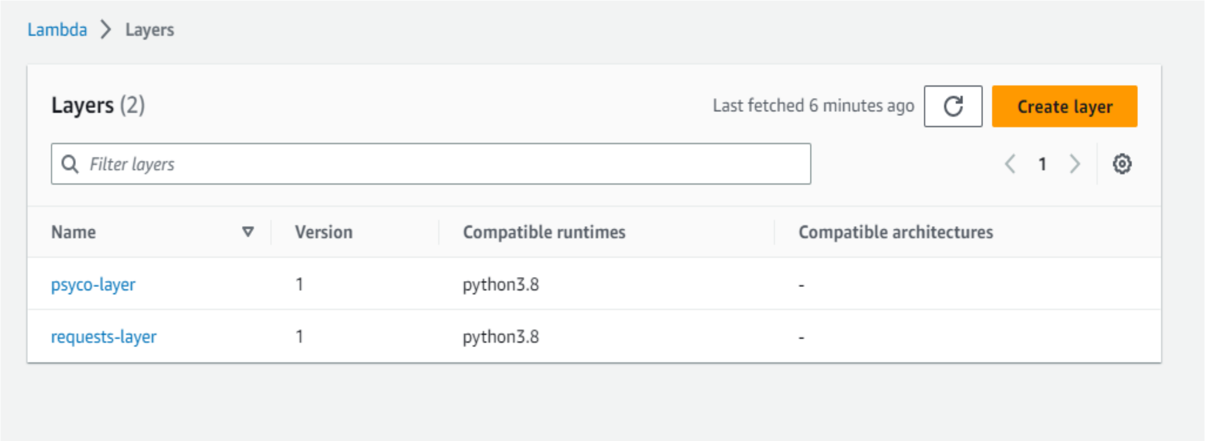


Permissions policies (2)
You can attach up to 10 managed policies.

	Policy name	Type	Description
□	AWSLambdaBasicExecutionRole-78a33b89-8182-415b-a9…	Customer managed	
□	AmazonRDSFullAccess	AWS managed	Provides full access to Amazon RI

Created the lambda layer to import the required libraries

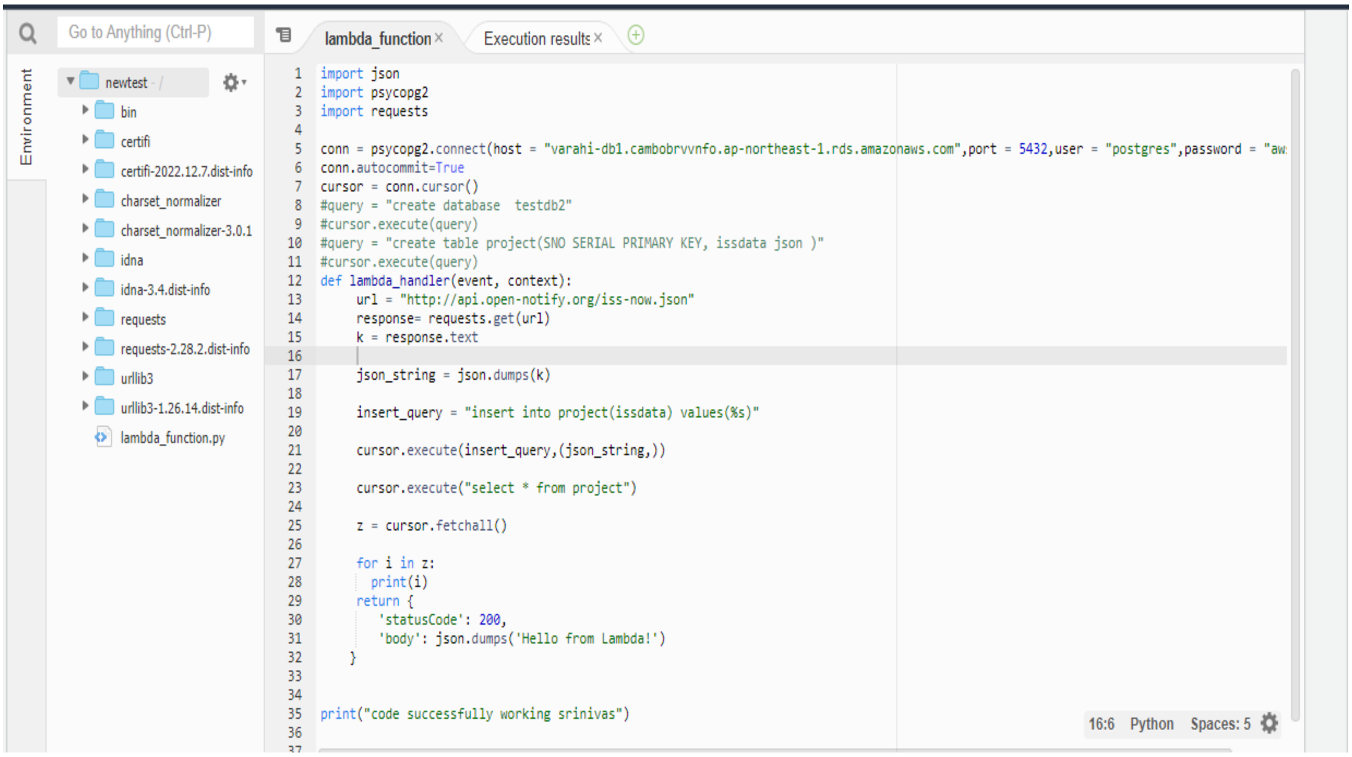
1. Psycopg2
2. Request



Layers (2)

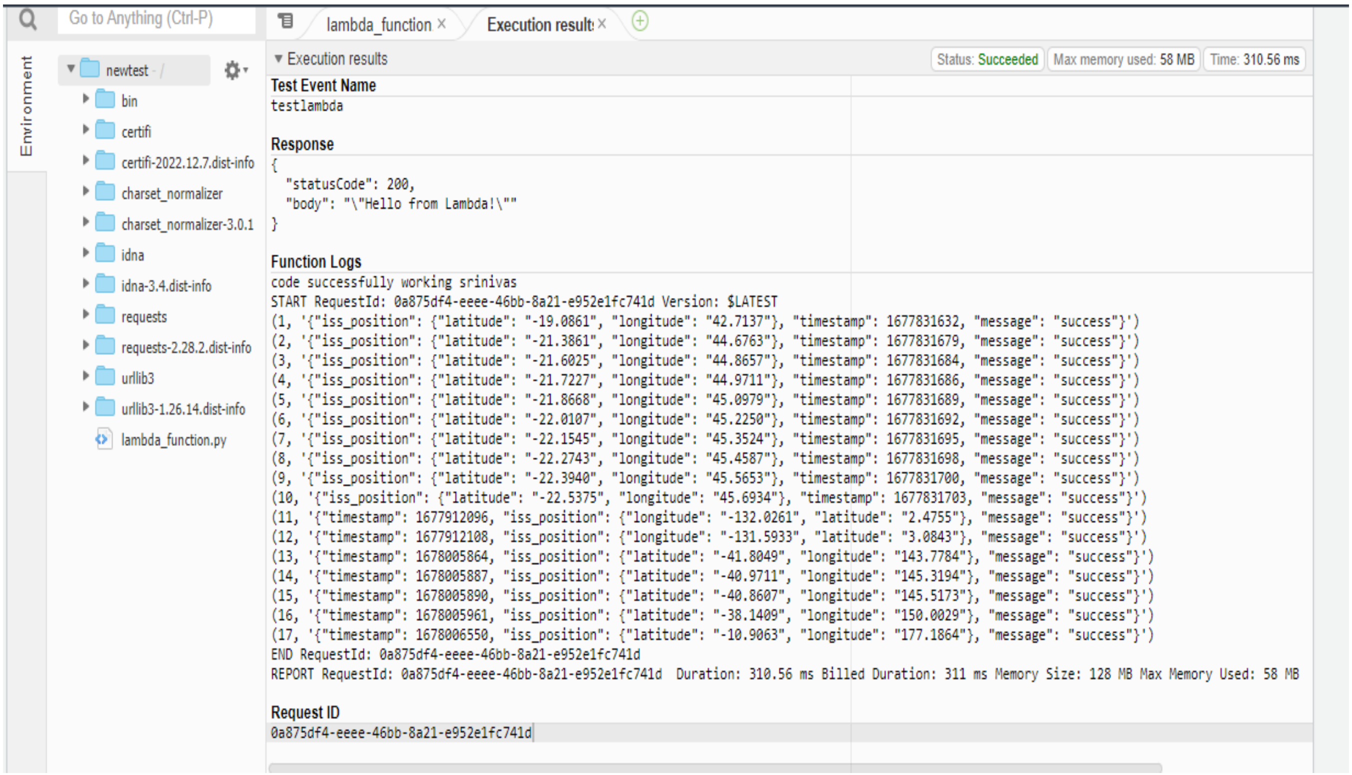
Name	Version	Compatible runtimes	Compatible architectures
psyco-layer	1	python3.8	-
requests-layer	1	python3.8	-

- Uploaded our source code to lambda function with zip file upload option.



```
import json
import psycopg2
import requests

conn = psycopg2.connect(host = "varahi-db1.cambobrvvnfo.ap-northeast-1.rds.amazonaws.com",port = 5432,user = "postgres",password = "aw
conn.autocommit=True
cursor = conn.cursor()
#query = "create database  testdb2"
#cursor.execute(query)
#query = "create table project(SNO SERIAL PRIMARY KEY, issdata json )"
#cursor.execute(query)
def lambda_handler(event, context):
     url = "http://api.open-notify.org/iss-now.json"
     response= requests.get(url)
     k = response.text

     json_string = json.dumps(k)

     insert_query = "insert into project(issdata) values(%s)"

     cursor.execute(insert_query,(json_string,))

     cursor.execute("select * from project")

     z = cursor.fetchall()

     for i in z:
       print(i)
     return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }


print("code successfully working srinivas")
```

- Lambda code tested by running lambda code manually got result as successful



▾ Execution results

Status: Succeeded | Max memory used: 58 MB | Time: 310.56 ms

Test Event Name

```
testlambda
```

Response

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Function Logs

```
code successfully working srinivas
START RequestId: 0a875df4-eeee-46bb-8a21-e952e1fc741d Version: $LATEST
(1, '{"iss_position": {"latitude": "-19.0861", "longitude": "42.7137"}, "timestamp": 1677831632, "message": "success"}')
(2, '{"iss_position": {"latitude": "-21.3861", "longitude": "44.6763"}, "timestamp": 1677831679, "message": "success"}')
(3, '{"iss_position": {"latitude": "-21.6025", "longitude": "44.8657"}, "timestamp": 1677831684, "message": "success"}')
(4, '{"iss_position": {"latitude": "-21.7227", "longitude": "44.9711"}, "timestamp": 1677831686, "message": "success"}')
(5, '{"iss_position": {"latitude": "-21.8668", "longitude": "45.0979"}, "timestamp": 1677831689, "message": "success"}')
(6, '{"iss_position": {"latitude": "-22.0107", "longitude": "45.2250"}, "timestamp": 1677831692, "message": "success"}')
(7, '{"iss_position": {"latitude": "-22.1545", "longitude": "45.3524"}, "timestamp": 1677831695, "message": "success"}')
(8, '{"iss_position": {"latitude": "-22.2743", "longitude": "45.4587"}, "timestamp": 1677831698, "message": "success"}')
(9, '{"iss_position": {"latitude": "-22.3940", "longitude": "45.5653"}, "timestamp": 1677831700, "message": "success"}')
(10, '{"iss_position": {"latitude": "-22.5375", "longitude": "45.6934"}, "timestamp": 1677831703, "message": "success"}')
(11, '{"timestamp": 1677912096, "iss_position": {"longitude": "-132.0261", "latitude": "2.4755"}, "message": "success"}')
(12, '{"timestamp": 1677912108, "iss_position": {"longitude": "-131.5933", "latitude": "3.0843"}, "message": "success"}')
(13, '{"timestamp": 1678005864, "iss_position": {"latitude": "-41.8049", "longitude": "143.7784"}, "message": "success"}')
(14, '{"timestamp": 1678005887, "iss_position": {"latitude": "-40.9711", "longitude": "145.3194"}, "message": "success"}')
(15, '{"timestamp": 1678005890, "iss_position": {"latitude": "-40.8607", "longitude": "145.5173"}, "message": "success"}')
(16, '{"timestamp": 1678005961, "iss_position": {"latitude": "-38.1409", "longitude": "150.0029"}, "message": "success"}')
(17, '{"timestamp": 1678006550, "iss_position": {"latitude": "-10.9063", "longitude": "177.1864"}, "message": "success"}')
END RequestId: 0a875df4-eeee-46bb-8a21-e952e1fc741d
REPORT RequestId: 0a875df4-eeee-46bb-8a21-e952e1fc741d  Duration: 310.56 ms Billed Duration: 311 ms Memory Size: 128 MB Max Memory Used: 58 MB
```

Request ID

```
0a875df4-eeee-46bb-8a21-e952e1fc741d
```

3. Creation of cloud watch event to trigger the lambda function:

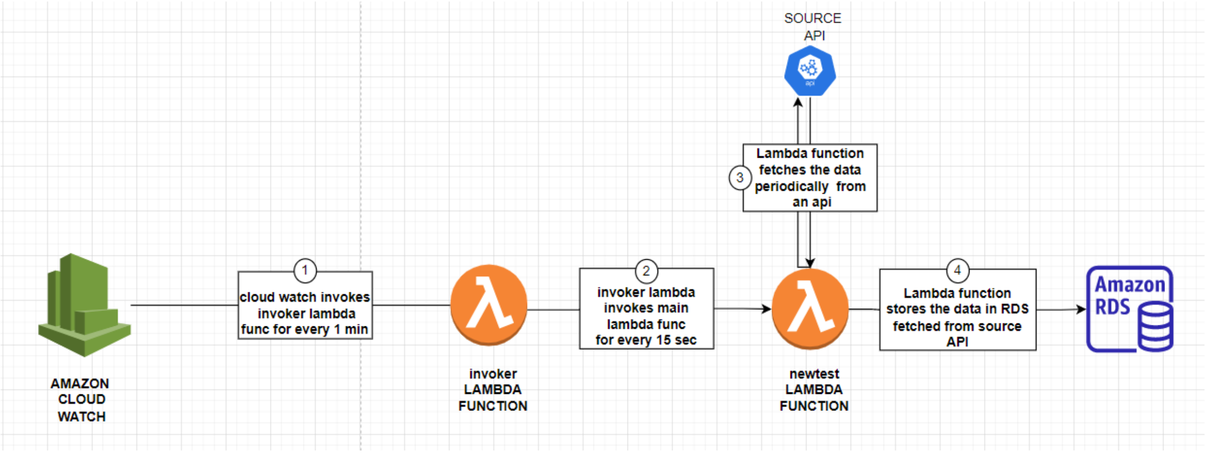
If you've used AWS cloud watch Events to schedule the invocation of a Lambda function at regular intervals, you may have noticed that the highest frequency possible is one invocation per minute. However, in some cases, you may need to invoke Lambda more often than that. In our project description we need to collect the data for every 15sec.

To solve this we are following the sub-minute invocation concept to invoke out newtest function.

For this we are creating one more lambda function to invoke our main lambda function for every 15sec.

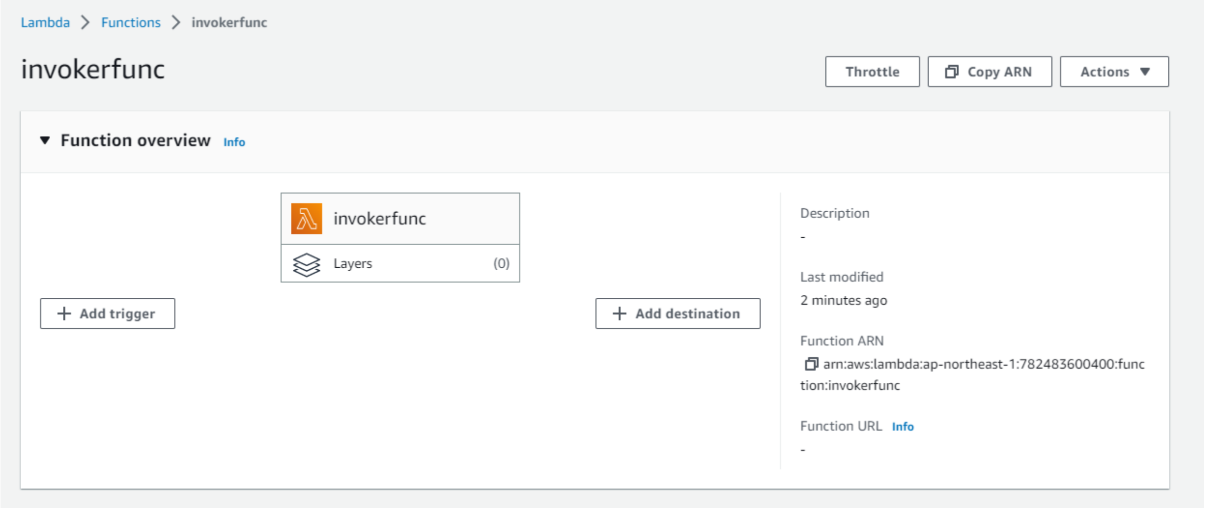
Sub-minute invocation: Creating one more lambda function to invoke our newtest function.

Sub minute invocation flow diagram:


SOURCE API
API
3
Lambda function fetches the data periodically from an api
AMAZON CLOUD WATCH
1
cloud watch invokes invoker lambda func for every 1 min
invoker LAMBDA FUNCTION
2
invoker lambda invokes main lambda func for every 15 sec
newtest LAMBDA FUNCTION
4
Lambda function stores the data in RDS fetched from source API
Amazon RDS

Creation of invokerfunc lambda function:

- Our main goal to create this invokerfunc for invoking the newtest lambda functions for every 15sec.



Lambda > Functions > invokerfunc

invokerfunc

Throttle | Copy ARN | Actions ▼

▼ Function overview Info

invokerfunc

Layers (0)

+ Add trigger

+ Add destination

Description
-

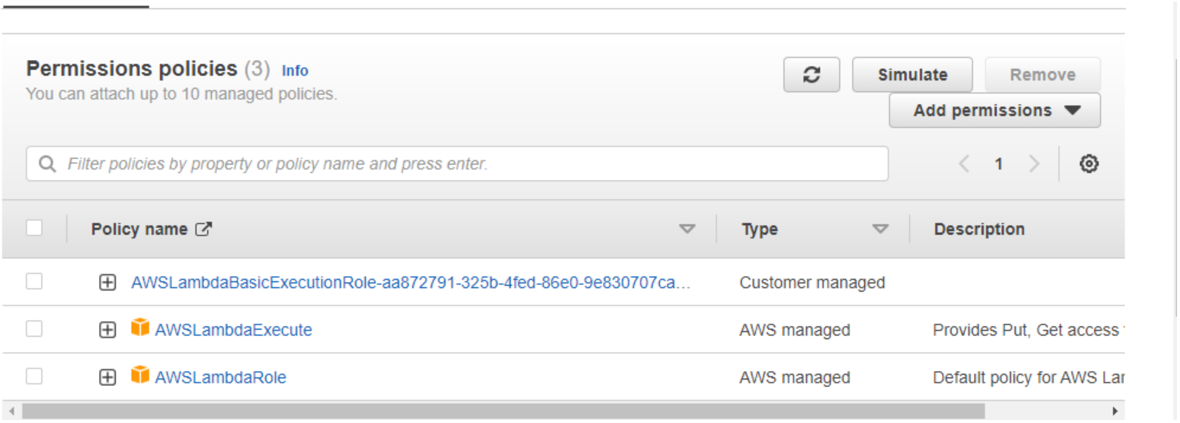
Last modified
2 minutes ago

Function ARN
arn:aws:lambda:ap-northeast-1:782483600400:function:invokerfunc

Function URL Info
-

- Invoker lambda function created.

Permissions given for invoker lambda function:

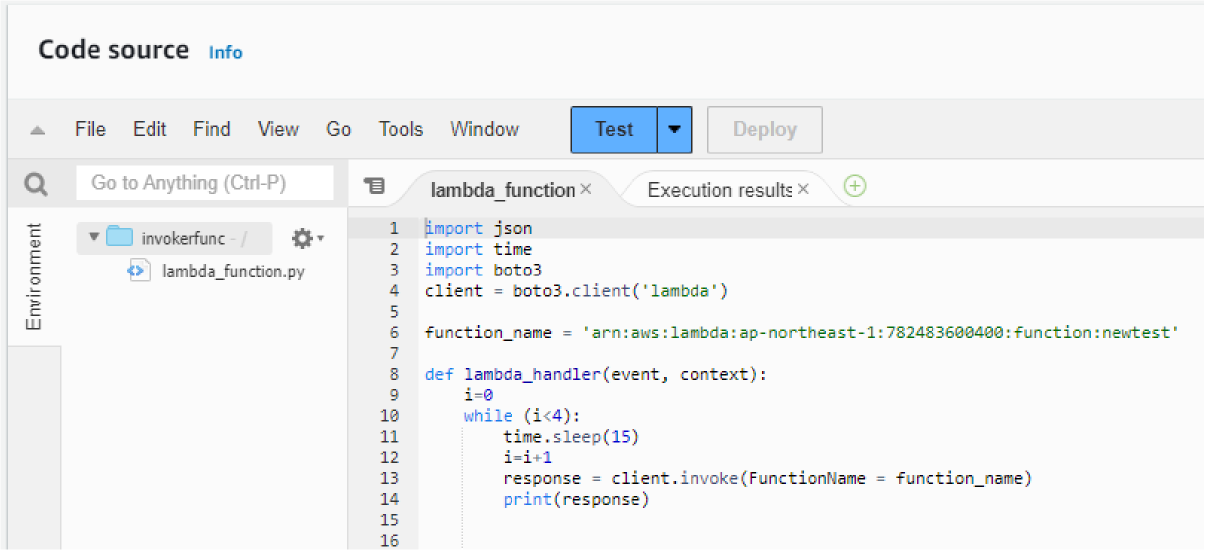


Permissions policies (3) Info

You can attach up to 10 managed policies.

□	Policy name	Type	Description
□	⊞ AWSLambdaBasicExecutionRole-aa872791-325b-4fed-86e0-9e830707ca…	Customer managed	
□	⊞ AWSLambdaExecute	AWS managed	Provides Put, Get access
□	⊞ AWSLambdaRole	AWS managed	Default policy for AWS Lar

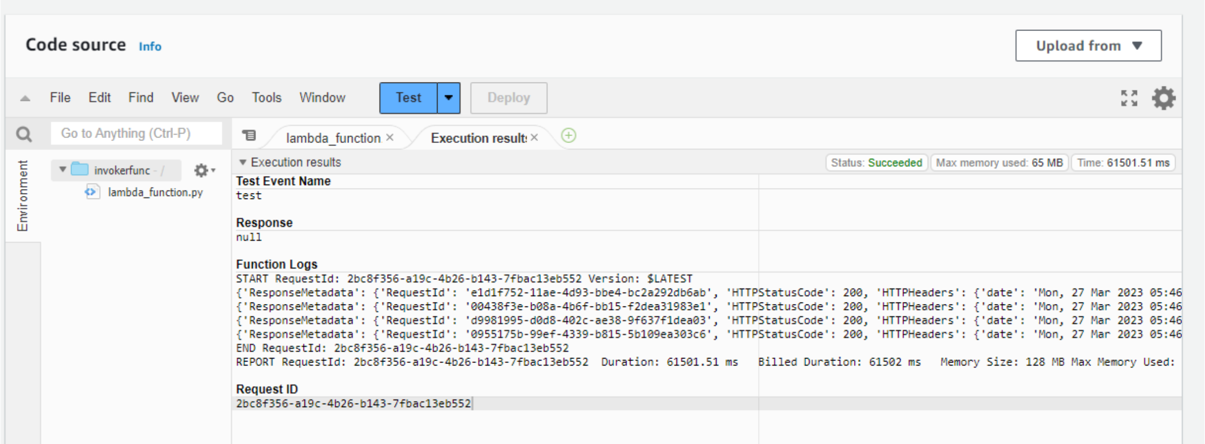
Code used for invoker lambda function:



```
import json
import time
import boto3
client = boto3.client('lambda')

function_name = 'arn:aws:lambda:ap-northeast-1:782483600400:function:newtest'

def lambda_handler(event, context):
    i=0
    while (i<4):
        time.sleep(15)
        i=i+1
        response = client.invoke(FunctionName = function_name)
        print(response)

```

The execution results for the invoker lambda function when manually tested:

- It’s running 4 times in for every time we tested it.



Status: Succeeded | Max memory used: 65 MB | Time: 61501.51 ms

Test Event Name
```
test
```

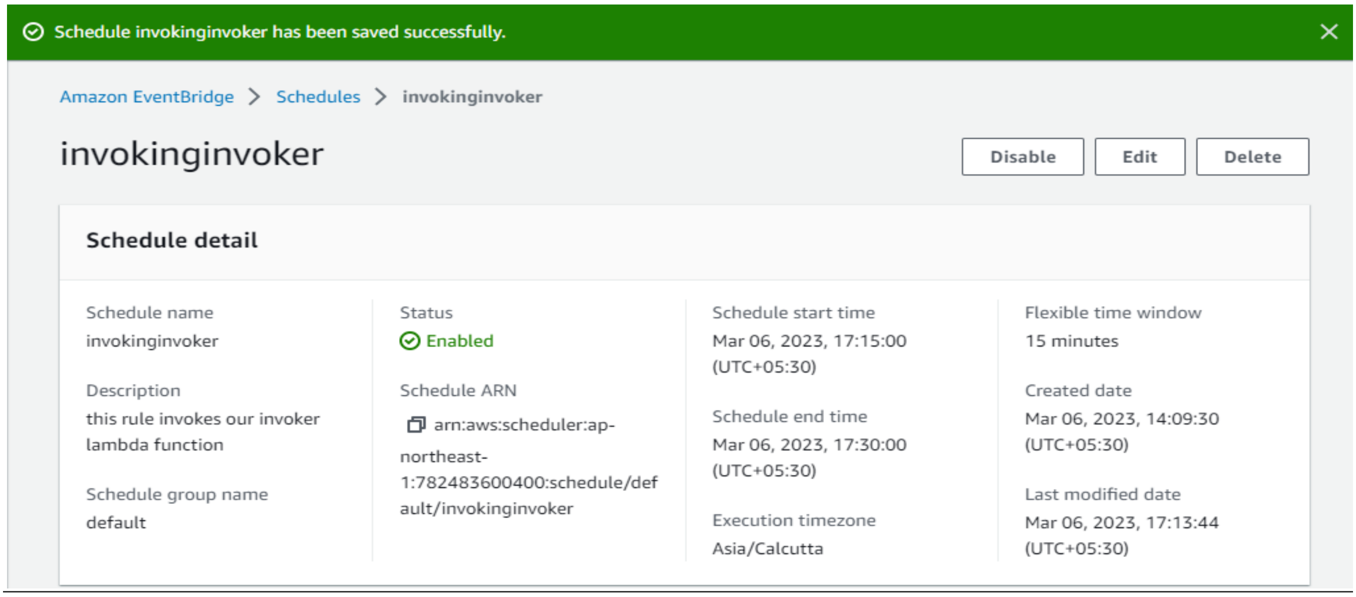
Response
```
null
```

Function Logs
```
START RequestId: 2bc8f356-a19c-4b26-b143-7fbac13eb552 Version: $LATEST
{'ResponseMetadata': {'RequestId': 'e1d1f752-11ae-4d93-bbe4-bc2a292db6ab', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Mon, 27 Mar 2023 05:46
{'ResponseMetadata': {'RequestId': '00438f3e-b08a-4b6f-bb15-f2dea31983e1', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Mon, 27 Mar 2023 05:46
{'ResponseMetadata': {'RequestId': 'd9981995-d0d8-402c-ae38-9f637f1dea03', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Mon, 27 Mar 2023 05:46
{'ResponseMetadata': {'RequestId': '0955175b-99ef-4339-b815-5b109ea303c6', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Mon, 27 Mar 2023 05:46
END RequestId: 2bc8f356-a19c-4b26-b143-7fbac13eb552
REPORT RequestId: 2bc8f356-a19c-4b26-b143-7fbac13eb552  Duration: 61501.51 ms   Billed Duration: 61502 ms   Memory Size: 128 MB Max Memory Used:
```

Request ID
```
2bc8f356-a19c-4b26-b143-7fbac13eb552
```

- As we know invoker lambda function will run for 4 times in a loop and every time it will invokes our newtest lambda function.
- Now we need to create the cloud watch schedule event to run our invoker lambda function for every minute.

Creating cloud watch schedule event:



Schedule invokinginvoker has been saved successfully.

invokinginvoker

Schedule detail

Schedule name
invokinginvoker

Description
this rule invokes our invoker lambda function

Schedule group name
default

Status
Enabled

Schedule ARN
arn:aws:scheduler:ap-northeast-1:782483600400:schedule/default/invokinginvoker

Schedule start time
Mar 06, 2023, 17:15:00 (UTC+05:30)

Schedule end time
Mar 06, 2023, 17:30:00 (UTC+05:30)

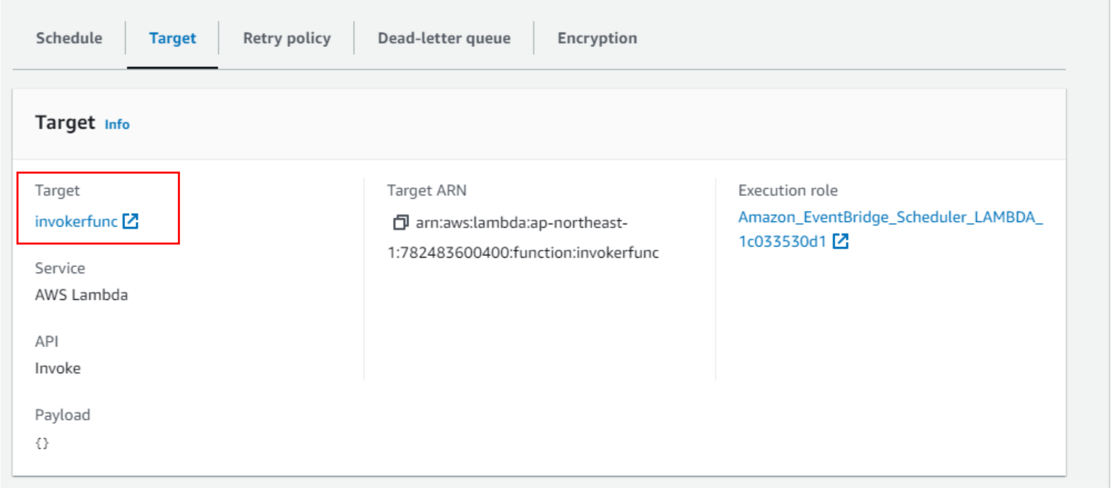
Execution timezone
Asia/Calcutta

Flexible time window
15 minutes

Created date
Mar 06, 2023, 14:09:30 (UTC+05:30)

Last modified date
Mar 06, 2023, 17:13:44 (UTC+05:30)

- Schedule event created 15 minutes.



Target Info

Target
invokerfunc

Service
AWS Lambda

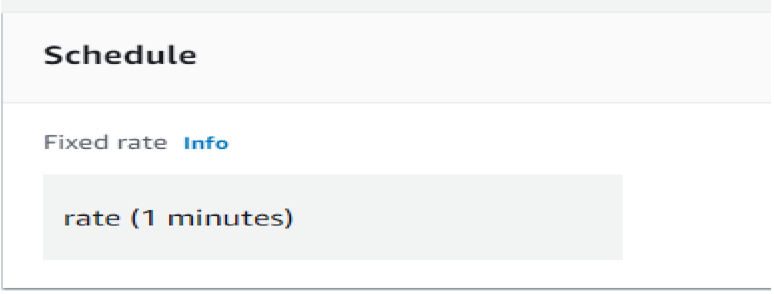
API
Invoke

Payload
{}

Target ARN
arn:aws:lambda:ap-northeast-1:782483600400:function:invokerfunc

Execution role
Amazon_EventBridge_Scheduler_LAMBDA_1c033530d1

- Target given as invokerfunc lambda function. Schedule rate given as 1minute.

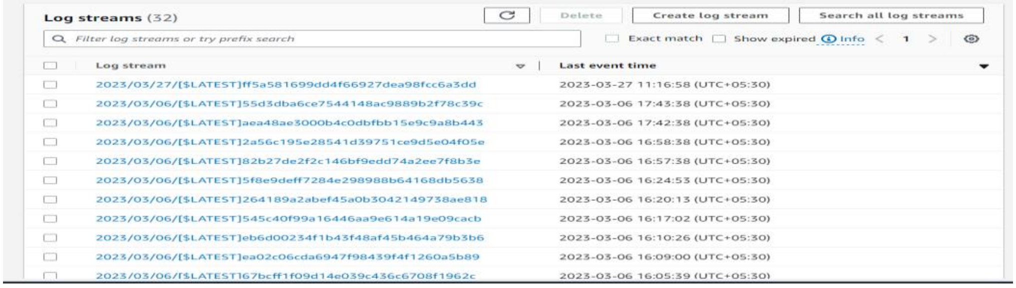


Schedule

Fixed rate Info

rate (1 minutes)

Invokerfunc Cloud watch log events snippet attached below:

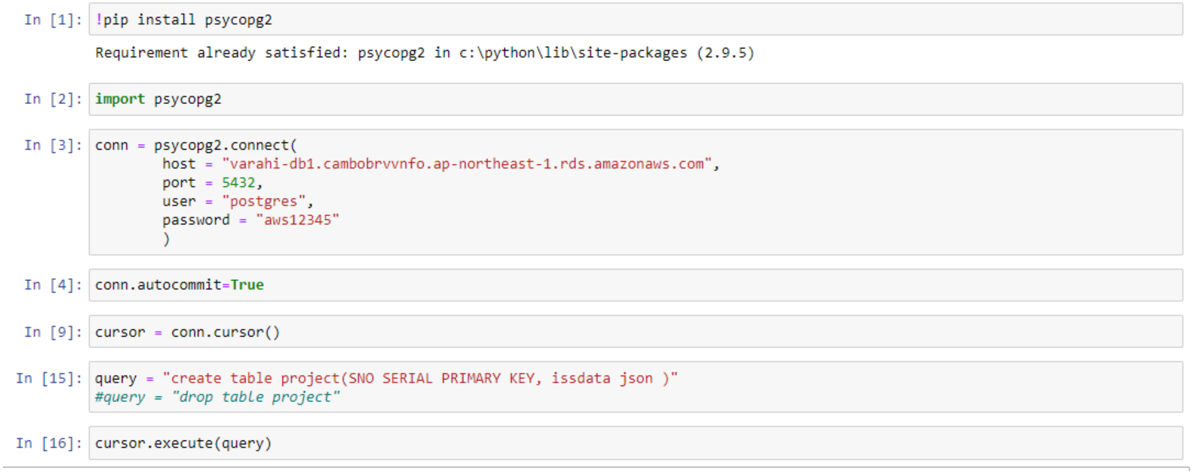


□	Log stream	Last event time
□	2023/03/27/[$LATEST]ff5a581699dd4f66927dea98fcc6a3dd	2023-03-27 11:16:58 (UTC+05:30)
□	2023/03/06/[$LATEST]55d3dba6ce7544148ac9889b2f78c39c	2023-03-06 17:43:38 (UTC+05:30)
□	2023/03/06/[$LATEST]aea48ae3000b4c0dbfbb15e9c9a8b443	2023-03-06 17:42:38 (UTC+05:30)
□	2023/03/06/[$LATEST]2a56c195e28541d39751ce9d5e04f05e	2023-03-06 16:58:38 (UTC+05:30)
□	2023/03/06/[$LATEST]82b27de2f2c146bf9edd74a2ee7f8b3e	2023-03-06 16:57:38 (UTC+05:30)
□	2023/03/06/[$LATEST]5f8e9deff7284e298988b64168db5638	2023-03-06 16:24:53 (UTC+05:30)
□	2023/03/06/[$LATEST]264189a2abef45a0b3042149738ae818	2023-03-06 16:20:13 (UTC+05:30)
□	2023/03/06/[$LATEST]545c40f99a16446aa9e614a19e09cacb	2023-03-06 16:17:02 (UTC+05:30)
□	2023/03/06/[$LATEST]eb6d00234f1b43f48af45b464a79b3b6	2023-03-06 16:10:26 (UTC+05:30)
□	2023/03/06/[$LATEST]ea02c06cda6947f98439f4f1260a5b89	2023-03-06 16:09:00 (UTC+05:30)
□	2023/03/06/[$LATEST]67bcff1f09d14e039c436c6708f1962c	2023-03-06 16:05:39 (UTC+05:30)

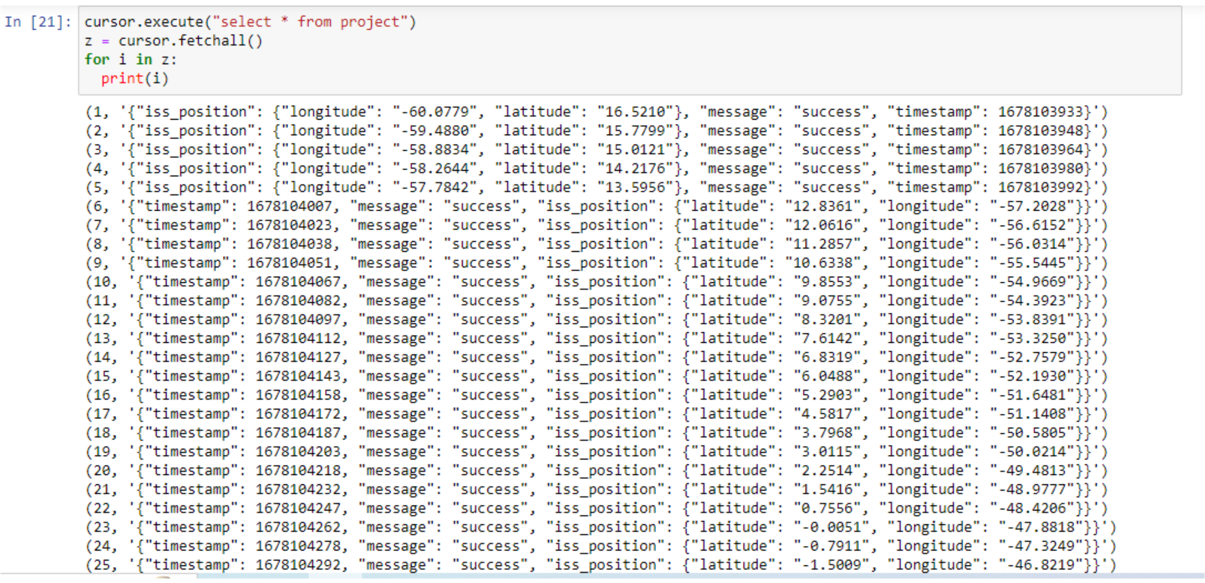
This schedule event will invokes our invokerfunc lambda function for 15 minutes of period for every one minute rate. Invoker lambda function will run for 4 times per one time invocation .

So that our main function (i.e.) newtest lambda function must run 60 times. So it should get the data from our ISS API and stores the data in Postgres RDS for 60 times. So 60 records must be printed.

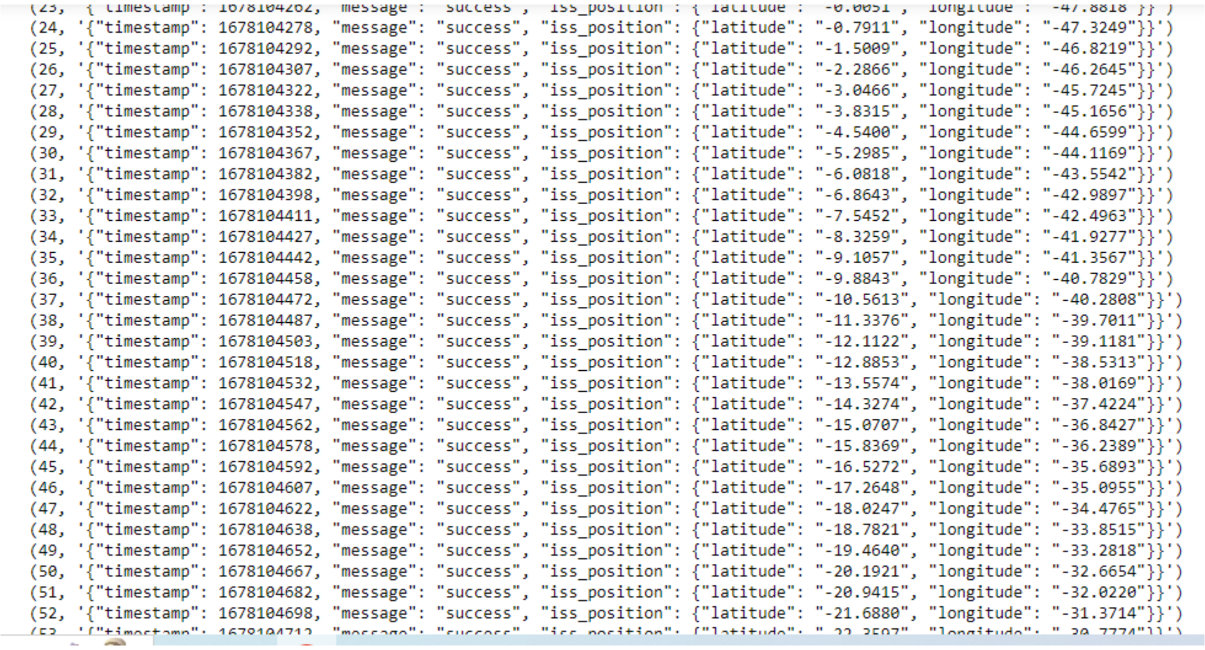
After running the 15 minutes of schedule verified our RDS data base by connecting with Jupiter notebook records. Pictures attached below.



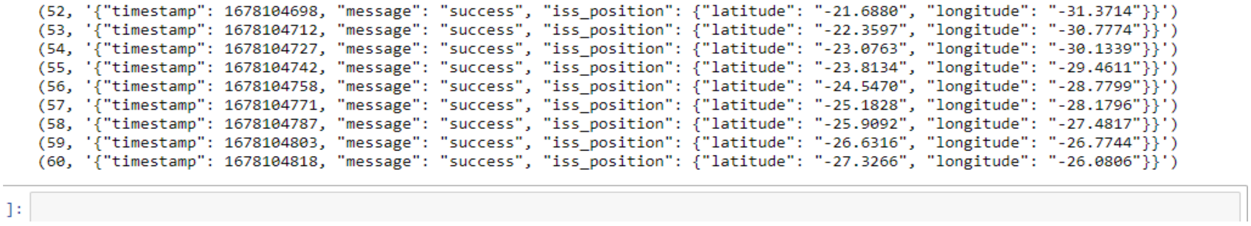
```
In [1]: !pip install psycopg2

        Requirement already satisfied: psycopg2 in c:\python\lib\site-packages (2.9.5)

In [2]: import psycopg2

In [3]: conn = psycopg2.connect(
                host = "varahi-db1.cambobrvvnfo.ap-northeast-1.rds.amazonaws.com",
                port = 5432,
                user = "postgres",
                password = "aws12345"
                )

In [4]: conn.autocommit=True

In [9]: cursor = conn.cursor()

In [15]: query = "create table project(SNO SERIAL PRIMARY KEY, issdata json )"
         #query = "drop table project"

In [16]: cursor.execute(query)
```



```
In [21]: cursor.execute("select * from project")
         z = cursor.fetchall()
         for i in z:
           print(i)

         (1, '{"iss_position": {"longitude": "-60.0779", "latitude": "16.5210"}, "message": "success", "timestamp": 1678103933}')
         (2, '{"iss_position": {"longitude": "-59.4880", "latitude": "15.7799"}, "message": "success", "timestamp": 1678103948}')
         (3, '{"iss_position": {"longitude": "-58.8834", "latitude": "15.0121"}, "message": "success", "timestamp": 1678103964}')
         (4, '{"iss_position": {"longitude": "-58.2644", "latitude": "14.2176"}, "message": "success", "timestamp": 1678103980}')
         (5, '{"iss_position": {"longitude": "-57.7842", "latitude": "13.5956"}, "message": "success", "timestamp": 1678103992}')
         (6, '{"timestamp": 1678104007, "message": "success", "iss_position": {"latitude": "12.8361", "longitude": "-57.2028"}}')
         (7, '{"timestamp": 1678104023, "message": "success", "iss_position": {"latitude": "12.0616", "longitude": "-56.6152"}}')
         (8, '{"timestamp": 1678104038, "message": "success", "iss_position": {"latitude": "11.2857", "longitude": "-56.0314"}}')
         (9, '{"timestamp": 1678104051, "message": "success", "iss_position": {"latitude": "10.6338", "longitude": "-55.5445"}}')
         (10, '{"timestamp": 1678104067, "message": "success", "iss_position": {"latitude": "9.8553", "longitude": "-54.9669"}}')
         (11, '{"timestamp": 1678104082, "message": "success", "iss_position": {"latitude": "9.0755", "longitude": "-54.3923"}}')
         (12, '{"timestamp": 1678104097, "message": "success", "iss_position": {"latitude": "8.3201", "longitude": "-53.8391"}}')
         (13, '{"timestamp": 1678104112, "message": "success", "iss_position": {"latitude": "7.6142", "longitude": "-53.3250"}}')
         (14, '{"timestamp": 1678104127, "message": "success", "iss_position": {"latitude": "6.8319", "longitude": "-52.7579"}}')
         (15, '{"timestamp": 1678104143, "message": "success", "iss_position": {"latitude": "6.0488", "longitude": "-52.1930"}}')
         (16, '{"timestamp": 1678104158, "message": "success", "iss_position": {"latitude": "5.2903", "longitude": "-51.6481"}}')
         (17, '{"timestamp": 1678104172, "message": "success", "iss_position": {"latitude": "4.5817", "longitude": "-51.1408"}}')
         (18, '{"timestamp": 1678104187, "message": "success", "iss_position": {"latitude": "3.7968", "longitude": "-50.5805"}}')
         (19, '{"timestamp": 1678104203, "message": "success", "iss_position": {"latitude": "3.0115", "longitude": "-50.0214"}}')
         (20, '{"timestamp": 1678104218, "message": "success", "iss_position": {"latitude": "2.2514", "longitude": "-49.4813"}}')
         (21, '{"timestamp": 1678104232, "message": "success", "iss_position": {"latitude": "1.5416", "longitude": "-48.9777"}}')
         (22, '{"timestamp": 1678104247, "message": "success", "iss_position": {"latitude": "0.7556", "longitude": "-48.4206"}}')
         (23, '{"timestamp": 1678104262, "message": "success", "iss_position": {"latitude": "-0.0051", "longitude": "-47.8818"}}')
         (24, '{"timestamp": 1678104278, "message": "success", "iss_position": {"latitude": "-0.7911", "longitude": "-47.3249"}}')
         (25, '{"timestamp": 1678104292, "message": "success", "iss_position": {"latitude": "-1.5009", "longitude": "-46.8219"}}')
```



```
(23, '{"timestamp": 1678104262, "message": "success", "iss_position": {"latitude": "-0.0051", "longitude": "-47.8818"}}')
(24, '{"timestamp": 1678104278, "message": "success", "iss_position": {"latitude": "-0.7911", "longitude": "-47.3249"}}')
(25, '{"timestamp": 1678104292, "message": "success", "iss_position": {"latitude": "-1.5009", "longitude": "-46.8219"}}')
(26, '{"timestamp": 1678104307, "message": "success", "iss_position": {"latitude": "-2.2866", "longitude": "-46.2645"}}')
(27, '{"timestamp": 1678104322, "message": "success", "iss_position": {"latitude": "-3.0466", "longitude": "-45.7245"}}')
(28, '{"timestamp": 1678104338, "message": "success", "iss_position": {"latitude": "-3.8315", "longitude": "-45.1656"}}')
(29, '{"timestamp": 1678104352, "message": "success", "iss_position": {"latitude": "-4.5400", "longitude": "-44.6599"}}')
(30, '{"timestamp": 1678104367, "message": "success", "iss_position": {"latitude": "-5.2985", "longitude": "-44.1169"}}')
(31, '{"timestamp": 1678104382, "message": "success", "iss_position": {"latitude": "-6.0818", "longitude": "-43.5542"}}')
(32, '{"timestamp": 1678104398, "message": "success", "iss_position": {"latitude": "-6.8643", "longitude": "-42.9897"}}')
(33, '{"timestamp": 1678104411, "message": "success", "iss_position": {"latitude": "-7.5452", "longitude": "-42.4963"}}')
(34, '{"timestamp": 1678104427, "message": "success", "iss_position": {"latitude": "-8.3259", "longitude": "-41.9277"}}')
(35, '{"timestamp": 1678104442, "message": "success", "iss_position": {"latitude": "-9.1057", "longitude": "-41.3567"}}')
(36, '{"timestamp": 1678104458, "message": "success", "iss_position": {"latitude": "-9.8843", "longitude": "-40.7829"}}')
(37, '{"timestamp": 1678104472, "message": "success", "iss_position": {"latitude": "-10.5613", "longitude": "-40.2808"}}')
(38, '{"timestamp": 1678104487, "message": "success", "iss_position": {"latitude": "-11.3376", "longitude": "-39.7011"}}')
(39, '{"timestamp": 1678104503, "message": "success", "iss_position": {"latitude": "-12.1122", "longitude": "-39.1181"}}')
(40, '{"timestamp": 1678104518, "message": "success", "iss_position": {"latitude": "-12.8853", "longitude": "-38.5313"}}')
(41, '{"timestamp": 1678104532, "message": "success", "iss_position": {"latitude": "-13.5574", "longitude": "-38.0169"}}')
(42, '{"timestamp": 1678104547, "message": "success", "iss_position": {"latitude": "-14.3274", "longitude": "-37.4224"}}')
(43, '{"timestamp": 1678104562, "message": "success", "iss_position": {"latitude": "-15.0707", "longitude": "-36.8427"}}')
(44, '{"timestamp": 1678104578, "message": "success", "iss_position": {"latitude": "-15.8369", "longitude": "-36.2389"}}')
(45, '{"timestamp": 1678104592, "message": "success", "iss_position": {"latitude": "-16.5272", "longitude": "-35.6893"}}')
(46, '{"timestamp": 1678104607, "message": "success", "iss_position": {"latitude": "-17.2648", "longitude": "-35.0955"}}')
(47, '{"timestamp": 1678104622, "message": "success", "iss_position": {"latitude": "-18.0247", "longitude": "-34.4765"}}')
(48, '{"timestamp": 1678104638, "message": "success", "iss_position": {"latitude": "-18.7821", "longitude": "-33.8515"}}')
(49, '{"timestamp": 1678104652, "message": "success", "iss_position": {"latitude": "-19.4640", "longitude": "-33.2818"}}')
(50, '{"timestamp": 1678104667, "message": "success", "iss_position": {"latitude": "-20.1921", "longitude": "-32.6654"}}')
(51, '{"timestamp": 1678104682, "message": "success", "iss_position": {"latitude": "-20.9415", "longitude": "-32.0220"}}')
(52, '{"timestamp": 1678104698, "message": "success", "iss_position": {"latitude": "-21.6880", "longitude": "-31.3714"}}')
(53, '{"timestamp": 1678104712, "message": "success", "iss_position": {"latitude": "-22.3597", "longitude": "-30.7774"}}')
```



```
(52, '{"timestamp": 1678104698, "message": "success", "iss_position": {"latitude": "-21.6880", "longitude": "-31.3714"}}')
(53, '{"timestamp": 1678104712, "message": "success", "iss_position": {"latitude": "-22.3597", "longitude": "-30.7774"}}')
(54, '{"timestamp": 1678104727, "message": "success", "iss_position": {"latitude": "-23.0763", "longitude": "-30.1339"}}')
(55, '{"timestamp": 1678104742, "message": "success", "iss_position": {"latitude": "-23.8134", "longitude": "-29.4611"}}')
(56, '{"timestamp": 1678104758, "message": "success", "iss_position": {"latitude": "-24.5470", "longitude": "-28.7799"}}')
(57, '{"timestamp": 1678104771, "message": "success", "iss_position": {"latitude": "-25.1828", "longitude": "-28.1796"}}')
(58, '{"timestamp": 1678104787, "message": "success", "iss_position": {"latitude": "-25.9092", "longitude": "-27.4817"}}')
(59, '{"timestamp": 1678104803, "message": "success", "iss_position": {"latitude": "-26.6316", "longitude": "-26.7744"}}')
(60, '{"timestamp": 1678104818, "message": "success", "iss_position": {"latitude": "-27.3266", "longitude": "-26.0806"}}')

]:
```

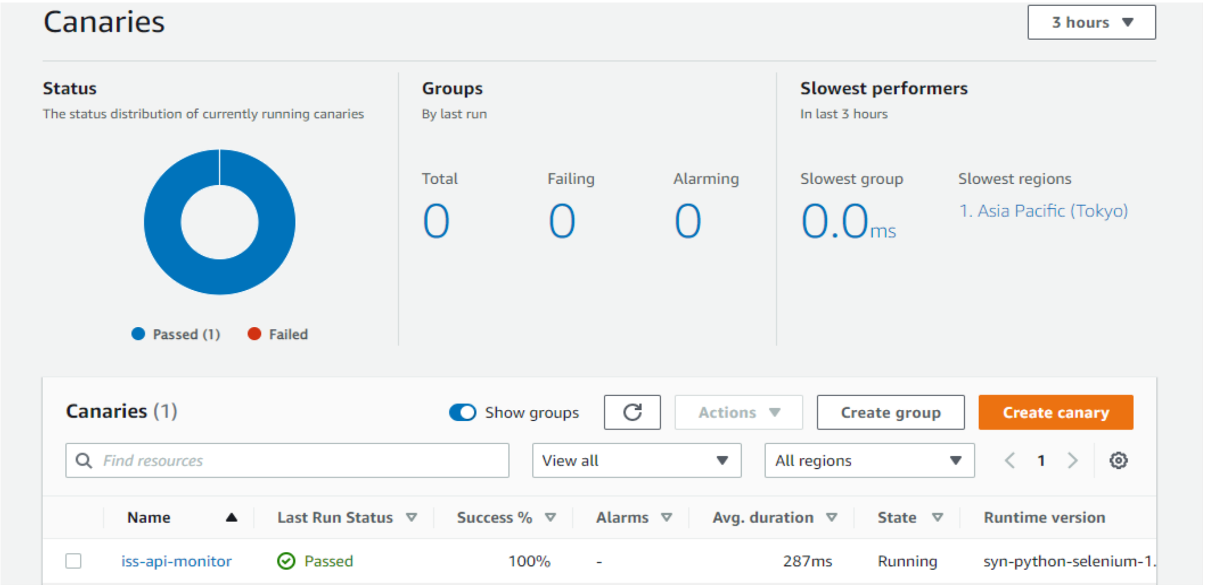
So as of now our project completed up to fetching and storing the data from given API to Postgres RDS for every 15 sec and a period of 15 minutes we can able to run this infinitely also by changing the schedule period.

Next step of the project is to monitor the API server and get the Slack notification for server down.

4. Creation of cloud watch alarm for source server availability:

Monitoring API server:

- For monitoring the server we have to created Synthetics Canaries name iss-api-monitor.



Canaries

Status
The status distribution of currently running canaries

Groups
By last run

Total: 0 | Failing: 0 | Alarming: 0

Slowest performers
In last 3 hours

Slowest group: 0.0ms

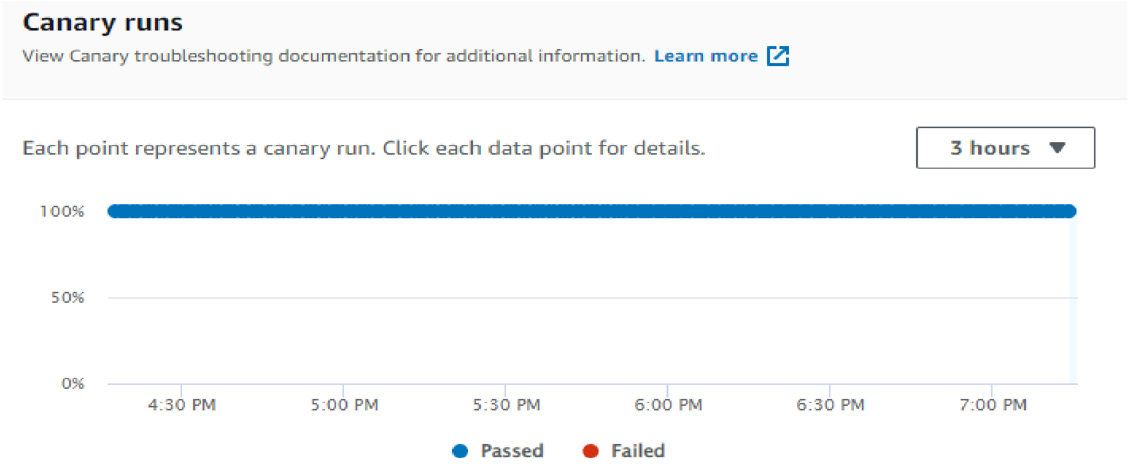
Slowest regions: 1. Asia Pacific (Tokyo)

Canaries (1)

Name	Last Run Status	Success %	Alarms	Avg. duration	State	Runtime version
iss-api-monitor	✓ Passed	100%	-	287ms	Running	syn-python-selenium-1.

This canary runs for 3hrs of period. It will get the metrics data from the given API.

After running the canary for 3hrs success rate metrics has been verified.



Canary runs

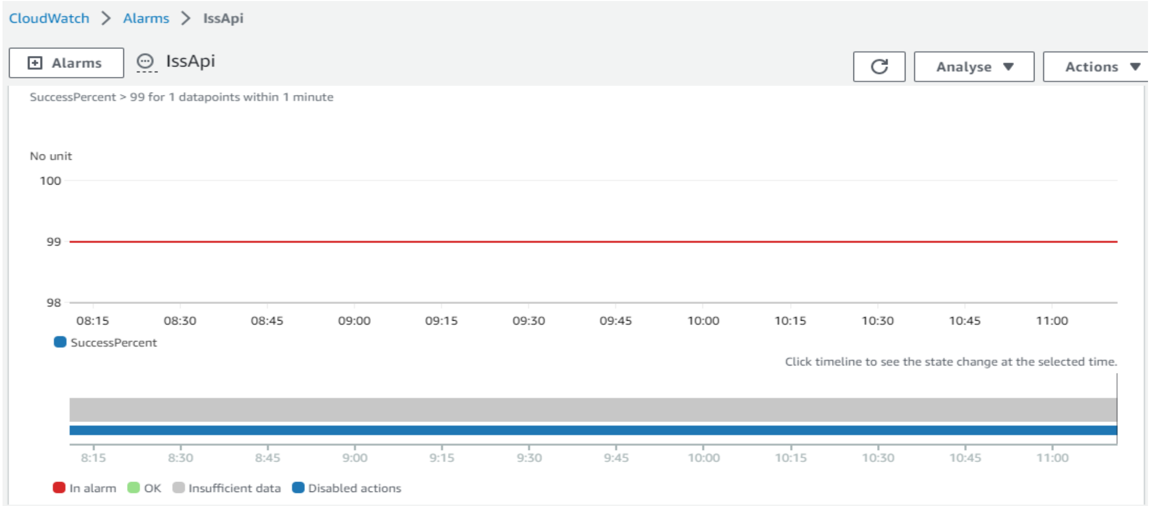
View Canary troubleshooting documentation for additional information. Learn more

Each point represents a canary run. Click each data point for details.

As per above picture we can find that the server running consistently not getting dropped even a single time. 100% success rate we can see.

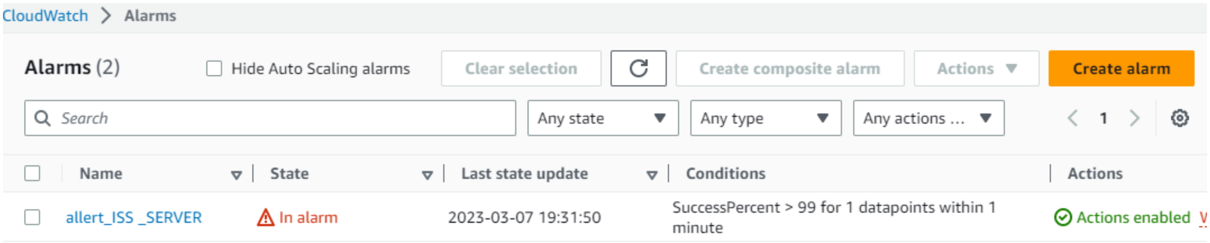
So as per the project statement we cannot do the server down alarm.

For completing the task I have make this alarm for success rate should not exceed 99% then it will goes to directly alarm stage as a result we will get the notification in slack.



CloudWatch > Alarms > IssApi

SuccessPercent > 99 for 1 datapoints within 1 minute



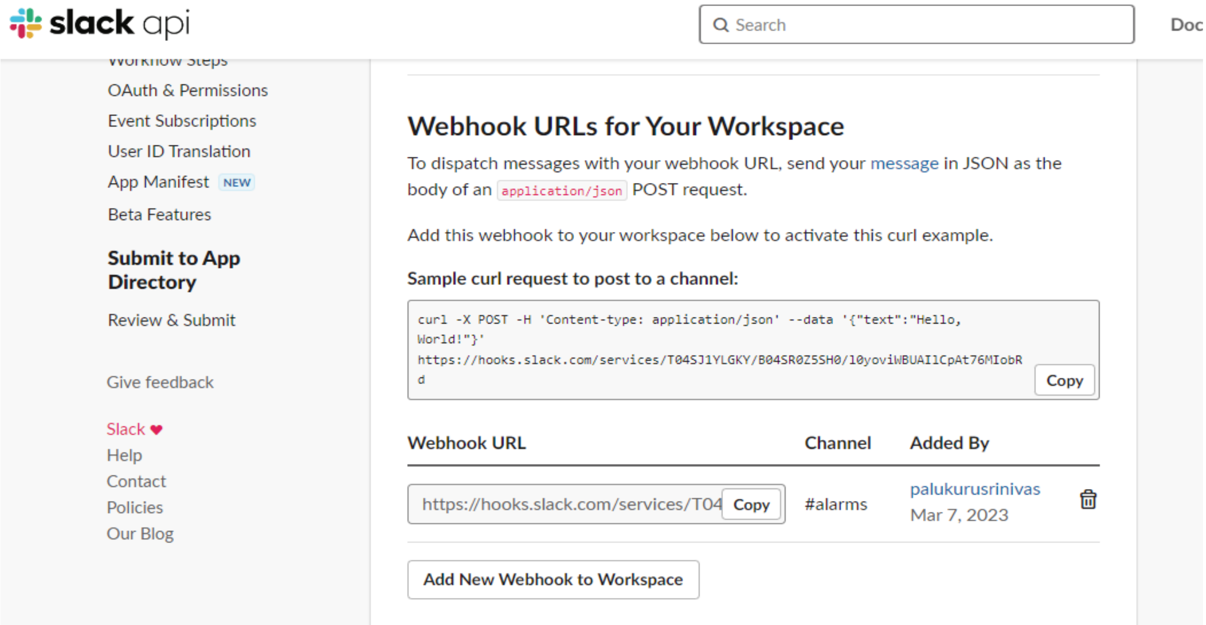
CloudWatch > Alarms

Alarms (2)

□	Name	State	Last state update	Conditions	Actions
□	allert_ISS _SERVER	⚠ In alarm	2023-03-07 19:31:50	SuccessPercent > 99 for 1 datapoints within 1 minute	✓ Actions enabled V

Slack community creation:

Slack api created picture attached below



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.

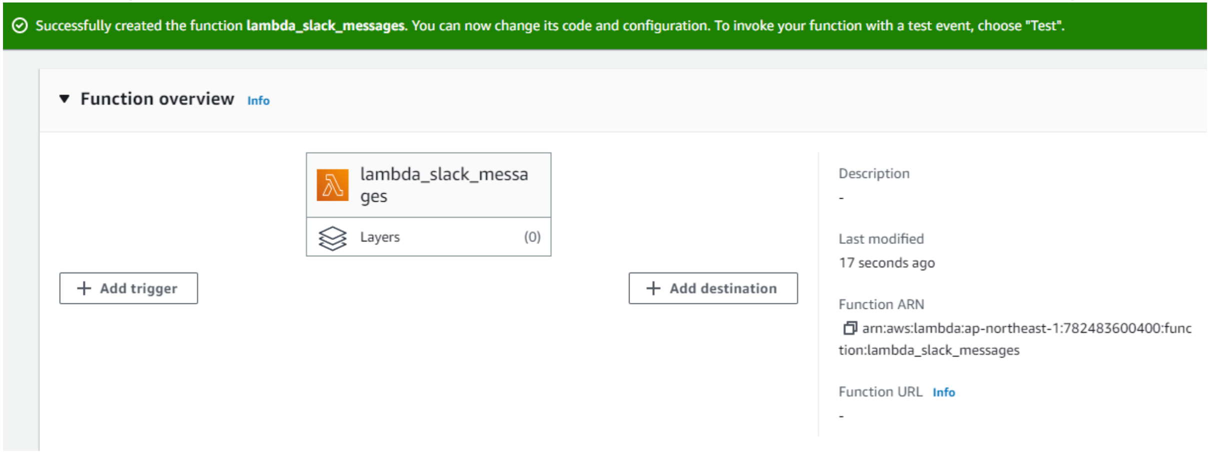
Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello,
World!"}'
https://hooks.slack.com/services/T04SJ1YLGKY/B04SR0Z5SH0/10yoviWBUAI1CpAt76MIobR
d
```

Webhook URL	Channel	Added By
https://hooks.slack.com/services/T04 Copy	#alarms	palukurusrinivas Mar 7, 2023

Created the lambda function to send the slack community:

For sending the server alarm we need to create the lambda function – lambda_slack_messages.



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

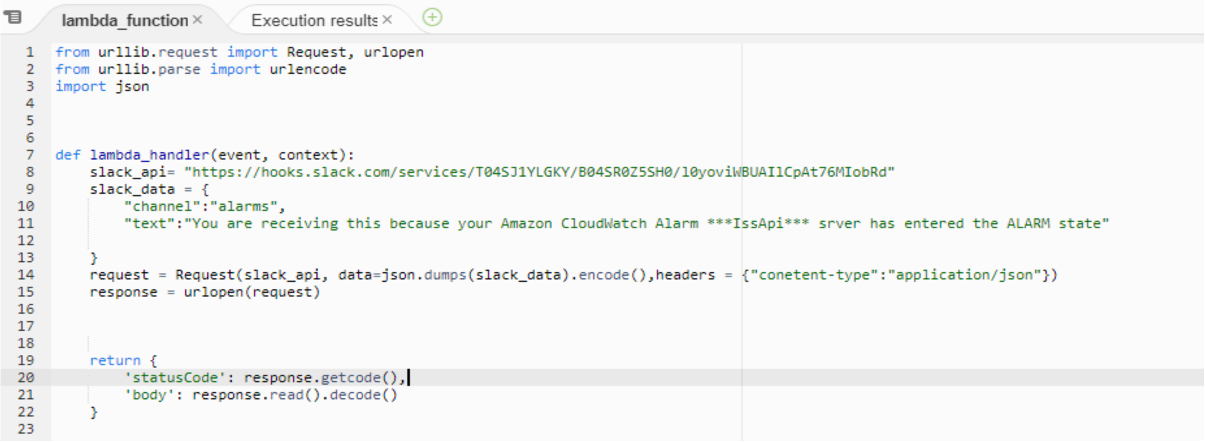
Description
-

Last modified
17 seconds ago

Function ARN
arn:aws:lambda:ap-northeast-1:782483600400:function:lambda_slack_messages

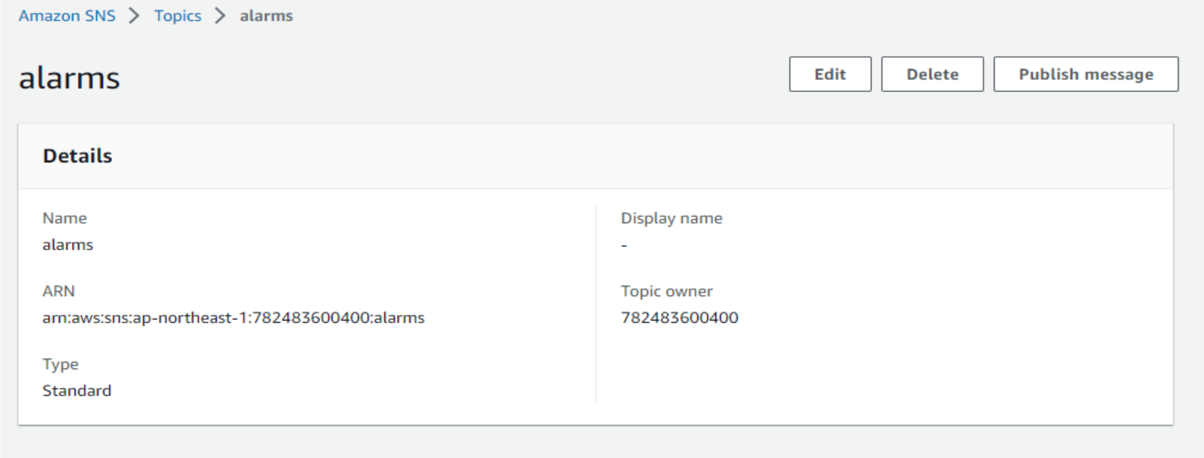
Function URL Info
-

lambda_slack_messages code:



```
from urllib.request import Request, urlopen
from urllib.parse import urlencode
import json



def lambda_handler(event, context):
    slack_api= "https://hooks.slack.com/services/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"
    slack_data = {
        "channel":"alarms",
        "text":"You are receiving this because your Amazon CloudWatch Alarm ***IssApi*** srver has entered the ALARM state"

    }
    request = Request(slack_api, data=json.dumps(slack_data).encode(),headers = {"conetent-type":"application/json"})
    response = urlopen(request)



    return {
        'statusCode': response.getcode(),
        'body': response.read().decode()
    }

```

Creating SNS topic to invoke our lambda function, Topic name: alarms



alarms

Details

Name
alarms

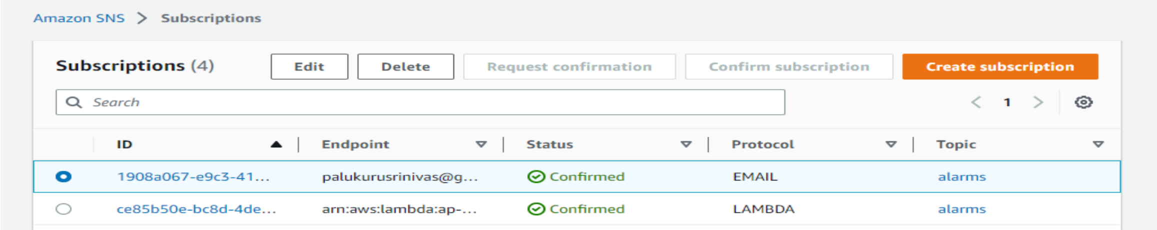
ARN
arn:aws:sns:ap-northeast-1:782483600400:alarms

Type
Standard

Display name
-

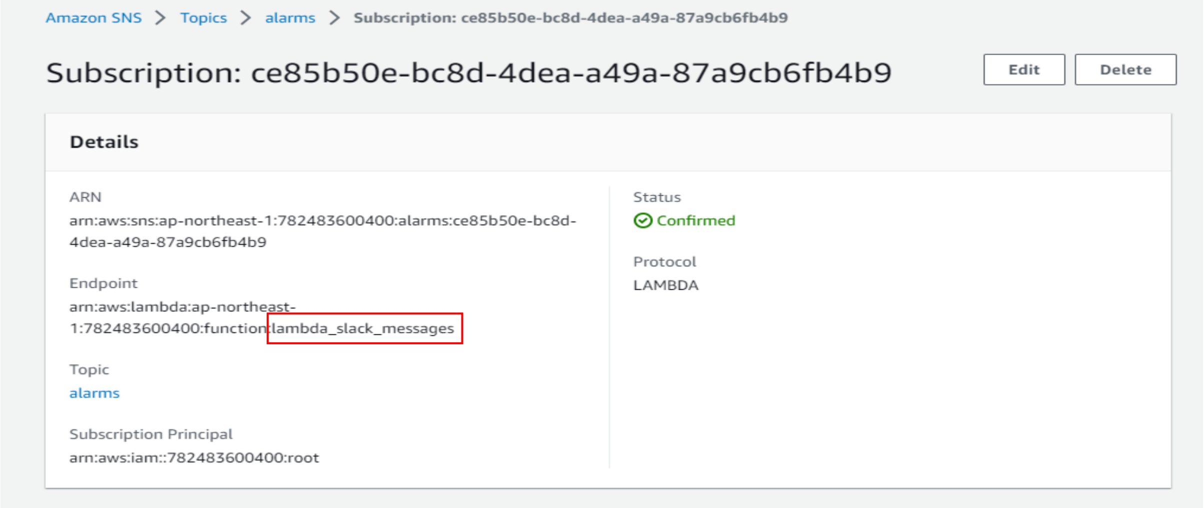
Topic owner
782483600400

SNS subscriptions: both the mail and lambda unction subscribed for sns .



Subscriptions (4)

	ID	Endpoint	Status	Protocol	Topic
●	1908a067-e9c3-41…	palukurusrinivas@g…	✓ Confirmed	EMAIL	alarms
○	ce85b50e-bc8d-4de…	arn:aws:lambda:ap-…	✓ Confirmed	LAMBDA	alarms


Amazon SNS > Topics > alarms > Subscription: ce85b50e-bc8d-4dea-a49a-87a9cb6fb4b9
Edit
Delete

Subscription: ce85b50e-bc8d-4dea-a49a-87a9cb6fb4b9

Details

ARN
arn:aws:sns:ap-northeast-1:782483600400:alarms:ce85b50e-bc8d-4dea-a49a-87a9cb6fb4b9

Endpoint
arn:aws:lambda:ap-northeast-1:782483600400:function:lambda_slack_messages

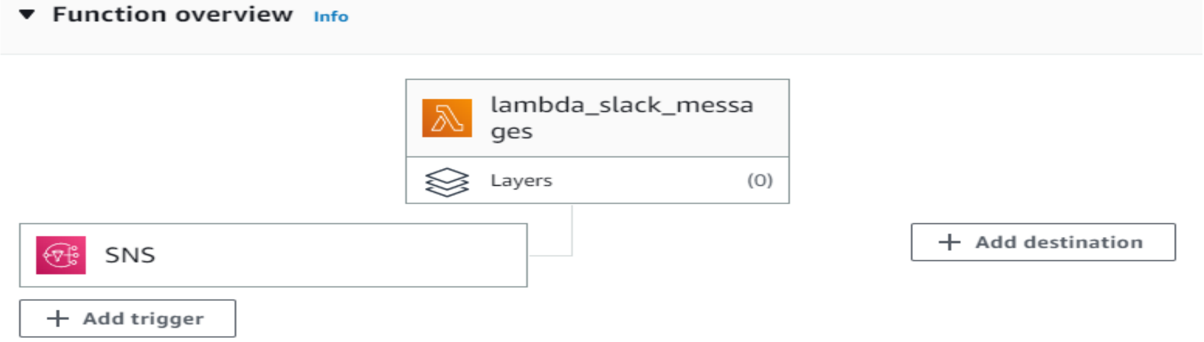
Topic
alarms

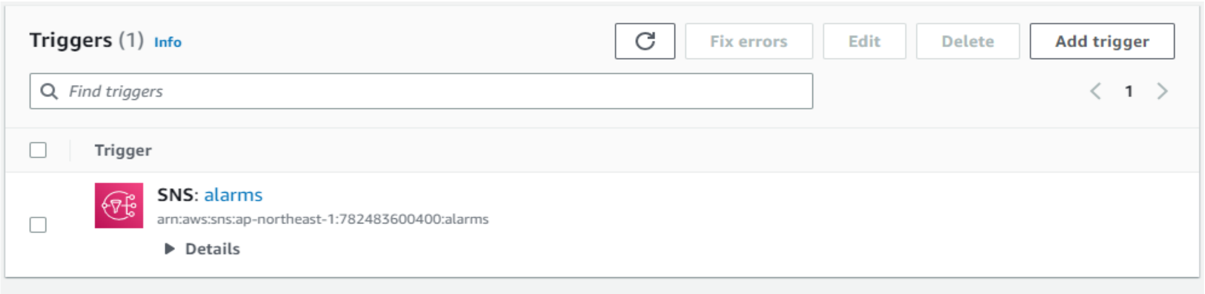
Subscription Principal
arn:aws:iam::782483600400:root

Status
✓ Confirmed

Protocol
LAMBDA

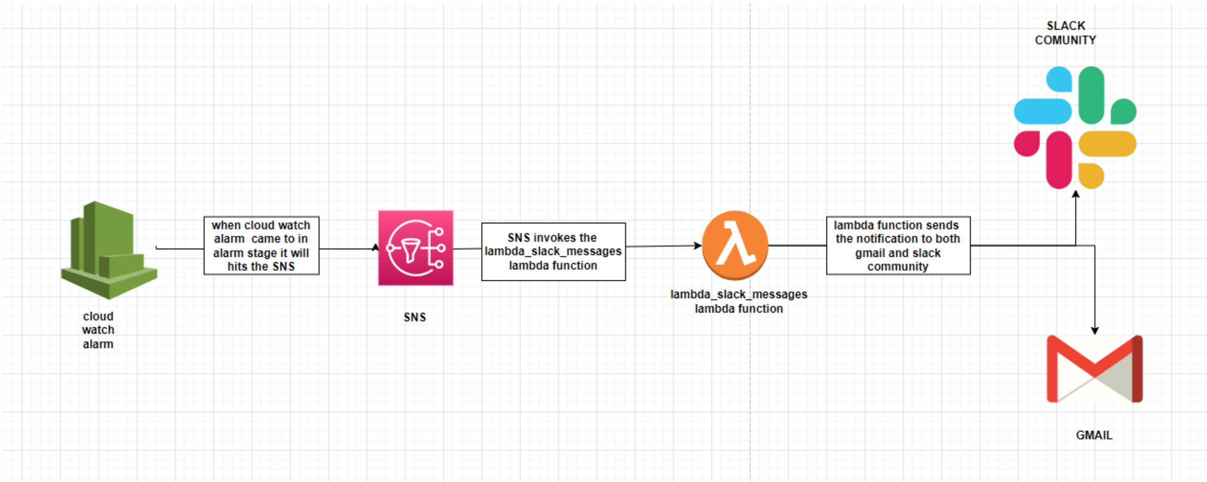
Added the SNS as trigger for “lambda_slack_messages” lambda function to send the slack community.



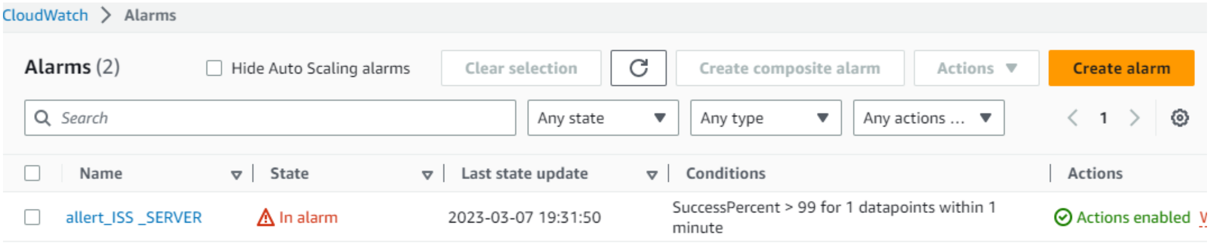


□	Trigger
□	SNS: alarms arn:aws:sns:ap-northeast-1:782483600400:alarms

So after canary metrics went to alarm state alarm will hits the SNS topic “alarms”, this sns topic will triggers ------> “lambda_slack_messages “ lambda function as a results we will get the notification in Gmail and slack community .

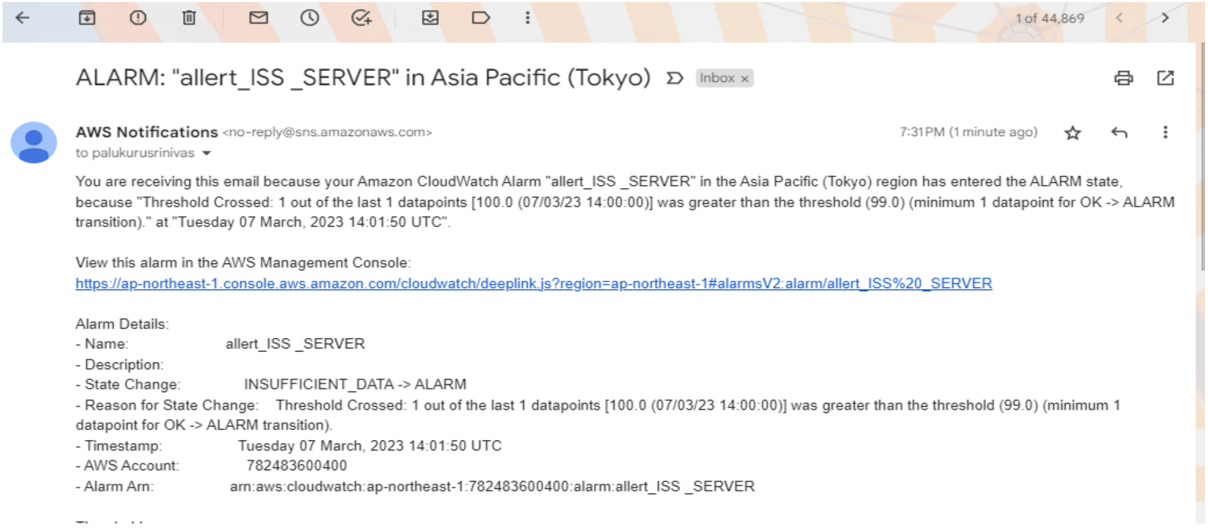

cloud
watch
alarm
when cloud watch alarm came to in alarm stage it will hits the SNS
SNS
SNS invokes the lambda_slack_messages lambda function
lambda_slack_messages lambda function
lambda function sends the notification to both gmail and slack community
SLACK COMUNITY
GMAIL

Metrics came to in alarm stage:



□	Name	State	Last state update	Conditions	Actions
□	allert_ISS _SERVER	⚠ In alarm	2023-03-07 19:31:50	SuccessPercent > 99 for 1 datapoints within 1 minute	✓ Actions enabled

Alarm notification to Gmail:



ALARM: "allert_ISS _SERVER" in Asia Pacific (Tokyo)

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

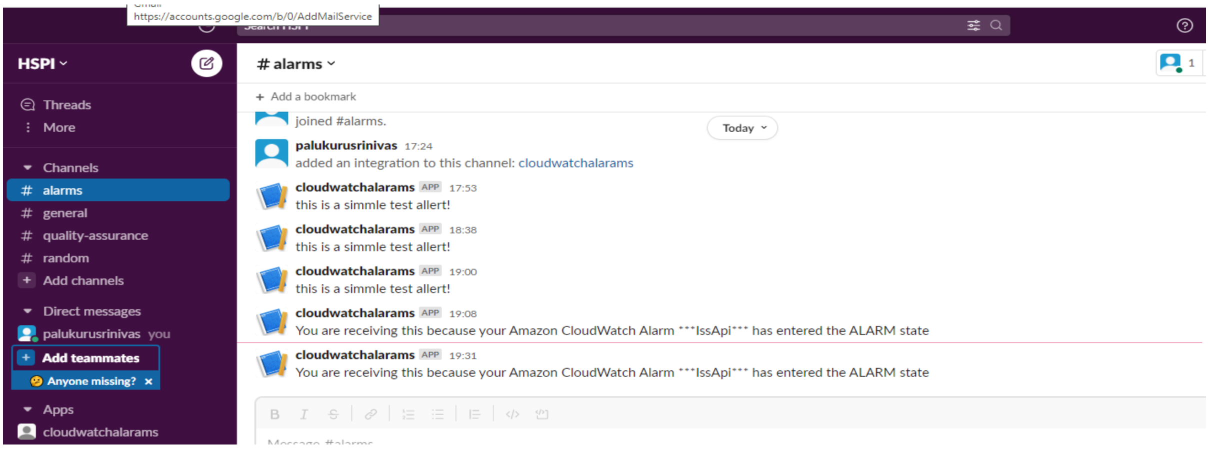
7:31PM (1 minute ago)

You are receiving this email because your Amazon CloudWatch Alarm "allert_ISS _SERVER" in the Asia Pacific (Tokyo) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [100.0 (07/03/23 14:00:00)] was greater than the threshold (99.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Tuesday 07 March, 2023 14:01:50 UTC".

View this alarm in the AWS Management Console:
https://ap-northeast-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=ap-northeast-1#alarmsV2:alarm/allert_ISS%20_SERVER

Alarm Details:
- Name: allert_ISS _SERVER
- Description:
- State Change: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [100.0 (07/03/23 14:00:00)] was greater than the threshold (99.0) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Tuesday 07 March, 2023 14:01:50 UTC
- AWS Account: 782483600400
- Alarm Arn: arn:aws:cloudwatch:ap-northeast-1:782483600400:alarm:allert_ISS _SERVER

Alarm notification to Slack community:



joined #alarms.

palukurusrinivas 17:24
added an integration to this channel: cloudwatchalarams

cloudwatchalarams APP 17:53
this is a simmle test allert!

cloudwatchalarams APP 18:38
this is a simmle test allert!

cloudwatchalarams APP 19:00
this is a simmle test allert!

cloudwatchalarams APP 19:08
You are receiving this because your Amazon CloudWatch Alarm ***IssApi*** has entered the ALARM state

cloudwatchalarams APP 19:31
You are receiving this because your Amazon CloudWatch Alarm ***IssApi*** has entered the ALARM state

CONCLUSION:

As per project statement we have fetched the data from the give API and stored in Postgres SQL RDS, we have fetched the data for every 15sec by using sub minute invocation. We have monitored API server availability and make an alarm notification to slack community using slack API.

Task completed.