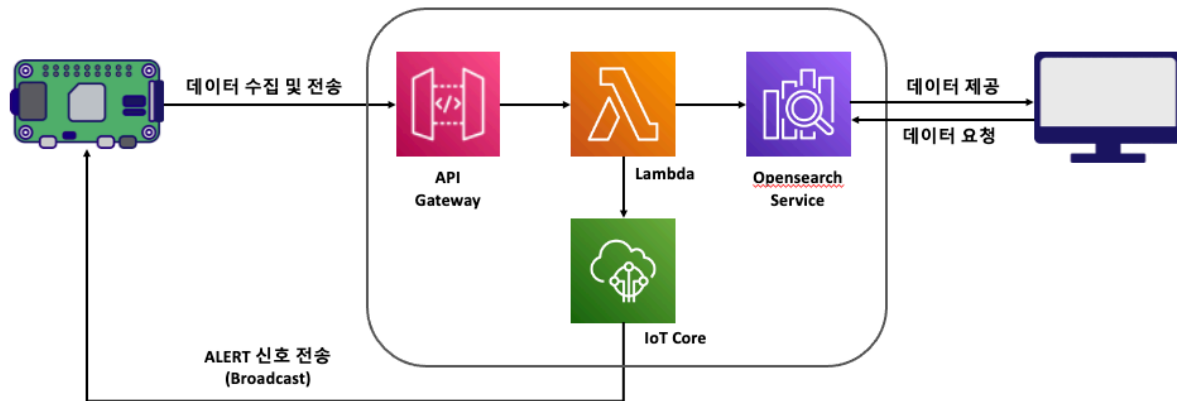


요약 다이어그램



AWS 서비스 소개

API Gateway

Amazon API Gateway는 규모와 관계없이 REST 및 WebSocket API를 생성, 게시, 유지, 모니터링 및 보호하기 위한 AWS 서비스입니다. API 개발자는 AWS 또는 다른 웹 서비스를 비롯해 AWS 클라우드에 저장된 데이터에 액세스하는 API를 생성할 수 있습니다. API Gateway API 개발자는 자체 클라이언트 애플리케이션에서 사용할 API를 생성할 수 있습니다. API Gateway는 최대 수십만 개의 동시 API 호출 허용 및 처리에 관련된 모든 작업을 다룹니다. 여기에는 트래픽 관리, 권한 부여 및 액세스 제어, 모니터링, API 버전 관리가 포함됩니다. AWS Lambda에서 실행 중인 코드, 웹 애플리케이션, 실시간 통신 애플리케이션과 같은 백엔드 서비스에서 데이터, 비즈니스 로직 또는 기능에 액세스할 수 있게 해주는 "gateway" 역할을 합니다. 앱에서 공개적으로 사용할 수 있는 AWS 서비스를 호출하려면 Lambda를 사용해 필요한 서비스와 상호 작용하고, API Gateway에서 API 메서드를 통해 Lambda 함수를 제공할 수 있습니다. AWS Lambda은 고가용성 컴퓨팅 인프라에서 코드를 실행하며 컴퓨팅 리소스의 필요한 실행 및 관리를 수행합니다.

Lambda

Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 서버리스 컴퓨팅 서비스입니다. 서버리스란, 개발자가 서버의 존재를 신경 쓸 필요가 없다는 뜻입니다. 서버가 필요 없이 AWS 서버가 알아서 처리해주시니 개발자는 요청받고 처리하는 서버에 대한 걱정없이 코드를 실행하고 사용한 컴퓨팅 시간에 대해서만 비용을 지불합니다. Lambda는 고가용성 컴퓨팅 인프라에서 코드를 실행하고 서버 및 운영 체제 유지 관리, 용량 프로비저닝 및 자동 확장, 코드 모니터링 및 로깅을 포함한 컴퓨팅 리소스의 모든 관리를 수행합니다. Lambda를 사용하면 거의 모든 유형의 애플리케이션 또는 백엔드 서비스에 대한 코드를 실행할 수 있습니다. API Gateway를 사용하여 HTTP 요청을 Lambda 함수로 라우팅하는 웹 API에 대한 안전하고 확장 가능한 게이트웨이를 제공합니다.

Opensearch service(전 Elasticsearch service)

(참고) 본래는 Elasticsearch service 이지만 2021년 1월 21일, Elastic NV는 소프트웨어 라이선스 전략을 변경하는 바, 퍼미시브 ALv2 라이선스 하에서 Elasticsearch 및 Kibana의 새로운 버전을 더 이상 릴리스하지 않는다고 발표했습니다. 때문에 이후에는 opensearch service라는 이름으로 서비스를 합니다.

배포형 오픈 소스 검색과 분석 제품군으로 실시간 애플리케이션 모니터링, 로그 분석 및 웹 사이트 검색과 같이 다양한 사용 사례에 사용됩니다. OpenSearch는 데이터 탐색을 쉽게 도와주는 통합 시각화 도구와 함께 확장성을 지닌 시스템을 제공하여 대량 데이터 볼륨에 빠르게 액세스가 가능합니다.

IoT Core

AWS IoT Core 서비스는 IoT 디바이스를 AWS IoT 서비스 및 기타 AWS 서비스에 연결합니다. AWS IoT Core에는 IoT 디바이스와 클라우드 간에 메시지를 연결하고 처리하는 디바이스 게이트웨이와 메시지 브로커가 포함됩니다.

실행 플로우

1. 라즈베리파이에서 데이터 값을 수집한 후 pub.py 파일을 실행시켜 api gateway의 엔드포인트로 전송을 합니다.



Payload 변수에 수집한 데이터 값을 넣어 준 후, 각 센서에 맞는 api gateway 엔드포인트로 데이터를 전송해줍니다.

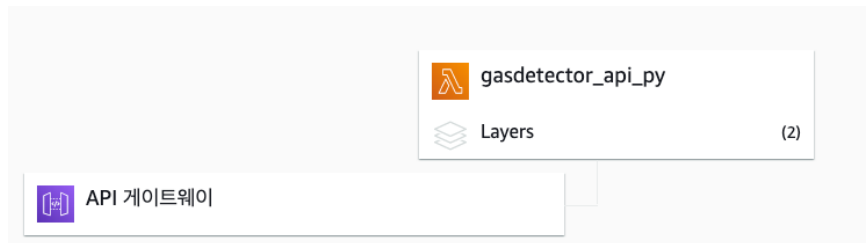
```
#set payload
payload = { ...
}
payload = dumps(payload)

#post payload
r = requests.post('https://[redacted]
                  headers=headers, data=payload)
```

pub.py 파일 코드 중 일부

2. api gateway 로 데이터를 전송하면, 해당 api gateway 와 연결되어있는 lambda 에서 들어온 데이터 값을 처리하게 됩니다.

gasdetect 값을 처리하기 위해 /gasdetector 에 해당하는 api gateway 로 보냈을 때 아래 사진과 같이 연결되어있는 gasdetector_api_py 라는 람다의 개요를 보여주는 화면입니다.



이 때, Lambda 에서 해주는 역할은 두가지 입니다.

1. lot core로 publish 요청하기
2. Opensearch(전 Elasticsearch) service를 위한 데이터 전송

2-1. lambda 함수에서 들어온 값에 따라 lot core 로 publish 를 요청하게 됩니다.

데이터 전송 실패/성공을 알리는 메세지 뿐 만 아니라, 들어온 값이 lambda 함수에서 설정해둔 기준치를 넘으면 보내는 warning 메세지도 같이 설정이 되어있습니다.

```
response = client.publish(  
    topic = 'pi/1',  
    qos = 1,  
    payload = json.dumps({"msg": msg})  
)
```

'gasdetector_api_py' lambda 함수 중 일부 - 데이터가 제대로 들어온지를 판단하여 msg 변수에 값을 담은 후, publish 를 요청하는 코드입니다.

```
if("no2" in event) :  
    value_no2 = float(event['no2']['value'])  
    if(value_no2 >= 88) :  
        response = client.publish(  
            topic = 'pi/1',  
            qos = 1,  
            payload = json.dumps({"warning": "no2 warning"})  
        )
```

'gasdetector_api_py' lambda 함수 중 일부 - 들어온 no2 값이 특정 기준치인 88 을 넘으면 warning: no2 warning 라는 publish 를 요청하는 코드입니다.

2-2. lambda 함수에서 들어온 값을 Opensearch (전 Elasticsearch) service 를 위하여 값을 저장합니다.

data 라는 변수에 들어온 값과 들어온 시간을 저장하여 해당 엔드포인트로 값을 전송해줍니다.

```
data = event
data['date'] = nowDatetime
headers = {'Content-type': 'application/json'}
response = requests.post(url, data = json.dumps(data), headers=headers).json()
```

'gasdetector_api_py' lambda 함수 중 일부 - 들어온 값을 data 라는 변수로 처리하여 대량 데이터 볼륨으로 데이터를 전송함.

Opensearch service 에서 쿼리문을 통해 검색을 시도해보면, 밑 화면과 같이 데이터가 제대로 들어가있는 것을 확인할 수 있습니다.

```
{
  "_index" : "gasdetect",
  "_type" : "_doc",
  "_id" : [REDACTED],
  "_score" : 1.0,
  "_source" : {
    "groupid" : "2",
    "gps" : "37.61002501514269, 126.99636868892937",
    "co" : {
      "sensorid" : "0005",
      "sensortype" : "8",
      "value" : "26.20",
      "createdtime" : "2021-09-03 18:14:49"
    },
    "no2" : {
      "sensorid" : "0006",
      "sensortype" : "9",
      "value" : "52.00",
      "createdtime" : "2021-09-03 18:14:49"
    },
    "c2h5oh" : {
      "sensorid" : "0007",
      "sensortype" : "27",
      "value" : "26.11",
      "createdtime" : "2021-09-03 18:14:49"
    },
    "h2" : {
      "sensorid" : "0008",
      "sensortype" : "30",
      "value" : "400",
      "createdtime" : "2021-09-03 18:14:49"
    }
  },
}
```

3. lot core 에서, 2-1 과 같이 요청 받은 publish 를 구독(subscribe)중인 기기에게 브로드캐스트로 메시지를 전송하게 됩니다.

pi/1

▼ pi/1

```
{
  "msg": "success"
}
```

▼ pi/1

```
{
  "warning": "co warning"
}
```

이 때, 구독을 원하는 라즈베리파이에서 sub.py 파일을 실행시키면 publish 를 보낸 'pi/1' 주소로 구독이 가능합니다. 구독을 하게 되면 iot core 에서 'pi/1' 주소를 구독하는 모든 라즈베리파이에 브로드캐스트로 전송한 메시지를 받을 수 있습니다.

```
def on_connect(mqttc, obj, flags, rc):
    if rc == 0:
        #alert connect
        print('connected!!')
        #subscribe
        mqttc.subscribe(sub, qos=1)

#connect
mqtt_client.connect(ENDPOINT, port=8883)
mqtt_client.loop_forever() # threaded network loop
```

sub.py 파일 코드 중 일부 - 'pi/1' 로 구독을 하고 iot_core 의 엔드포인트로 연결하여 메시지를 전달 받게 합니다.

4. 2-2 에서 저장한 데이터를 사용하여 쿼리문을 통해 Opensearch service 의 데이터를 검색합니다. 이를 웹사이트를 사용해 시각화 시킵니다.

```
"query" : {
  "bool": {
    "must": [
      {"match": {"_index": selectType}},
      {"match": {"groupid": groupId}}
    ]
  }
}
```

웹사이트 코드 중 일부 - 데이터 검색을 위한 query 문을 작성.



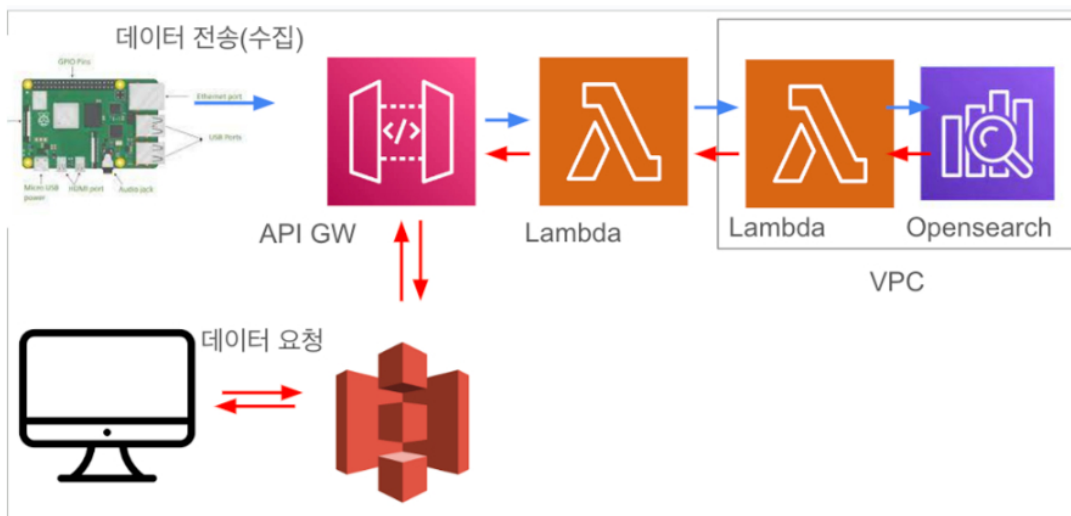
요청된 데이터의 시각화 된 웹사이트 화면

센서플랫폼VPC내용추가

보안강화를 위한 VPC설정 및 코드개선, 결과화면

기존 Opensearch는 vpc로 설정되어 있지않아 보안성이 떨어짐
이를 해결하고자 vpc를 적용하기로하였으며, 그 과정과 결과를 다룬다.

VPC를 통해서는 오직 API GW를 통해서만 람다, 오픈서치를 이용할 수 있도록 구성된다.



- VPC

아래와 같이 구성하였다.

Your VPCs (1/1) [Info](#)

Name : `crc-sensorplatform-vpc` ✕
Clear filters

< 1 >

<input checked="" type="checkbox"/>	Name	VPC ID	State	IPv4 CIDR	IPv6
<input checked="" type="checkbox"/>	crc-sensorplatform-vpc	vpc-0fdd6427e81be1195	Available	172.0.0.0/24	-

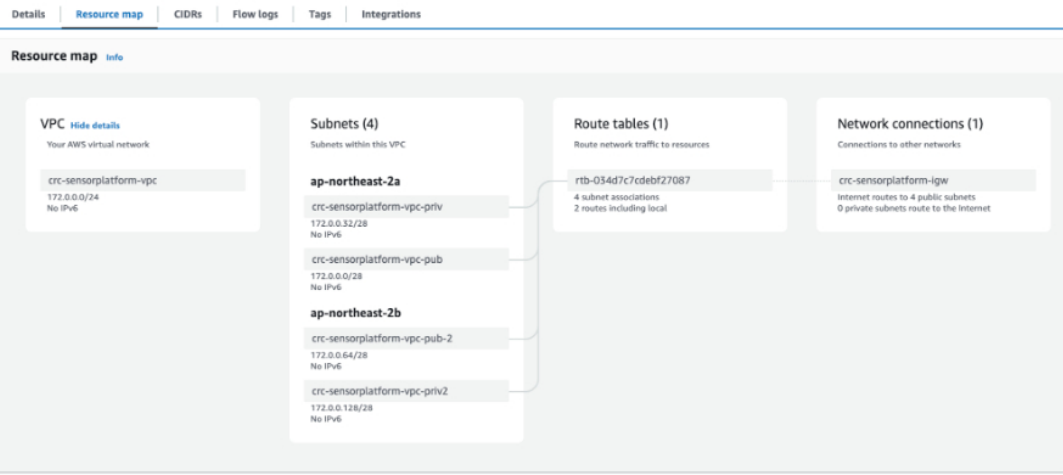
vpc-0fdd6427e81be1195 / crc-sensorplatform-vpc

[Details](#)
[Resource map](#)
[CIDRs](#)
[Flow logs](#)
[Tags](#)
[Integrations](#)

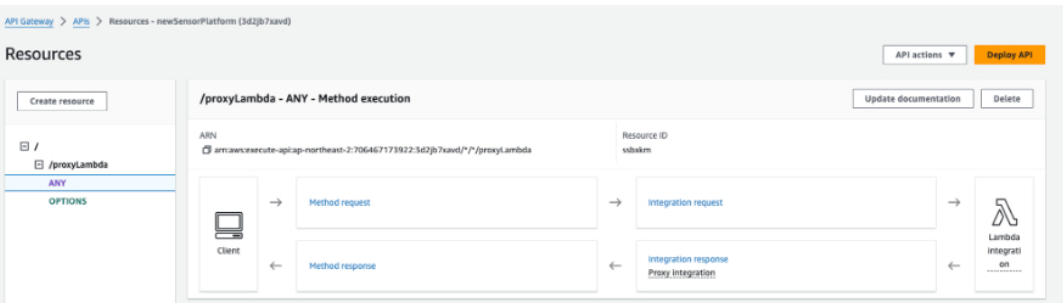
Details

VPC ID vpc-Ofdd6427e81be1195	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-b3a704d8	Main route table rtb-034d7c7cdebf27087	Main network ACL acl-0937b3241ded7c7a0
Default VPC No	IPv4 CIDR 172.0.0.0/24	IPv6 pool -	IPv6 CIDR (Network border group) -
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID 706467173922	

vpc-Ofdd6427e81be1195 / crc-sensorplatform-vpc



- API GW 구성화면이다.



ANY를 통해서 POST와 GET의 요청을 통합실행하는 방법을 택했다.

다음은 해당 GW와 연결된 람다화면이다.

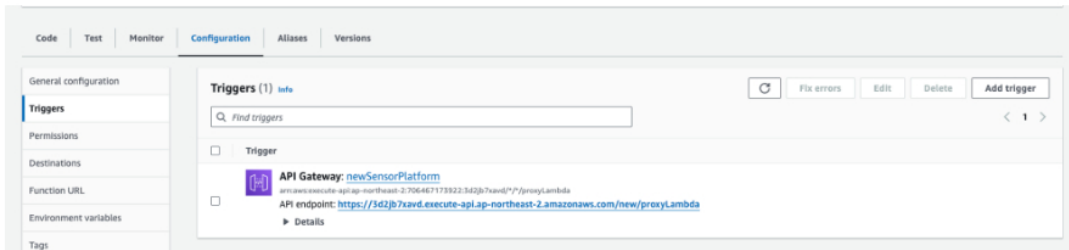
- Lamdba (외부)

The Lambda console shows the configuration for the proxyLambda function. The function overview includes a diagram showing the function triggered by an API Gateway. The code source section shows the following code:

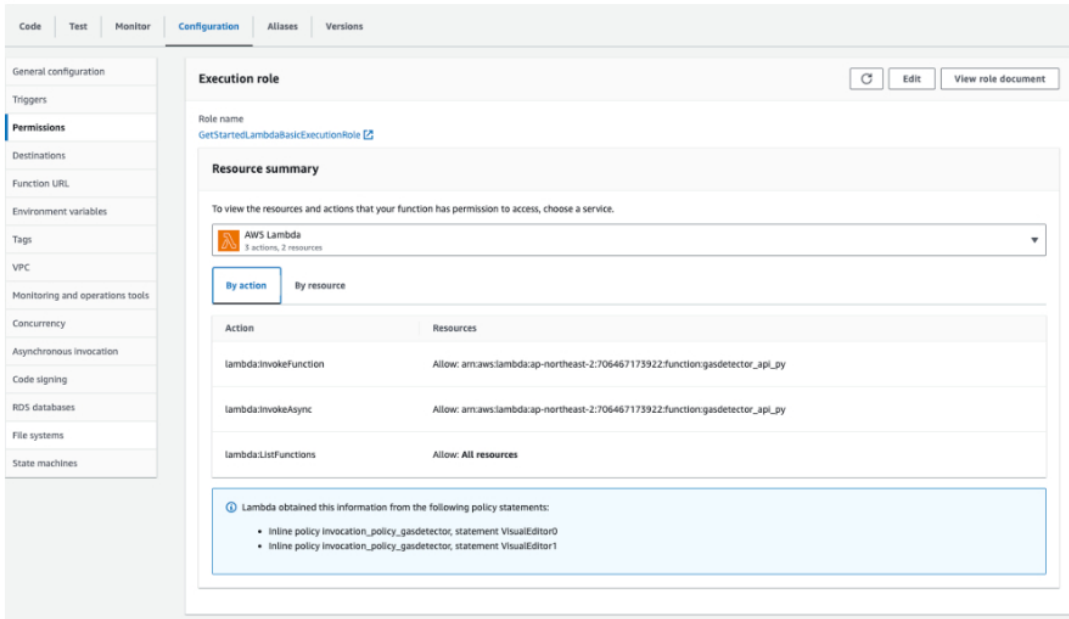
```

1 import boto3, json
2 client = boto3.client('lambda', region_name='ap-northeast-2')
3
4 def invokeFunction(payload):
5     p = json.loads(payload)
6     function_name = 'proxyLambda'
7     response = client.invoke(FunctionName=function_name, Payload=json.dumps(p))
8     return response['Payload']
  
```


다이어그램에서 보는 것과 같이 API GW와 연동되어있음을 확인할 수 있다.



configuration > Triggers 를 통해서도 확인할 수도 있다.



또한 이 람다함수는 오픈서치

람다함수의 코드이며

[<https://github.com/kmucrc/sensorplatform/blob/main/2022/aws/lambda/proxy-lambda-function.py>] 에서도 확인할 수 있다.

```
import boto3, json
client = boto3.client('lambda', region_name="ap-northeast-2")

def invokePrivateLambda(payload):
    r = client.invoke(
        FunctionName='gasdetector_api_py',
        InvocationType='RequestResponse',
        Payload=bytes(payload, encoding='utf-8')
    )
    return r['Payload'].read().decode('utf-8')

def lambda_handler(event, context):
    method = event['httpMethod']

    if method == 'POST':
        body = event['body']
        index = event['headers']['index']
        payload = json.dumps({'body': body, 'method': method, 'index': index})
        response = invokePrivateLambda(payload)

    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
```

```

        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
    },
    'body': response
}

elif method == 'GET':
    index = event["queryStringParameters"]['index']
    query = event["queryStringParameters"]['query']
    payload = json.dumps({'query':query, 'method': "GET", 'index': index })
    response = invokePrivateLambda(payload)

    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': response
    }

return {
    'statusCode': 400,
    'body': json.dumps('Invalid_Access')
}

```

- Lambda(VPC, 내부)

gasdetector_api_py

Throttle Copy ARN Actions

Function overview Info

Export to Application Composer Download

Diagram Template

gasdetector_api_py

Layers (2)

+ Add trigger + Add destination

Description

Last modified last year

Function ARN

arn:aws:lambda:ap-northeast-2:706467173:922:function:gasdetector_api_py

Function URL Info

Code Test Monitor Configuration Aliases Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

Monitoring and operations tools

Concurrency

Asynchronous invocation

VPC Info Edit

VPC

vpc-0fdd6427e81be1195 (172.0.0.0/24) | crc-sensorplatform-vpc

Subnets

- Allow IPv6 traffic = false
- subnet-00ad5d06c9b6b7c80 (172.0.0.32/28) | ap-northeast-2a, crc-sensorplatform-vpc-priv

Security groups

- sg-07bb9915e04e0fcf0 (crc-sensorplatform-sg)

Inbound rules Outbound rules

Security group ID	Protocol	Ports	Source
sg-07bb9915e04e0fcf0	All	All	sg-0062ae26c6c9e92ec

오픈서치서비스와 연결되는 람다함수는 위와 같이 VPC설정을 한다.

코드는 아래와 같으며, 링크는

(https://github.com/kmucrc/sensorplatform/blob/main/2022/aws/lambda/gasdetector_api_py.py)이다.

```
from pytz import timezone
import constants
from urllib import parse

client = boto3.client(
    'iot-data',
    region_name = 'ap-northeast-2',
    endpoint_url = 'https://a26pnpi3qj130t-ats.iot.ap-northeast-2.amazonaws.com'

def publish_to_iot_core(data):
    now = datetime.datetime.now(timezone('Asia/Seoul'))
    nowDatetime = now.strftime('%Y-%m-%d %H:%M:%S')
    data['date'] = nowDatetime;

    return data;

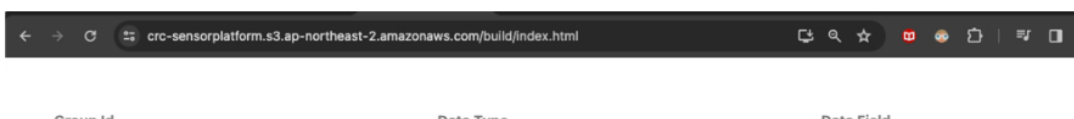
def lambda_handler(event, context):
    print
    headers = {
        # 'authorization': 'Basic a211Y3JjZGV2X29zOiFLbXVjcmMyMDIy',
        'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
        'Content-Type': 'application/json'
    };
    auth = HTTPBasicAuth('오픈서치아이디', '오픈서치비밀번호');

    if event['method'] == "GET":
        url = 'https://vpc-crc-sensorplatform-os-ncsx6wiaesylldgi3q462uvmaeq.ap-northeast-2.amazonaws.com/build/index.html'
        r = requests.get(url, headers=headers, auth=auth, data=event['query'])
        result = r.text

    if event['method'] == "POST":
        url = 'https://vpc-crc-sensorplatform-os-ncsx6wiaesylldgi3q462uvmaeq.ap-northeast-2.amazonaws.com/build/index.html'
        data = json.loads(event["body"]);
        publish_to_iot_core(data);
        result = requests.post(url = url, data = json.dumps(data), headers = headers)

    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': result
    }
```

데이터요청 결과화면



Group No

1

Device Type

oximetry

Device Name

HbA1c(당화혈색소)

Created Time

Start Date

2023-11-01

Start Time

03:01 AM

End Date

2024-02-29

End Time

03:01 AM

Search

