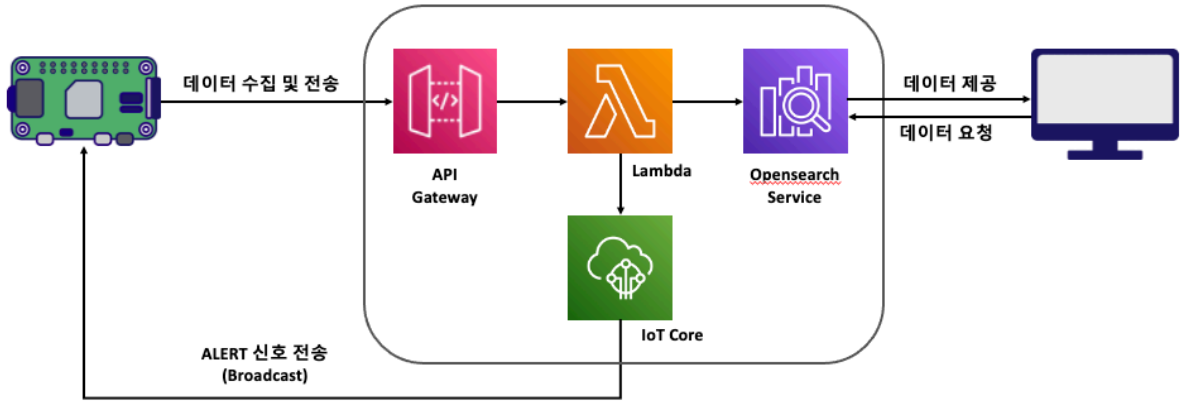
요약 다이어그램



AWS 서비스 소개

API Gateway

Amazon API Gateway 는 규모와 관계없이 REST 및 WebSocket API 를 생성, 게시, 유지, 모니터링 및 보호하기 위한 AWS 서비스입니다. API 개발자는 AWS 또는 다른 웹 서비스를 비롯해 AWS 클라우드에 저장된 데이터에 액세스하는 API 를 생성할 수 있습니다. API Gateway API 개발자는 자체 클라이언트 애플리케이션에서 사용할 API 를 생성할 수 있습니다. API Gateway 는 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 Lambda 에서 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 Lambda 를 사용해 필요한 서비스와 상호 작용하고, API Gateway 에서 API 메서드를 통해 Lambda 함수를 제공할 수 있습니다. AWS Lambda 은 고가용성 컴퓨팅 인프라에서 코드를 실행하며 컴퓨팅 리소스의 필요한 실행 및 관리를 수행합니다.

Lambda

Lambda 는 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 처리해주니 개발자는 요청받고 처리하는 서버에 대한 걱정없이 코드를 실행하고 사용한 컴퓨팅 시간에 대해서만 비용을 지불합니다.
Lambda 는 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. Lambda 를 사용하면 거의 모든 유형의 애플리케이션 또는 백엔드 서비스에 대한 코드를 실행할 수 있습니다. API Gateway 를 사용하여 HTTP 요청을 Lambda 함수로 라우팅하는 웹 API 에 대한 안전하고 확장 가능한 게이트웨이를 제공합니다.

Opensearch service(전 Elasticsearch service)

(참고) 본래는 Elasticsearch service 이지만 2021 년 1 월 21 일, Elastic NV 는 소프트웨어 라이선스 전략을 변경하는 바, 퍼미시브 ALv2 라이선스 하에서 Elasticsearch 및 Kibana 의 새로운 버전을 더 이상 릴리스하지 않는다고 발표했습니다. 때문에 이후에는 opensearch service 라는 이름으로 서비스를 합니다.
배포형 오픈 소스 검색과 분석 제품군으로 실시간 애플리케이션 모니터링, 로그 분석 및 웹 사이트 검색과 같이 다양한 사용 사례에 사용됩니다. OpenSearch 는 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 Core 에는 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 gateway 의 엔드포인트로 전송을 합니다.

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

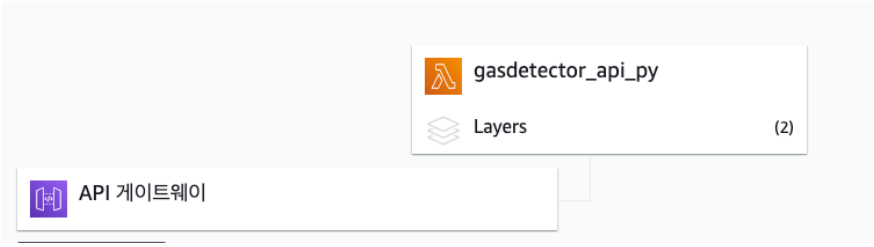
```
#set payload
payload = {…
}
payload = dumps(payload)

#post payload
r = requests.post('https://
                  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. Iot core로 publish 요청하기
2. Opensearch(전 Elasticsearch) service를 위한 데이터 전송

2-1. lambda 함수에서 들어온 값에 따라 Iot 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.

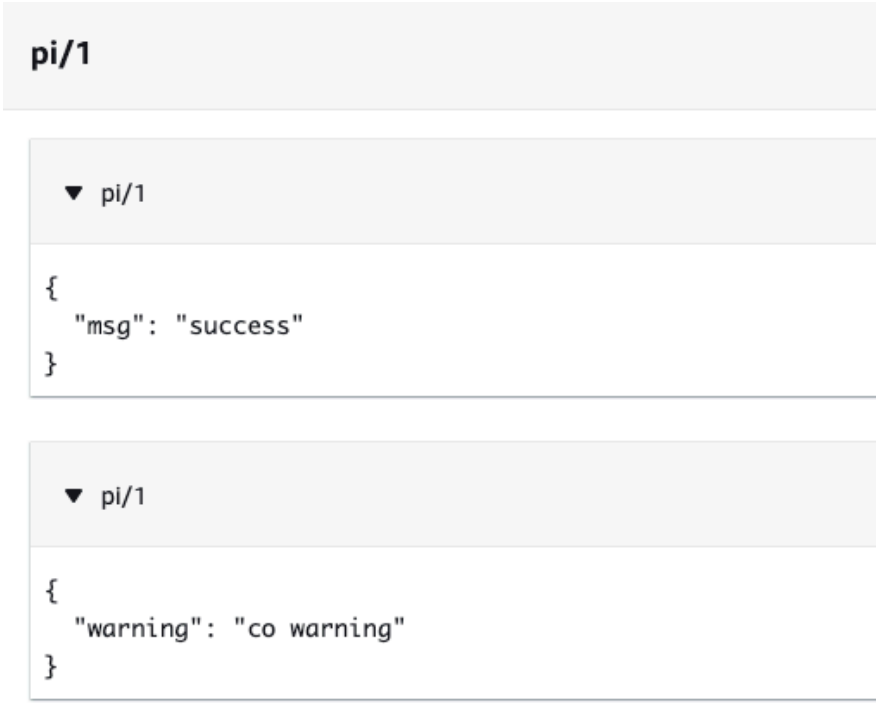
```
data = event
data['date'] = nowDatetime
headers = {'Content-type': 'application/json'}
response = requests.post(url, data = json.dumps(data), headers=headers).json()
```

'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" :
  "_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. Iot core 에서, 2-1 과 같이 요청 받은 publish 를 구독(subscribe)중인 기기에게 브로드캐스트로 메세지를 전송하게 됩니다.



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 메세지를 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 메세지를 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.

날짜	체온	온도	습도
2021-11-29T05:48:24	26.11	26.2	52
2021-11-29T05:49:48	27.11	28.2	80
2021-11-29T05:50:24	29.11	29.2	60
2021-11-29T07:19:30	29.11	29.2	60
2021-11-29T14:34:15	38.25	30.1	13
2021-11-29T14:35:14	38.36	29.8	13
2021-11-29T14:36:14	38.36	29.5	14
2021-11-29T14:37:15	38.32	29.3	15
2021-11-29T14:38:15	38.27	29.1	16
2021-11-29T14:39:45	38.25	29.1	18
2021-11-29T17:20:37	37.91	30	17
2021-11-29T17:21:36	38.01	29.7	17
2021-11-29T17:22:37	38.04	29.5	17
2021-11-29T17:23:37	38.03	29	18
2021-11-29T17:24:37	38.04	29	19

요청된 데이터의 시각화 된 웹사이트 화면