**EC2 상태 확인 및 자동 복구 Lambda 함수 구현**



과목 : 클라우드 시스템 구축

담당교수 : 안해리 교수님

학과 : 컴퓨터 소프트웨어

학번 : 2023136054

이름 : 이세종

목차

1. 개요
2. 구현 내용
3. 설계 이유 및 느낀점

* 개요

본 과제는 AWS Lambda를 활용하여 EC2 인스턴스의 상태를 확인하고, 중지(Stopped)인 경우 경고 메시지를 이메일로 전송한 뒤 인스턴스를 자동으로 실행(running)시키는 시스템을 설계하고 구현하는 것을 목표로 한다.

두 개의 Lambda 함수로 구성하며, LambdaA는 상태를 점검하고 상태가 running이 아닐 경우 SNS 주제를 통해 이메일 알람을 발송하고 LambdaB를 비동기 방식으로 호출한다. LambdaB는 전달받은 EC2 인스턴스 ID를 사용하여 대상 인스턴스가 stopped 상태라면 대상 인스턴스를 시작 처리한다.

* 구현 내용

본 내용은 EC2 상태 점검 및 자동 복구 시스템을 구현하기 위해 사용된 AWS 리소스들과, 각 Lambda 함수의 세부 구현 내용을 설명한다

1. SNS 주제 및 이메일 알람 설정

- 이메일로 알람을 받을 수 있도록 이메일 구독 설정을 한다.

2. sgu-202537-LambdaA 람다 함수 생성

import json

import boto3

def lambda\_handler(event, context):

#클라이언트 초기화

ec2 = boto3.client('ec2')

sns = boto3.client('sns')

lambda\_client = boto3.client('lambda')

#인스턴스 id 와 sns arn, LambdaB 함수 변수 설정

instance\_id = 'i-00f629e077d12d02b'

topic\_arn = 'arn:aws:sns:ap-northeast-2:443370697536:sgu-202537\_CPU\_Alarm'

lambda\_b\_name = 'sgu-202537-LambdaB'

#인스턴스 상태 확인

response = ec2.describe\_instance\_status(InstanceIds =[instance\_id])

statuses = response.get('InstanceStatuses', [])

#상태가 없으면 꺼져 있다고 가정

state = statuses[0]['InstanceState']['Name'] if statuses else 'stopped'

print(f"[INFO] 현재 EC2 상태: {state}")

#인스턴스가 running 이 아닐경우

if state != 'running':

#sns로 알림 전송

message = f"경고: EC2 인스턴스 {instance\_id} 상태가 '{state}' 입니다."

sns.publish(

TopicArn = topic\_arn,

Message = message,

Subject = 'EC2 상태 경고'

)

print("[INFO] SNS 메시지 전송 완료")

#LambdaB 비동기 호출

lambda\_client.invoke(

FunctionName = lambda\_b\_name,

InvocationType='Event',

Payload=json.dumps({'instance\_id': instance\_id})

)

print("[INFO] LambdaB 호출 완료")

#응답반환

return {

"statusCode": 200,

"statusDescription": "200 OK",

"isBase64Encoded": False,

"headers": {

"Content-Type": "application/json"

},

"body": json.dumps({

"message": f"현재 EC2 상태: {state}"

})

}

2. sgu-202537-LambdaB 람다 함수 생성

import boto3

def lambda\_handler(event, context):

#ec2 클라이언트 초기화

ec2 = boto3.client('ec2')

#LambdaA 에서 전달한 인스턴스 id 가져옴

instance\_id = event.get('instance\_id')

#인스턴스 아이디가 있는경우

if instance\_id:

# 인스턴스 실행

ec2.start\_instances(InstanceIds=[instance\_id])

print(f"[INFO] EC2 인스턴스 {instance\_id} 시작 요청 완료")

return {

"statusCode": 200,

"body": f"EC2 인스턴스 {instance\_id}를 시작했습니다."

}

else:

#해당 인스턴스 id 가 없으면 오류표시

return {

"statusCode": 400,

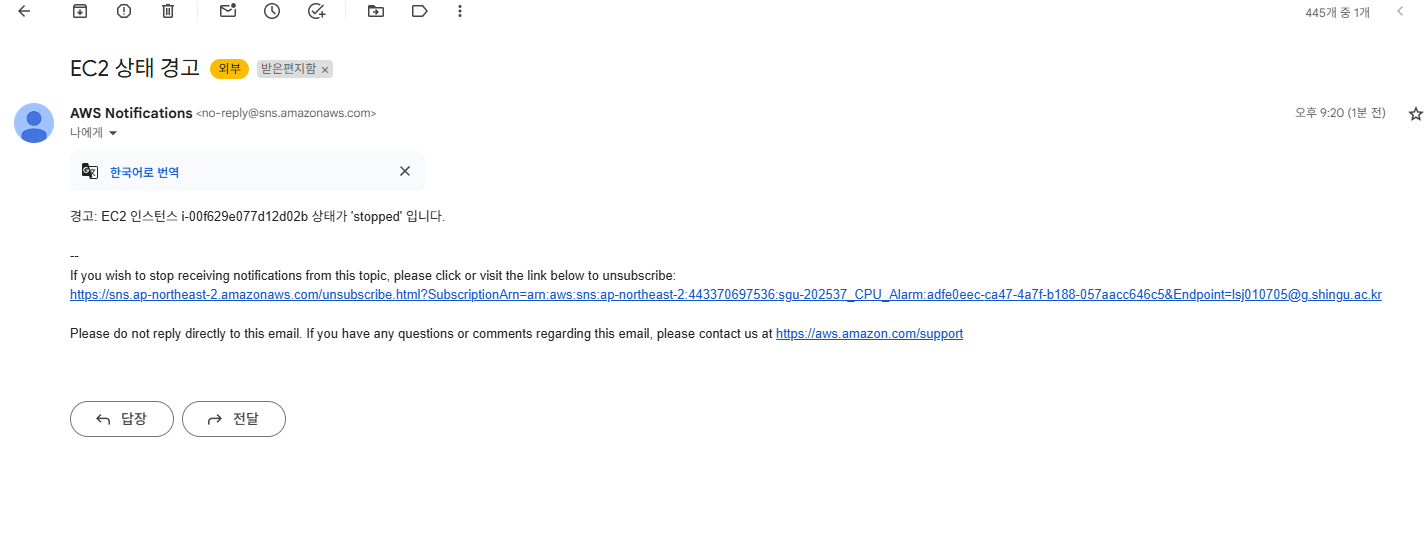
"body": "인스턴스 ID가 없습니다."

}

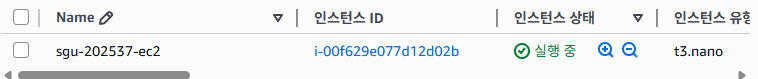
4. 인스턴스 상태가 중지인걸 확인



5. LambdaA를 실행한 후 전송된 상태 경고 이메일을 확인



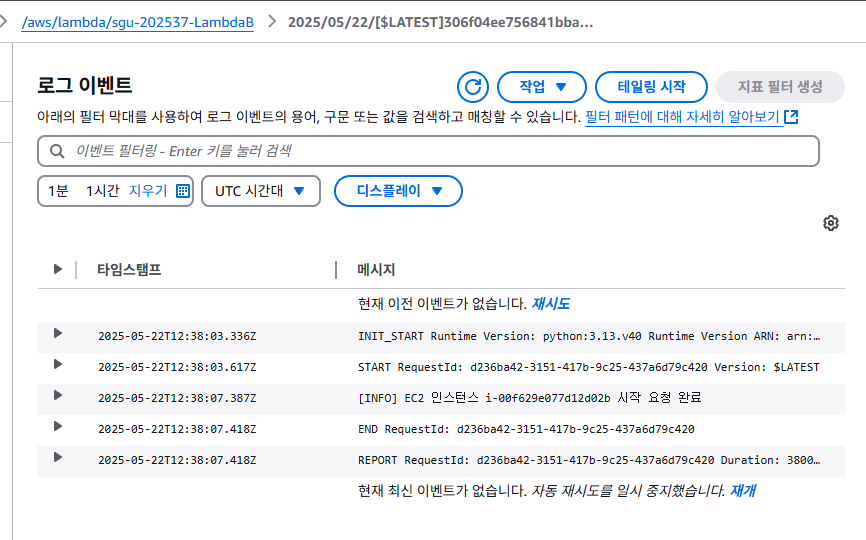
6. 인스턴스가 running 상태로 바뀌었는지 확인



7. CloudWatch 로그를 통한 결과 확인

- 확인하기 쉽도록 코드 실행 위치에 동작을 print하도록 하였다.

LambdaA

LambdaB

* 설계 이유 및 느낀점
* 설계 이유

1. LambdaA와 LambdaB의 분리 이유

LambdaA 는 인스턴스의 상태를 점검하고, 상태에 따라 SNS 를 통해 경고 메세지를 발송한 뒤 LambdaB를 호출하는 역할을 한다.

LambdaB는 LambdaA로부터 호출되어 전달받은 인스턴스 ID를 기준으로 인스턴스를 시작하는 기능을 담당한다.

이러한 구조는 역할 분리 원칙으로 설계된 것으로, 각 함수가 하나의 책임에 집중함으로써 유지보수와 가독성을 높일 수 있다.

함수를 재사용 할 수도 있으며 함수 간의 결합도를 낮춤으로써 유연성을 확보할 수 있다.

2. 비동기 호출 사용 이유

LambdaA에서 LambdaB를 호출할 때, 비동기 방식을 사용하는 이유는 LambdaA의 실행 흐름을 불필요하게 지연시키지 않기 위함이다.

비동기 호출 방식을 적용함으로써 LambdaA는 SNS 알림 전송 이후 LambdaB를 호출한 직후 바로 종료되며, LambdaB는 독립적으로 EC2 시작 명령을 수행할 수 있다.

이를 통해 시스템의 응답성과 자원 효율성을 높일 수 있다

3. 실무 적용 시 기대 효과

이와 같이 역할을 분리한 구조는 실무 환경에서도 여러 가지 장점을 제공한다.

역할이 명확히 분리된 모듈형 구조는 유지보수에 용이하며, 최소한의 수정으로 재사용이 가능하다. 또, 독립적 구조로 테스트 및 디버깅이 용이하여 오류 발생 시 빠르게 추적할 수 있다.

이러한 구조는 실시간 자동화, 서버 비용 절감, 운영 간소화 등의 실무적 이점들이 있다.

* 느낀 점

이번 과제를 하며 AWS Lambda를 활용한 아키텍처 설계와 자동화 처리 방식에 대해서 흐름을 확실하게 이해할 수 있었다.

특히 LambdaA에서 LambdaB를 비동기적으로 호출하는 구조를 직접 구성하며 이미 알고 있던 방식을 단순하지만 확실하게 알 수 있었고,

이런 실무적인 경험을 직접적으로 하며 실질적인 자신감을 쌓을 수 있는 유익한 실습이었다.