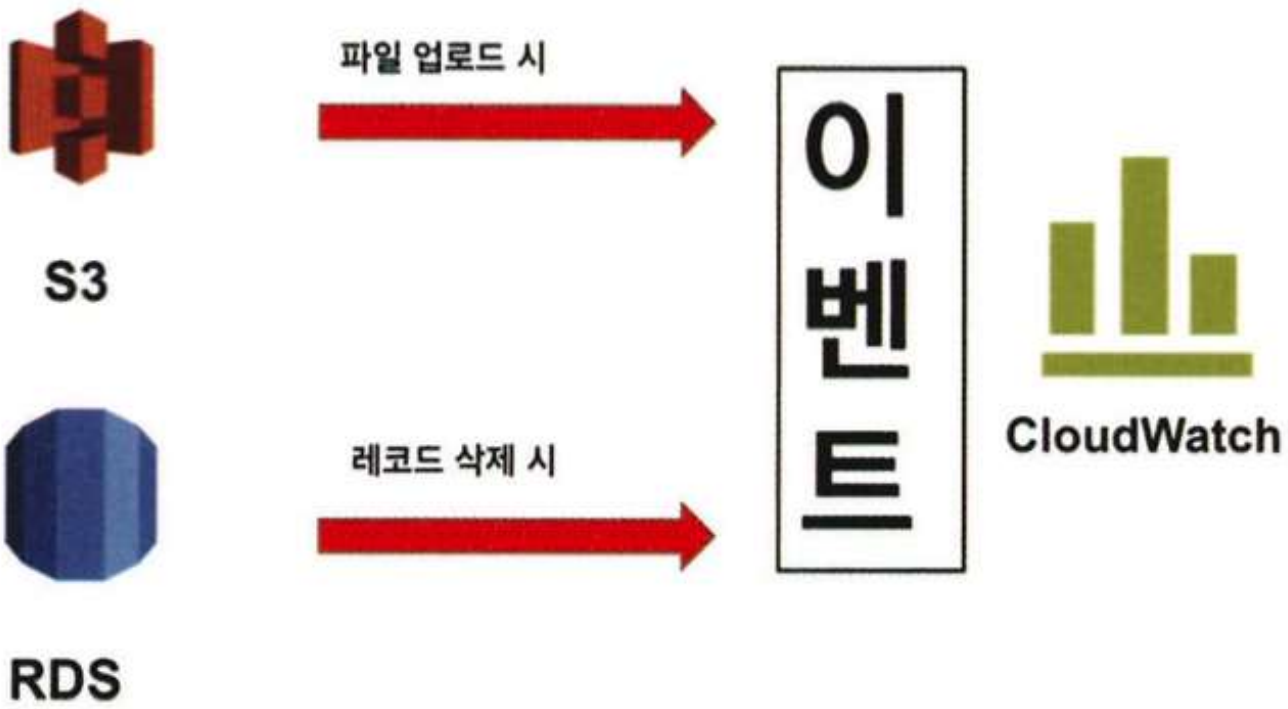
AWS Lambda

Cloud Watch

❖ 개요

✓ CloudWatch는 리소스 와 애플리케이션을 실시간으로 모니터링하는 서비스



Cloud Watch

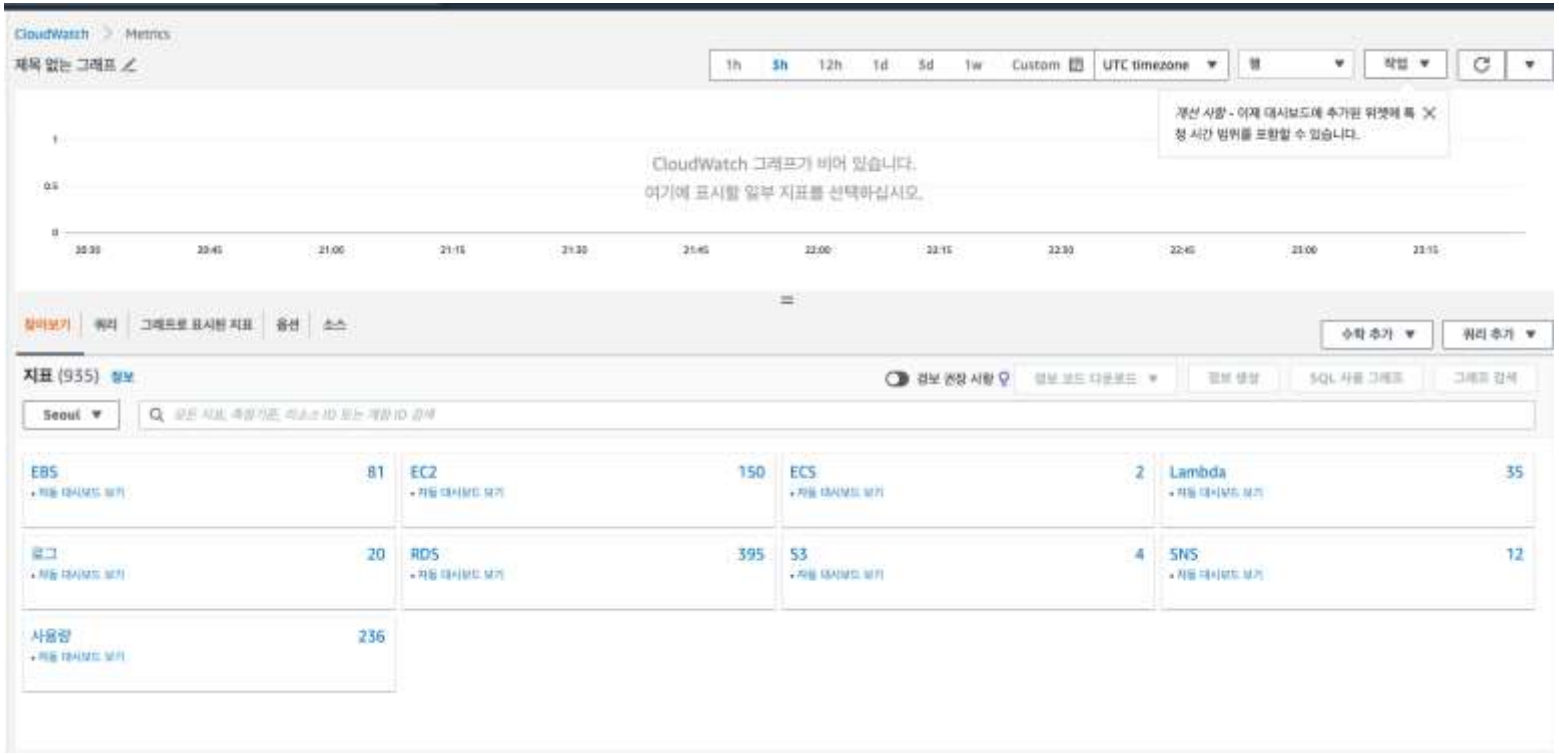
- ❖ 개요
  - ✓ 로그
    - 애플리케이션의 리소스를 기록
    - CloudWatch의 로그는 로그 그룹 단위로 기록
    - 여러 서비스에서 발생하는 로그를 기록하며 이를 필요에 따라 필터링하여 원하는 데이터를 검색할 수 있음
    - 로그 기록 만료 시점을 설정할 수 있음
    - 로그 그룹 명을 클릭하여 세부 정보를 확인할 수 있음
    - 로그 그룹의 현재 저장된 용량 을 확인할 수 있고 삭제 및 Amazon S3에 저장할 수 있다
    - 로그 그룹에서 로그 데이터를 확인할 수 있음
    - 이를 삭제하거나 필터링하여 원하는 기준으로 데이터를 찾을 수 있는데 검색 결과를 복사하여 CSV로 다운로드할 수 있음
    - Logs Insights를 사용하면 원하는 로그를 쿼리로 호출하여 확인 가능

Cloud Watch

❖ 개요

✓ 지표

- CloudWatch에서는 지표를 통해 그래프 형식으로 로그를 표현

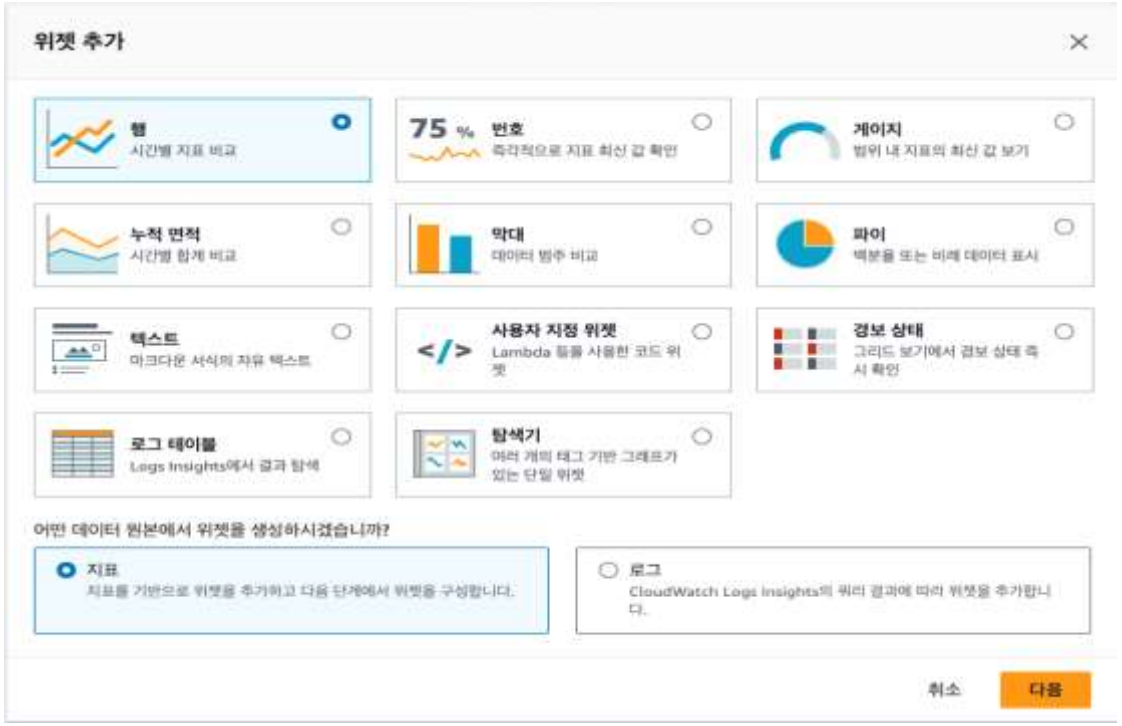


Cloud Watch

- ❖ 개요
  - ✓ 이벤트
    - CloudWatch는 이벤트를 제공
    - 규칙을사용하면 서비스 트리거, Cron job과 같은 형식의 작업이 가능
    - 규칙은 이벤트 패턴 또는 일정으로 생성할 수 있음
    - 이벤트 패턴은 서비스와 트리거하는 개념으로 특정 서비스를 지정하고 이벤트 유형을 선택한 후 대상 서비스를 지정하여 동작하도록 설정할 수 있음
    - 이벤트 패턴은 서비스 단위를 선택할 뿐만 아니라 사용자 지정 이벤트 패턴으로 직접 작성할 수 있음

Cloud Watch

- ❖ 개요
  - ✓ 대시보드
    - CloudWatch에서 대시보드를 사용할 수 있음
    - 한 페이지 내에서 원하는 로그를 원하는 형식으로 확인할 수 있음
    - 대시보드는 사용자가 직접 생성 및 수정 및 삭제가 가능하며 원하는 레이아웃 형태로 생성 가능



Cloud Watch

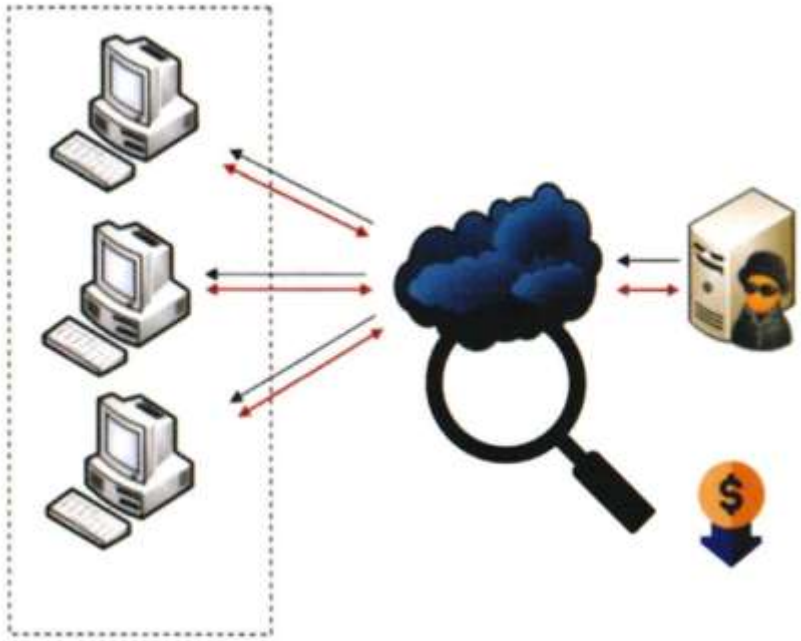
- ❖ 개요
  - ✓ 모니터링 종류
    - ● 기본 모니터링
      - ◆ 무료이며 5분 간격으로 최소한의 데이터를 수집하고 사용자에게 이벤트 발생 여부를 알려줌
      - ◆ CPU 사용량, 디스크 사용량, 네트워크 I/O 처리량 관련 상황을 체크할 때 사용되는 모니터링 종류
      - ◆ AWS 리소스 사용 시 ELB와 RDS를 제외하고 디폴트로 기본 모니터링 종류가 사용됨
    - ● 상세 모니터링
      - ◆ 무료가 아니지만 1분 간격으로 매우 자세한 데이터를 수집
      - ◆ 대부분의 리소스는 기본 모니터링에서 상세 모니터링으로 변경 가능하나 ELB와 RDS는 상세 모니터링 옵션이 디폴트로 선택되어 있으며 이 둘은 기본 모니터링 기능을 사용할 수 없음

Cloud Watch

- ❖ 개요
  - ✓ 사용 사례
    - ● 서버 과부한 문제 해결
      - ◆ 쇼핑 앱을 하나 만들었고 매일 전 세계에서 얼마나 많은 유저가 앱을 사용하는지 알아야 하는 경우 특정 요일과 시간 별로 나눠서 언제 유저가 많이 몰리는지 패턴을 분석해야 함
      - ◆ 만약 공휴일에 비정상적으로 유저가 많이 몰리면 서버 과부하 문제가 발생하며 수많은 트래픽으로 인해 병목 현상이 일어날 수 있는데 매일 개발자가 확인하는 것은 현실적으로 불가능한데 Cloud Watch를 통해 언제 얼마나 많은 사람이 앱에 접속했는지 로그와 매트리스를 통해 한눈에 확인할 수 있으며 이를 기반으로 앱 사용 패턴을 파악하고 병목 현상 문제를 유연하게 대처할 수 있음

Cloud Watch

- ❖ 개요
  - ✓ 사용 사례
    - ● 비용 절감 효과
      - ◆ Cloud Watch 경보 기능을 활용하면 서버 트래픽이 특정 임계점에 도달했을 때 개발자에게 알람을 알려주고 성능을 자동으로 낮출 수 있음

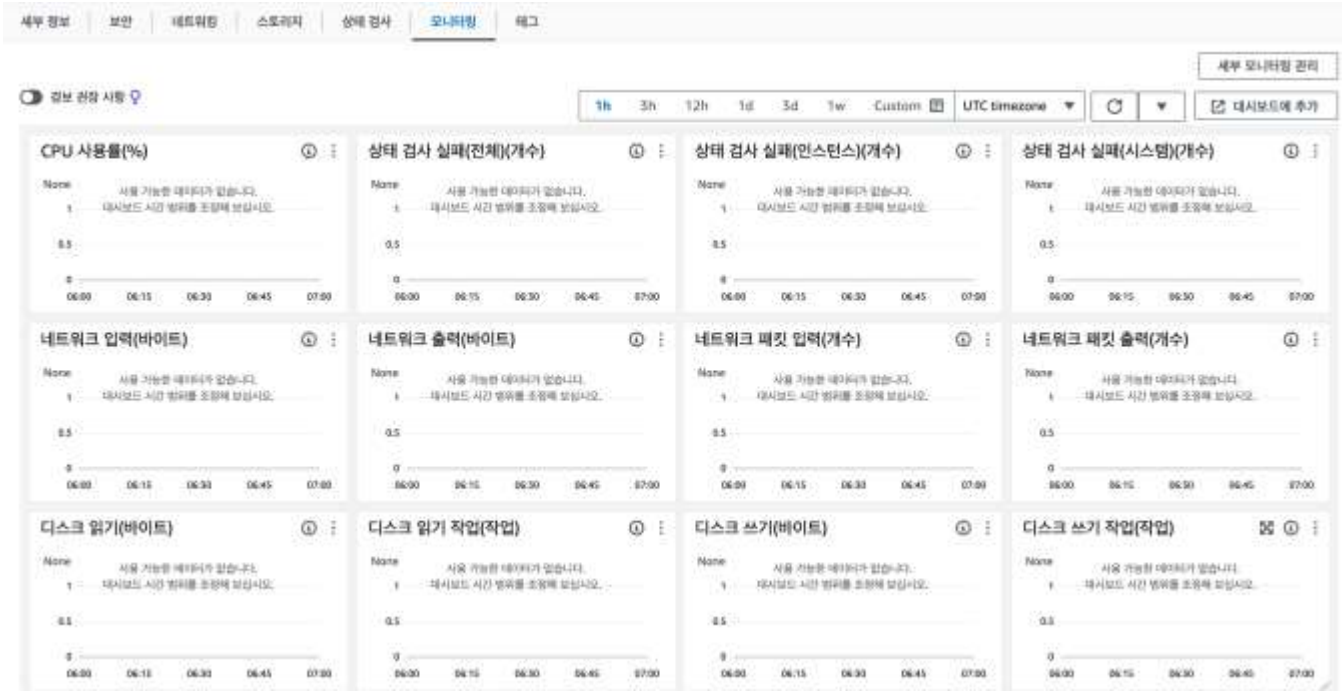


Cloud Watch

- 경보
  - ✓ 세가지 상태
    - CloudWatch 경보에는 세 가지 다른 상태가 존재하며 예외는 없음
    - OK 상태는 Cloud Watch에서 수집하는 매트리스가 개발자가 지정해 놓은 임곗값에서 타당한 값으로 들어온다면 OK 상태로 경보가 울릴 필요없는 정상적인 상태를 의미하며 임곗값을 벗어나게 된다면 상태는 ALARM으로 변경되는데 개발자는 경보가 울렸을 경우 취할 수 있는 행동을 정의할 수 있음
    - INSUFFICIENT_DATA 상태는 OK 혹은 ALARM으로 구분 지을 수 있을 만큼 충분한 데이터가 쌓이지 않거나 전달받지 못했을 경우에 발생
    - IoT 기기에서 매초 데이터를 전달받아야 하는데 알 수 없는 방해 요인으로 데이터 전송 지연이 발생하여 CloudWatch는 데이터가 쌓이지 않고 있다고 판단하면 불충분 데이터라고 알려주는데 데이터가 없다면 매트리스가 존재하지 않기 때문

Cloud Watch

- ❖ 모니터링 실습
  - ✓ EC2 인스턴스 생성
    - SSH 나 Putty 로 접속할 수 있도록 생성하고 인스턴스 실행
    - 모니터링 탭을 눌러서 현재 상태 확인



Cloud Watch

- ❖ 모니터링 실습
  - ✓ EC2 인스턴스 생성
    - SSH로 접속해서 내부 상태 확인: ls

Cloud Watch

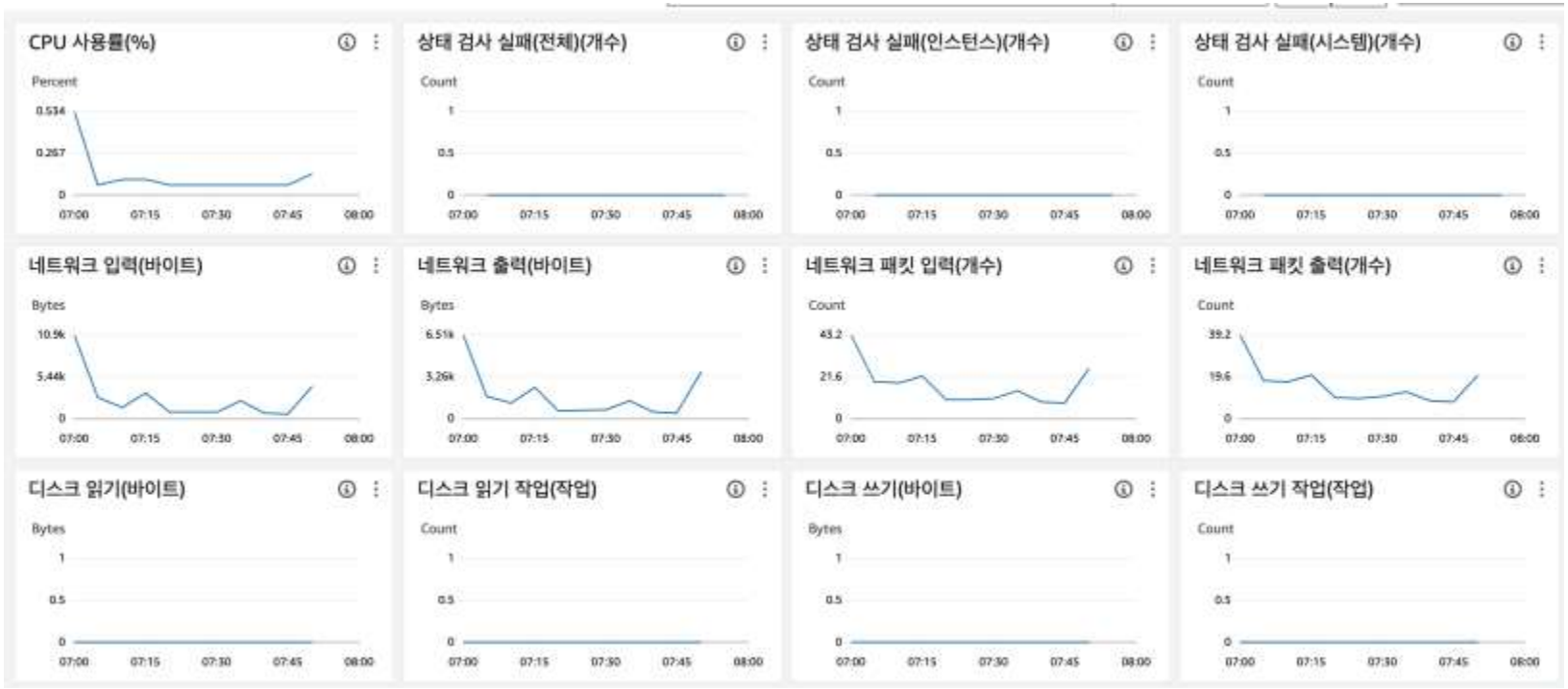
- ❖ 모니터링 실습
  - ✓ 파일 전송 후 확인
    - 용량이 큰 파일을 다운로드: https://www.jetbrains.com/datagrip/download/
    - 리눅스 서버로 파일 전송
      - ◆ Mac

        scp -i pem파일경로 전송할파일경로 계정@접속할서버:저장될파일경로
      - ◆ Windows

        pscp -i ppk파일경로 전송할파일경로 계정@접속할서버:저장될파일경로

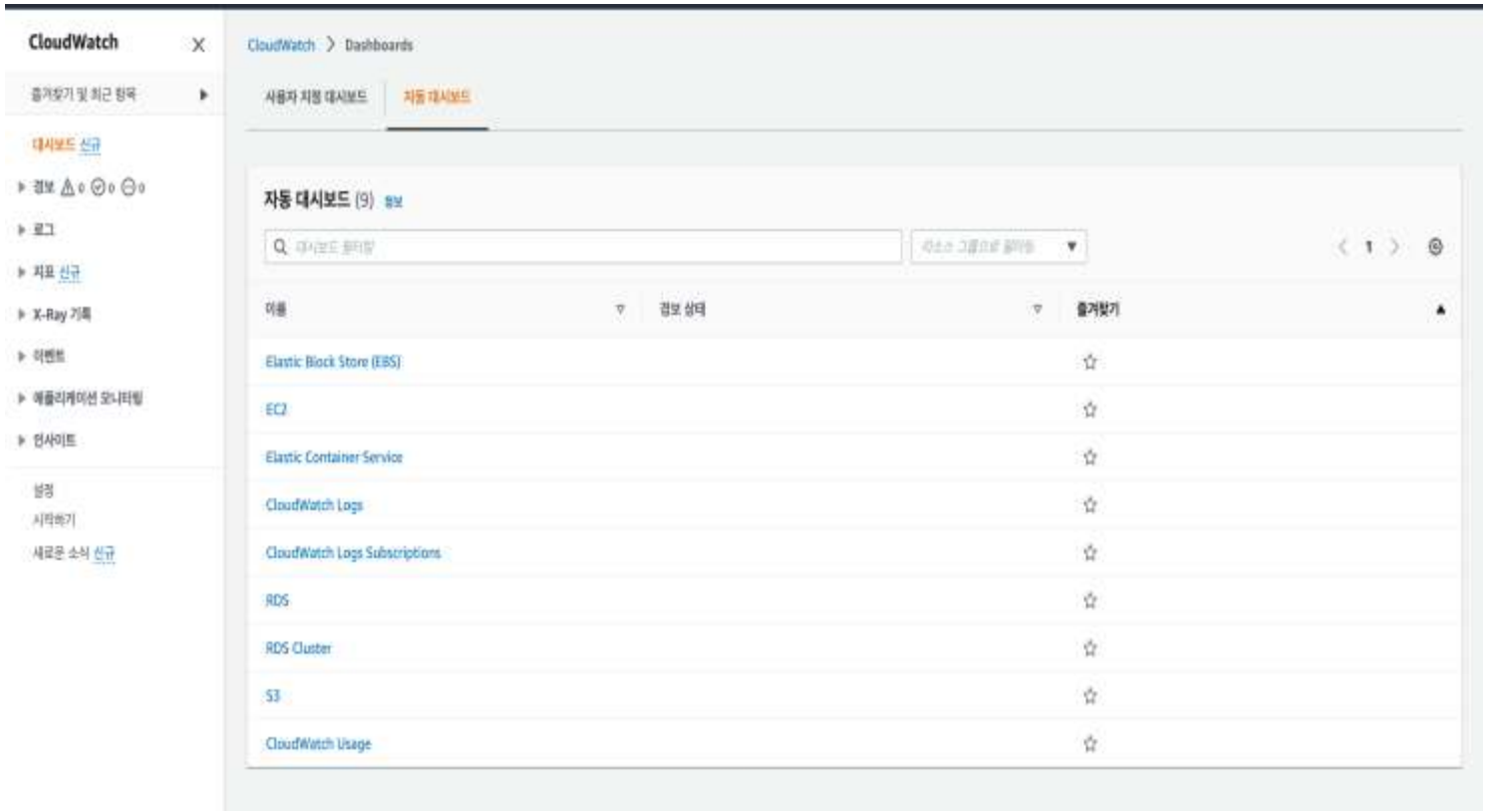
Cloud Watch

- ❖ 모니터링 실습
  - ✓ 파일 전송 후 확인
    - ● 모니터링 확인



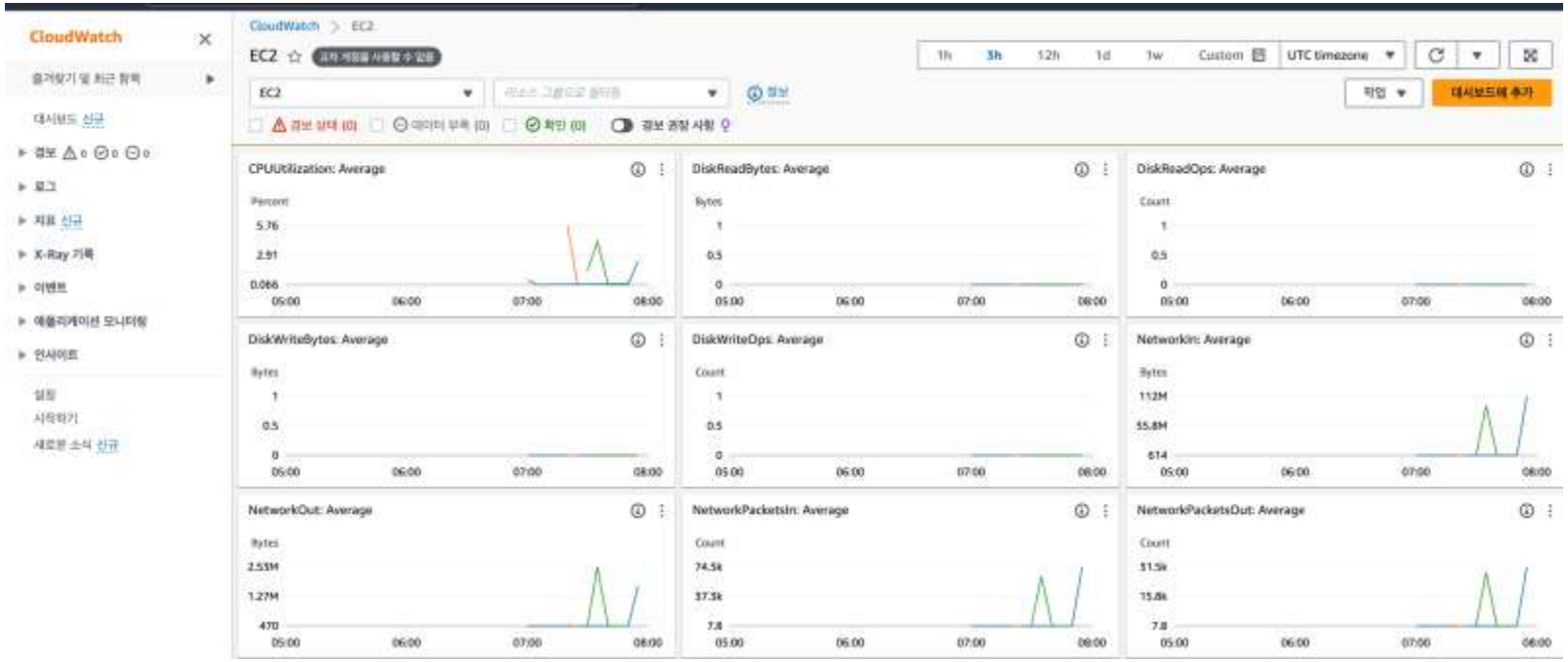
Cloud Watch

- Cloud Watch 대시보드에서 모니터링
  - ✓ Cloud Watch 에서 EC2 대시보드 확인



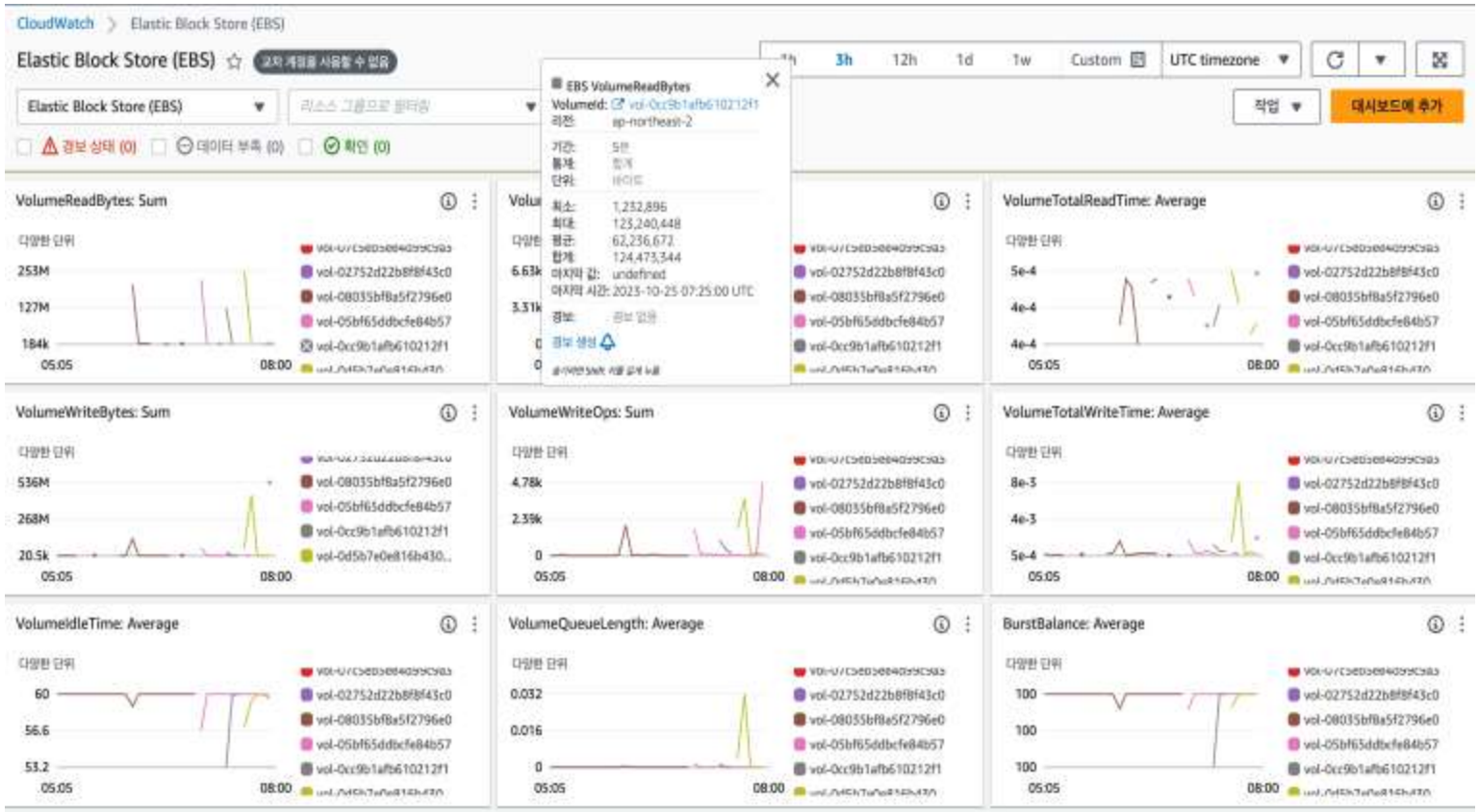
Cloud Watch

- ❖ Cloud Watch 대시보드에서 모니터링
  - ✓ Cloud Watch 에서 EC2 대시보드 확인



Cloud Watch

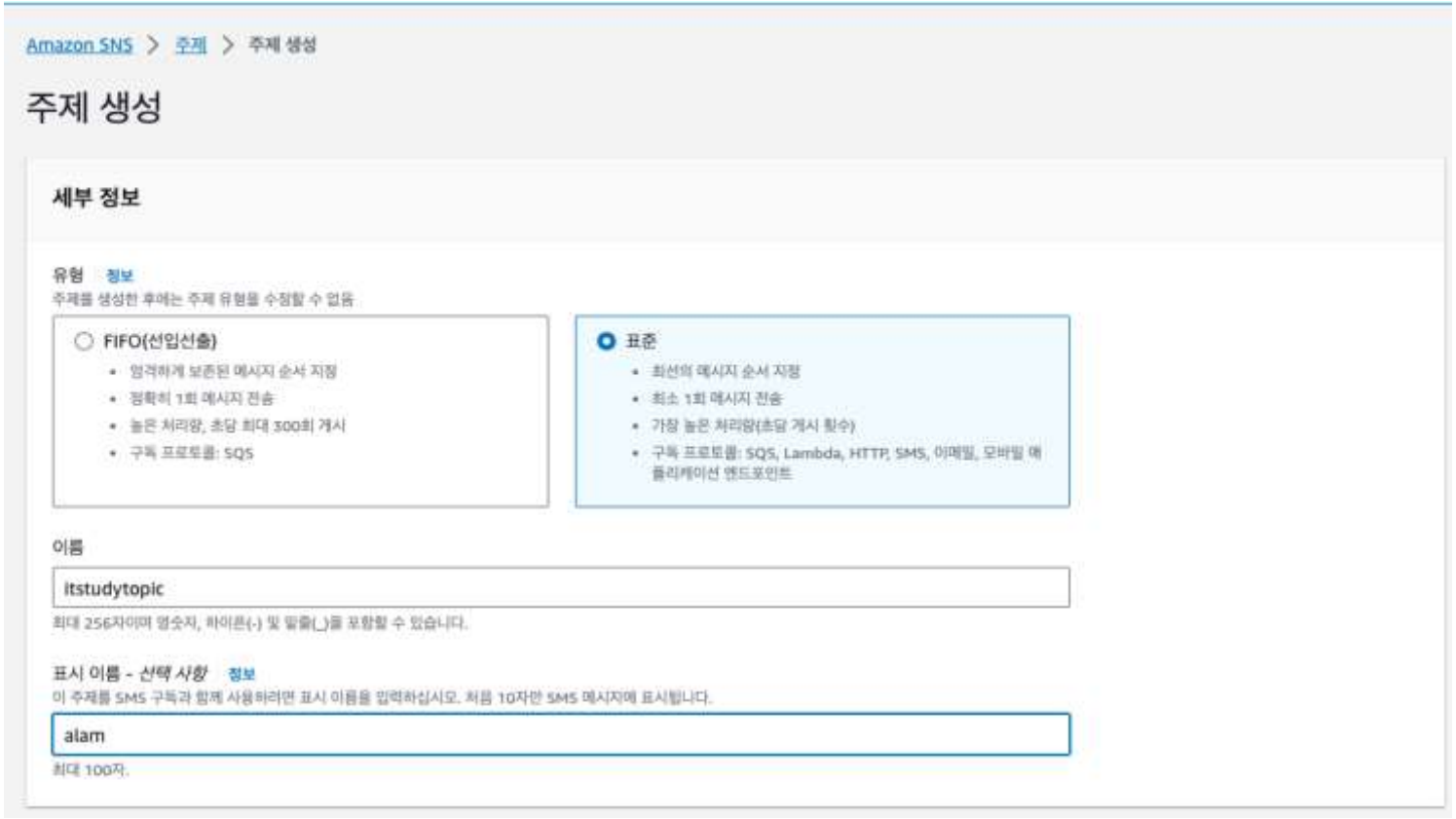
- Cloud Watch 대시보드에서 모니터링
  - ✓ Cloud Watch 에서 EBS 대시보드 확인



Cloud Watch

❖ Cloud Watch 경보 생성

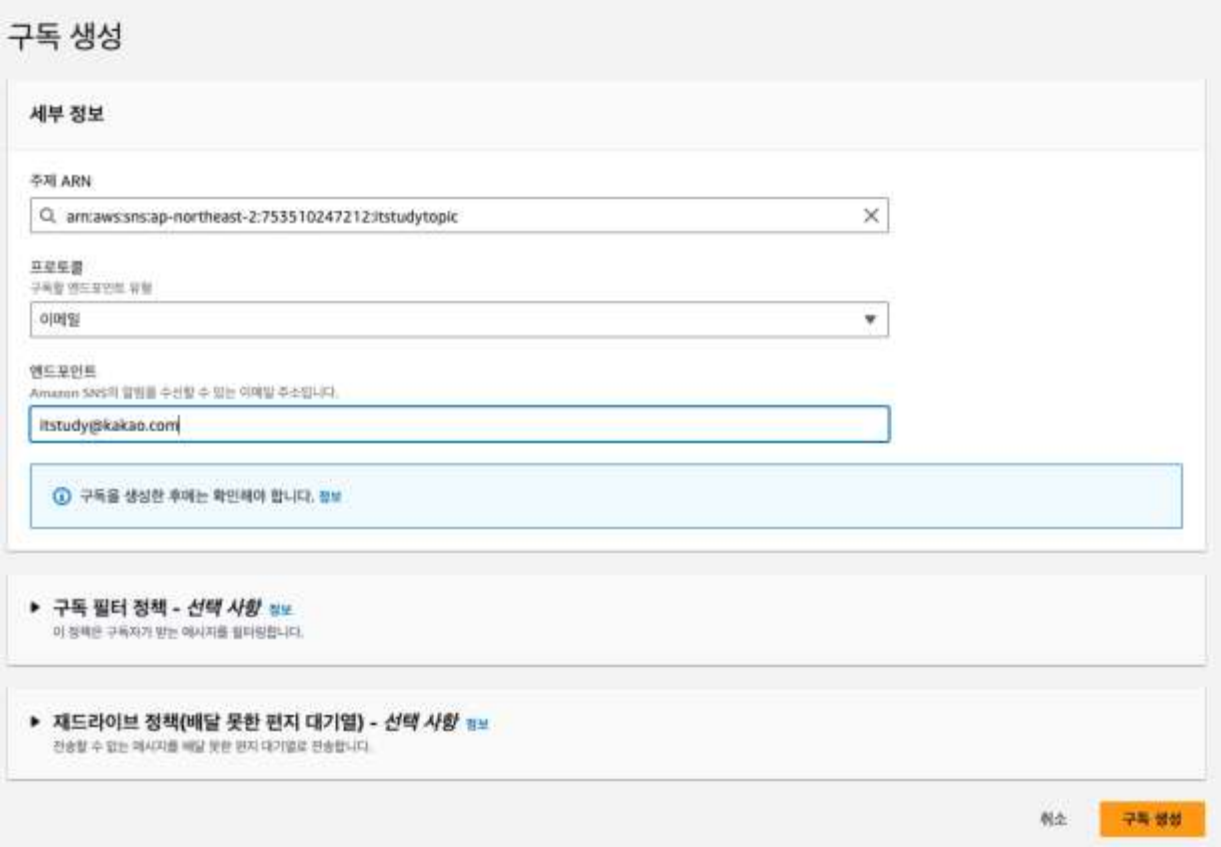
✓ SNS 관리 콘솔에서 주제 생성



Cloud Watch

❖ Cloud Watch 경보 생성

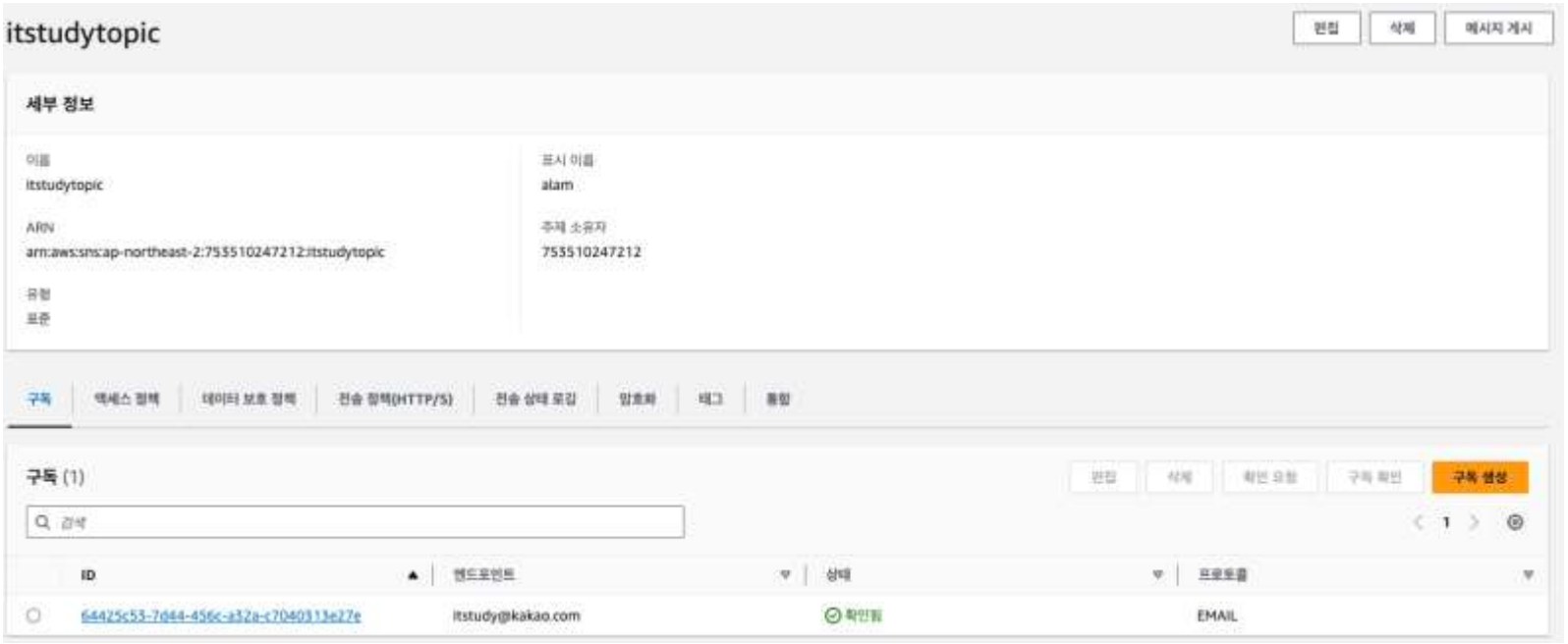
✓ SNS 관리 콘솔에서 주제 안에서 구독 생성



Cloud Watch

❖ Cloud Watch 경보 생성

✓ 이메일이 전송되는데 이 때 confirm

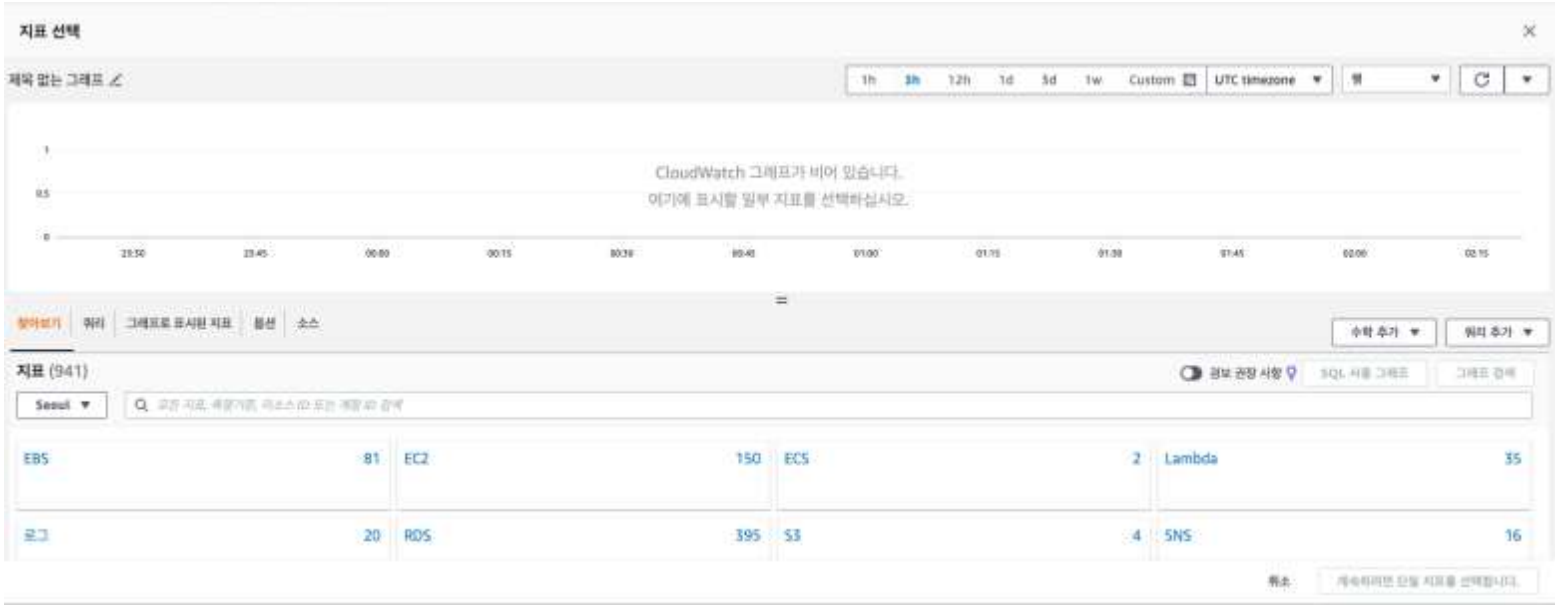


Cloud Watch

- ❖ Cloud Watch 경보 생성
  - ✓ 경보 생성
    - ● 경보 생성 – 지표를 선택

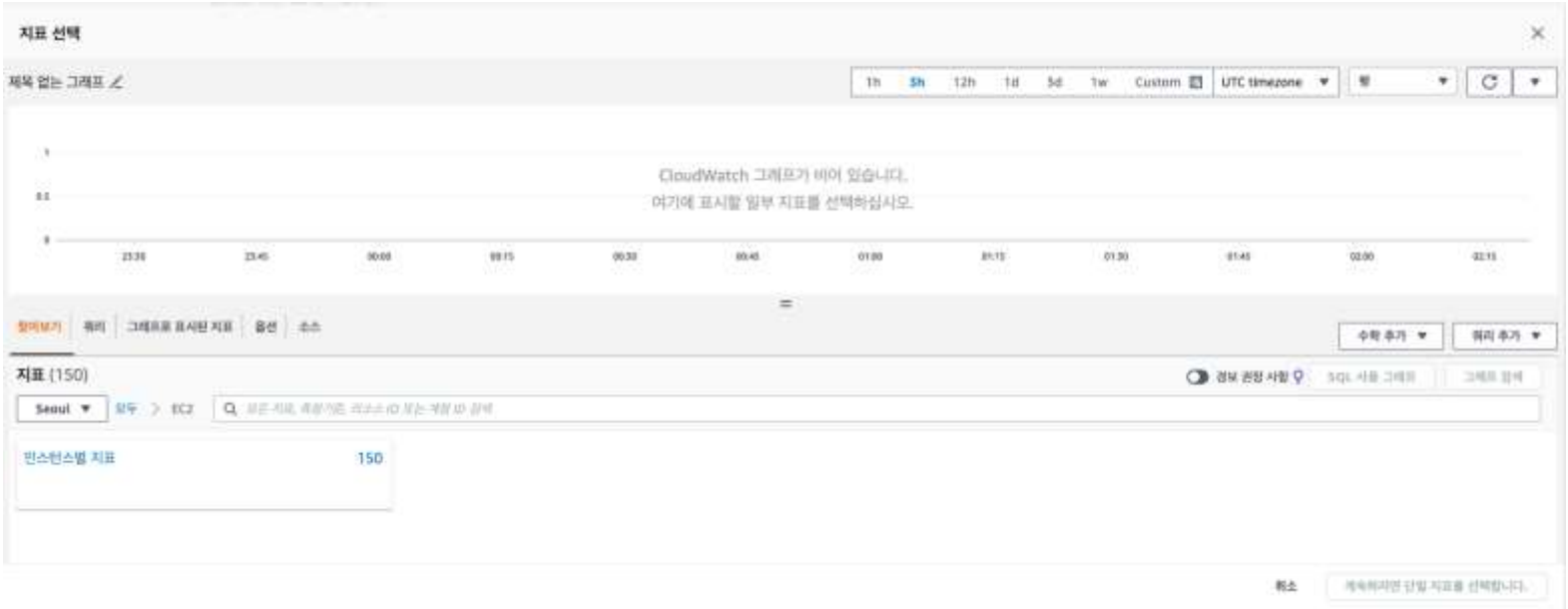
Cloud Watch

- ❖ Cloud Watch 경보 생성
  - ✓ 경보 생성
    - ● 경보 생성 – 지표를 선택



Cloud Watch

- ❖ Cloud Watch 경보 생성
  - ✓ 경보 생성
    - ● 경보 생성 – 지표를 선택



Cloud Watch

❖ Cloud Watch 경보 생성

✓ 경보 생성

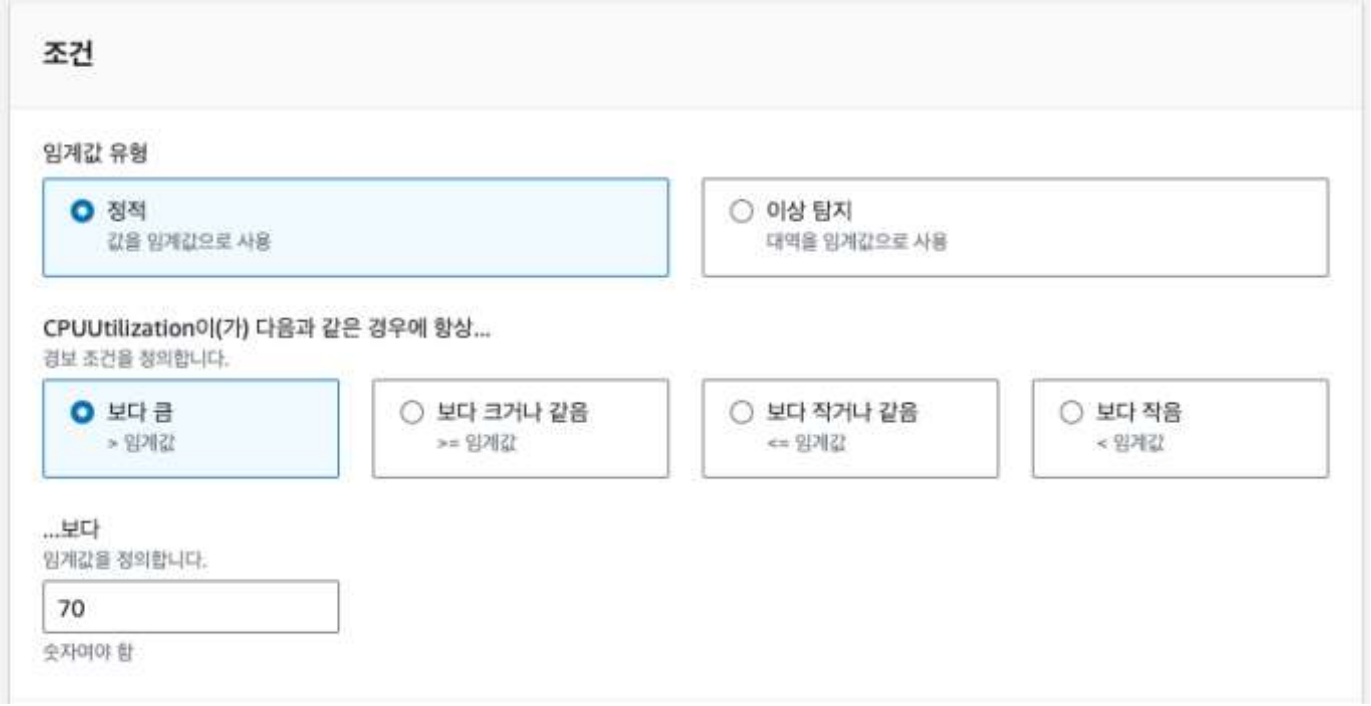
● 경보 생성 – 지표를 선택

찾아보기 | 쿼리 | 그래프로 표시된 지표(1) | 옵션 | 소스

☐	i-0442a136a3d5cf30d	StatusCheckFailed ⓘ	경보 없음
☐	i-0442a136a3d5cf30d	MetadataNoToken ⓘ	경보 없음
☐	i-099ad793dee38d156	MetadataNoToken ⓘ	경보 없음
☑	i-099ad793dee38d156	CPUUtilization ⓘ	1개의 경보
☐	i-099ad793dee38d156	NetworkOut ⓘ	경보 없음
☐	i-099ad793dee38d156	DiskReadOps ⓘ	경보 없음

Cloud Watch

- ❖ Cloud Watch 경보 생성
  - ✓ 경보 생성
    - ● 경보 생성 – 조건 설정



Cloud Watch

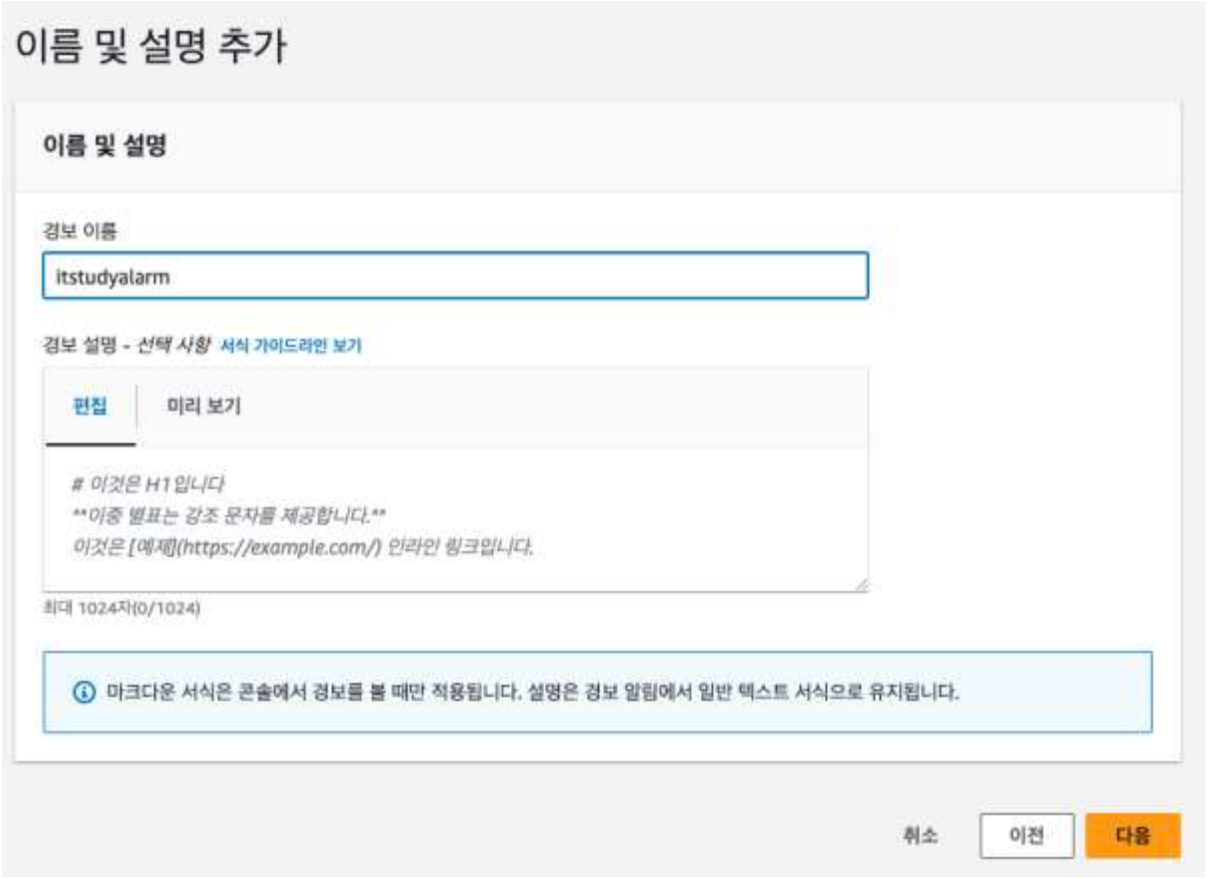
❖ Cloud Watch 경보 생성

✓ 경보 생성

● 작업 구성 – 이전에 만든 주제를 선택

Cloud Watch

- ❖ Cloud Watch 경보 생성
  - ✓ 경보 생성
    - ● 이름 설정



Cloud Watch

- ❖ Cloud Watch 경보 생성
  - ✓ 컴퓨터에 부하 발생
    - ssh로 EC2에 접속
    - 패키지 정보 업데이트: sudo apt-get update
    - stress 패키지 설치: sudo apt install stress
    - CPU 코어 수 확인: grep –c processor /proc/cpuinfo
    - CPU 스트레스: stress –c 2
    - 메모리 스트레스: stress --vm 3 --vm-bytes 1024m
    - Disk 스트레스: stress --hdd 3 -hdd-bytes 1024m
    - 다른 컴퓨터에서 EC2에 파일 전송
      - ◆ scp -i <pem/ppk 파일명> <업로드할 파일> ec2-user@<퍼블릭 IPv4 DNS이름>:<인스턴스에 업로드될 파일명>
      - ◆ Windows 는 scp 대신에 pscp
    - 잠시 후 메일 확인

ServerLess

- Serverless
  - ✓ 개요
    - 서버가 없는건 아니고 그저 특정 작업을 수행하기 위해서 서버 프로그래밍을 별도로 할 필요가 없다 라는 의미
    - BaaS(Backend as a Service) 혹은 FaaS(Function as a Service)에 의존하여 작업을 처리
    - BaaS를 제공하는 서비스에는 Firebase, Kinvey 등이 있고 FaaS를 제공하는 서비스로는 AWS Lambda, Azure Functions, Google Cloud Functions 등이 있음

ServerLess

- Serverless
  - ✓ 기존 기술
    - 자체적 시스템 설계
      - 시스템에서 필요한 모든 인프라를 직접 관리하는 것
      - 공간, 하드웨어, 네트워크, 운영체제, 모두 직접 관리
      - 가장 큰 문제점은 시스템이 커지면 전산실을 유지 할 관리자가 필요 할 것이고 이 인력에 대한 비용이 나가면 시스템의 확장이 어려움
    - IaaS (Infrastructure as a Service)
      - AWS, Azure 등의 서비스를 이용하는 방식으로 서버 자원, 네트워크, 전력 등의 인프라를 모두 직접 구축 할 필요 없음
      - 인프라를 가상화하여 관리하기 쉽게 해주는 서비스를 통하여 관리자 패널에서 인프라를 구성하고 사용하는 방식으로 사용자는 가상 머신을 만들고 네트워크, 와 하드웨어를 설정하고 운영체제를 설치해서 애플리케이션을 구동하는 방식
    - PaaS (Platform as a Service)
      - IaaS 에서 한번 더 추상화된 모델로 네트워크 및 런타임까지 제공
      - 사용자는 애플리케이션만 배포하면 바로 구동시킬 수 있는 방식으로 AWS Elastic Beanstalk, Azure App Services 등이 있으며 이를 사용하면 Auto Scaling 및 Load Balancing 도 손쉽게 적용 할 수 있음

ServerLess

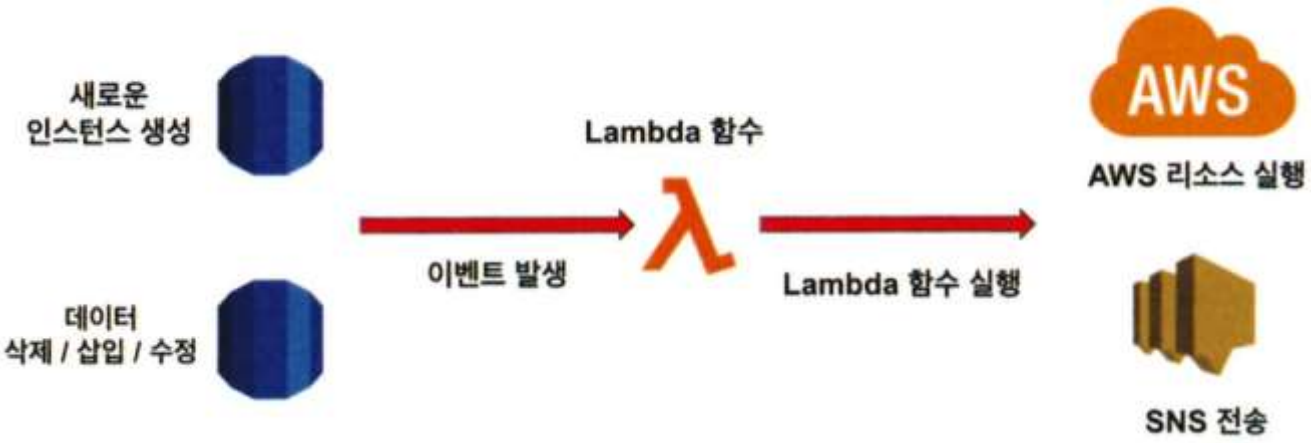
- Serverless
  - ✓ BaaS 와 FaaS
    - BaaS(Backend as a Service)
      - 모바일 혹은 웹 애플리케이션을 만들게 될 때 데이터를 저장하고 다른 기기에서도 접근하고 공유하기 위해서는 서버 개발은 필수적
      - 서버 개발을 하다 보면 고려할 사항이 꽤 많은데 유저가 늘어나게 된다면 서버의 확장도 고려해야 하고 보안성 또한 고려해야 하는데 이는 어려운 작업
      - Firebase 같은 BaaS 에서는 앱 개발에 있어서 필요한 다양한 서버의 기능들 (데이터베이스, 소셜 서비스 연동, 파일시스템 등)을 API로 제공해 줌으로써 개발자들이 서버 개발을 하지 않고서도 필요한 기능을 쉽고 빠르게 구현 할 수 있게 해주고 서버의 이용자가 순식간에 늘어나게 되면 자동 확장
      - 현존하는 BaaS 중 가장 대중화 되어있는 것은 Firebase
      - 장점
        - 개발 시간의 단축
        - 서버 확장 작업의 불필요함
      - 단점
        - 클라이언트 위주의 코드
        - 가격
        - 복잡한 쿼리가 불가능함

ServerLess

- Serverless
  - ✓ BaaS 와 FaaS
    - FaaS(Function as a Service)
      - FaaS는 개발자에게 서버를 관리할 필요 없이 이벤트에 대응하여 애플리케이션을 실행하도록 지원하는 추상화 기능을 제공
      - FaaS 인프라는 주로 이벤트 기반 실행 모델을 통해 서비스 제공업체가 온디맨드로 미터링하는 것이 일반적
      - 서비스로서의 플랫폼(Platform-as-a-service, PaaS) 처럼 서버 프로세스를 백그라운드에서 계속 실행할 필요는 없음
      - 현대적인 PaaS 솔루션은 개발자가 애플리케이션을 배포하는 데 사용할 수 있는 일반적인 워크플로우의 일부로 서버리스 기능을 제공하여 PaaS와 FaaS 간 구분이 모호해짐
      - 실제로 전체 애플리케이션은 기능, 마이크로서비스, 지속 실행 서비스 같은 여러 솔루션이 혼합된 형태로 구성
      - FaaS의 이점
        - 개발자 생산성 향상 및 개발 시간 단축
        - 서버 관리의 부담이 없음
        - 손쉬운 확장 및 플랫폼에서 관리하는 수평적 스케일링
        - 필요한 경우에만 리소스를 사용하거나 리소스 비용 지불
        - 거의 모든 프로그래밍 언어로 기능 작성 가능

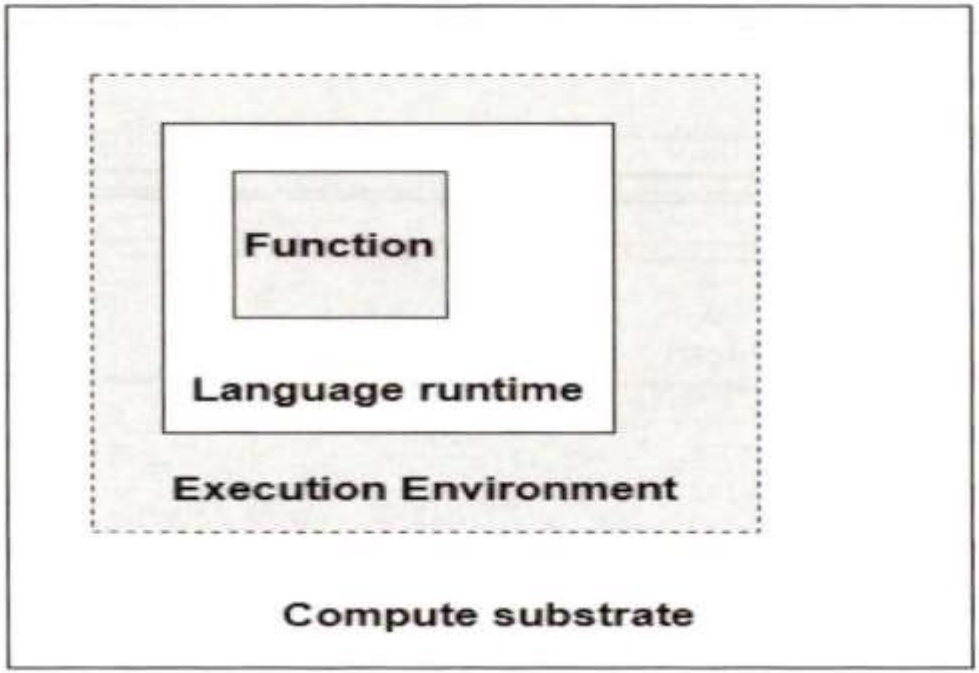
ServerLess

- AWS Lambda
  - ✓ AWS Lambda란 Amazon Web Services에서 2014년부터 제공하는 FaaS 서비스
  - ✓ AWS Lambda는 이벤트를 감지하여 아마존 리눅스 환경의 Micro VM을 띄우고 함수를 실행하고 결과를 처리
  - ✓ 함수가 실행될 때 필요한 환경이 있는데 이것을 런타임이라고 하고 런타임은 어떤 언어로 작성하는지에 따라 다르며 그 환경에 따라 성능 차이가 있음



ServerLess

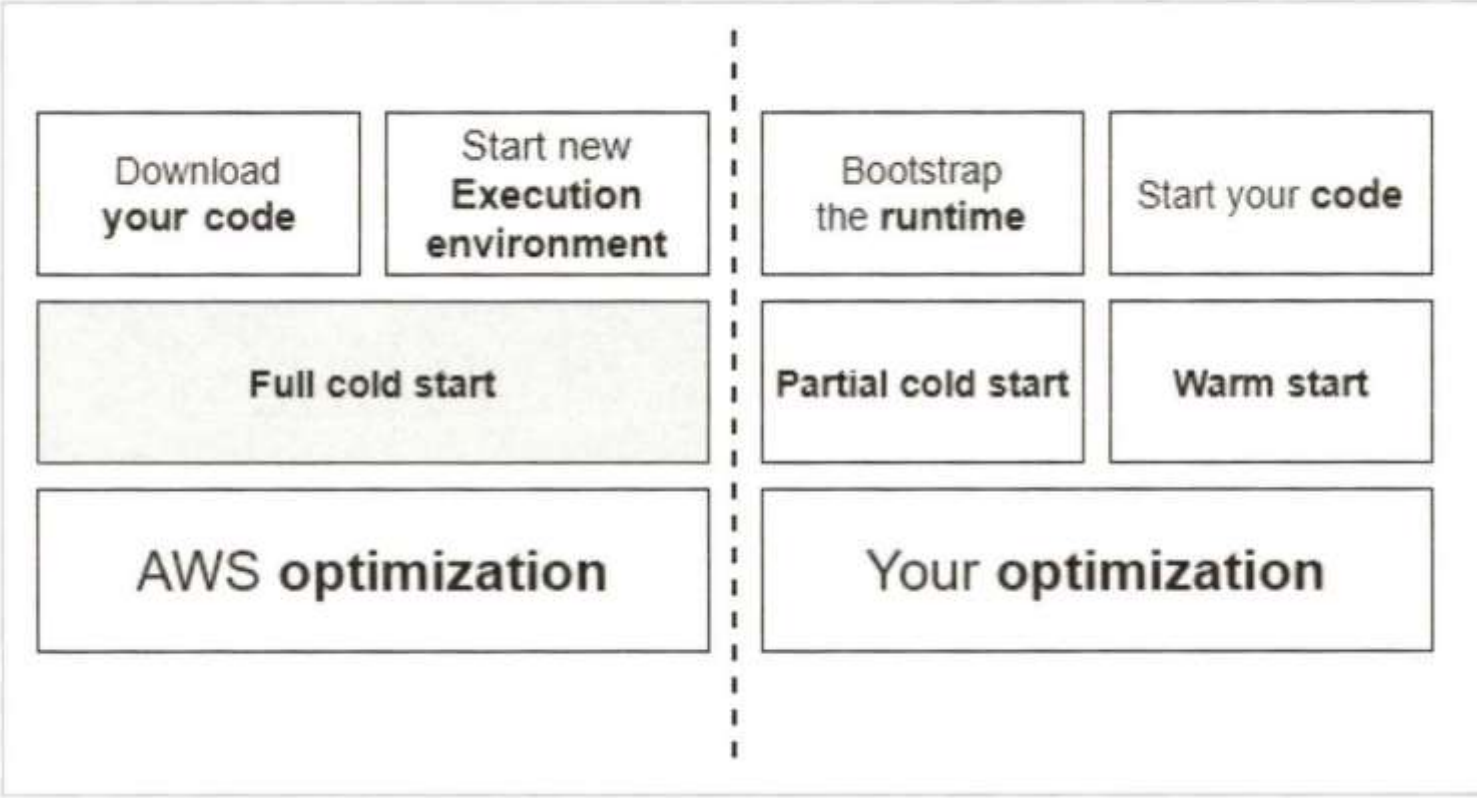
- AWS Lambda
  - ✓ 내부 구조



Compute substrate	함수가 실행될 Micro VM
Execution Environment	환경 변수 등의 실행환경
Language runtime	언어별 런타임
Function	함수

ServerLess

- Cold Start 와 Warm Start
  - ✓ Life Cycle



ServerLess

- ❖ Cold Start 와 Warm Start
  - ✓ Cold Start 발생함으로써 딜레이 존재
    - AWS Lambda가 실행되면 작성한 코드를 다운로드하고 실행 환경을 구성하는데 이때를 Full cold start 라고 하며 런타임을 준비하는 과정은 Partial cold start 라고 하며 마지막으로 함수가 실행 될 때를 Warm start 라고 함
    - 함수가 실행될 때 순서가 정해져 있고 이에 따라 지연 시간이 발생하는데 Micro VM이 올라갔다가 내려간 뒤 다시 실행하면 Full cold start 부터 시작하고 만약 Micro VM이 유지되는 시간 이전에 재요청하면 Partial cold start가 바로 진행되어 지연 시간을 줄일 수 있음
    - Cold Start를 줄이기 위해서는 지속적인 호출을 통해 구동되어 있는 Micro VM을 유지하는 것인데 이를 보통 5분 마다 호출하라고 권고하지만 실제로 각자 구현한 람다의 환경이 다르기에 정확한 수치가 아님

      https://theburningmonk.com/2017/06/aws-lambda-compare-coldstart-time-with-different-languages-memory-and-code-sizes/
    - cold start time은 언어/런타임, 리소스 양(mb) 및 함수 실행 시 갖고 오는 패키지/종속성에 따라 다름

ServerLess

- Runtime
  - ✓ AWS Lambda에서는 Runtime을 사용하여 각각 다른 언어들로 작성한 코드도 동일한 기본 실행 환경에서 실행할 수 있음
  - ✓ Runtime은 AWS Lambda 서비스 와 작성한 함수 코드 사이에 위치해 이벤트와 컨텍스트 정보 등 응답을 중계해주는 역할을 수행
  - ✓ 기본적으로 제공하는 런타임 환경이 있고 직접 필요한 런타임 환경을 만들어 사용할 수 있음
  - ✓ 제공하는 런타임 환경에는 여러 언어를 지원
  - ✓ 함수를 생성할 때 런타임을 선택할 수 있고 필요시 런타임 환경을 변경할 수 있음
  - ✓ 런타임 환경은 Amazon Linux
  - ✓ 제공하는 런타임 환경에서는 각 언어의 버전 별 또는 운영체제의 수명이 60일 이내로 예정될 때 마이그레이션 안내 메일을 발송해 줌
  - ✓ 런타임 지원이 중단되면 AWS Lambda는 호출을 비활성화 하는데 사용 중단된 런타임은 보안 업데이트 기술 지원을 받을 수 없기에 마이그레이션이 필수적
  - ✓ 현재는 Node, Python, Java, Ruby, Go, .NET 런타임을 제공

ServerLess

- Event
  - ✓ 호출 방법

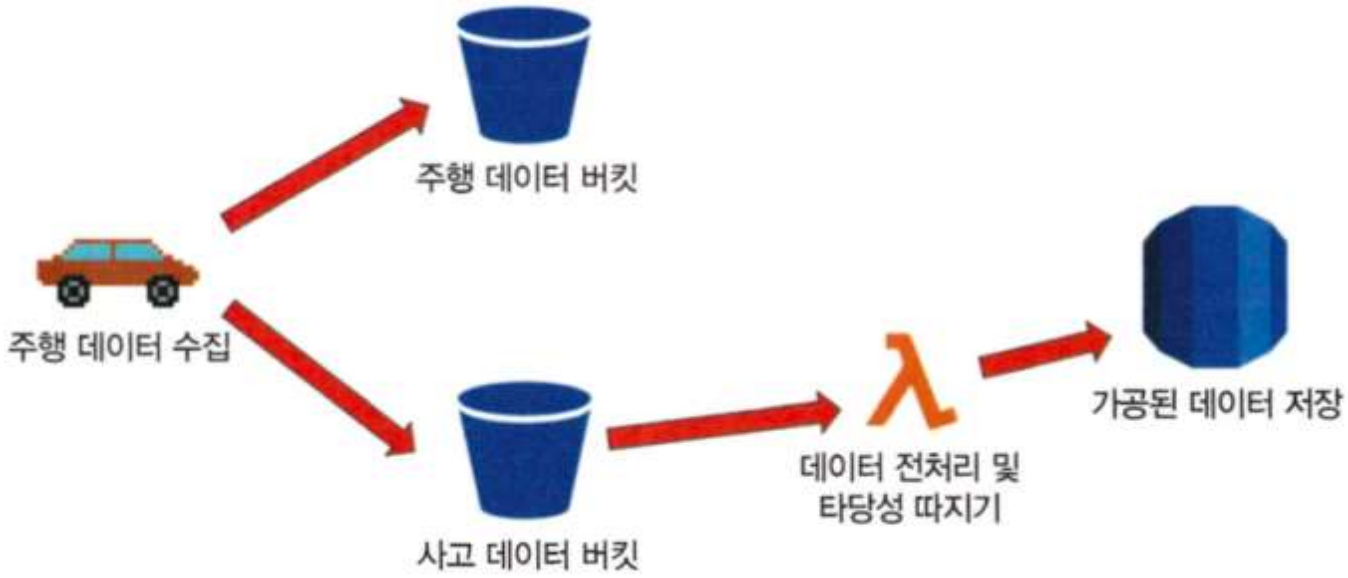
동기적으로 호출하는 서비스	비동기적으로 호출하는 서비스
Elastic Load Balancing (Application Load Balancer)	Amazon Simple Storage Service
Amazon Cognito	Amazon Simple notification Service
Amazon Lex	Amazon Simple Email Service
Amazon Alexa	AWS CloudFormation
Amazon API Gateway	Amazon CloudWatch Logs
Amazon CloudFront (Lambda@Edge)	Amazon CloudWatch Events
Amazon Kinesis Data Firehose	AWS CodeCommit
AWS Step Functions	AWS Config
	AWS IoT
	AWS IoT Events
	AWS CodePipeline

ServerLess

- Event
  - ✓ 호출 방법 차이
    - 동기식은 클라이언트가 람다 함수에 이벤트를 보내고 클라이언트는 함수의 응답을 받을 수 있음
    - 동기식으로 호출할 경우 파라미터 값을 invoke로 실행
    - 비동기식은 클라이언트의 요청 이벤트와 람다 함수 사이에 대기열을 이용하는 방식으로 클라이언트는 성공 응답만 받음
    - 비동기식으로 호출할 경우에는 호출 유형 파라미터 값을 Event로 설정해야 함
    - 비동기식으로 진행할 때 오류가 발생하면 계속해서 재시도하는데 동시성 설정이 작게 되어 있을 경우(오류 = 492) 이거나 서버 오류(500번대) 일 때는 이벤트를 대기열로 다시 보내고 이는 최대 6시간 동안 재시도하며 재시도 간격은 최대 5분까지 증가
    - 이벤트는 대기열에서 삭제될 수 있는데 동시성을 어떻게 조절하느냐 에 따라 이벤트를 보장할 수 있는데 대기열은 백업이 되기도 하며 이때는 람다 함수로 전송하기 전에 이벤트가 만료될 수 있으며 이 때문에 이벤트가 삭제될 수 있음

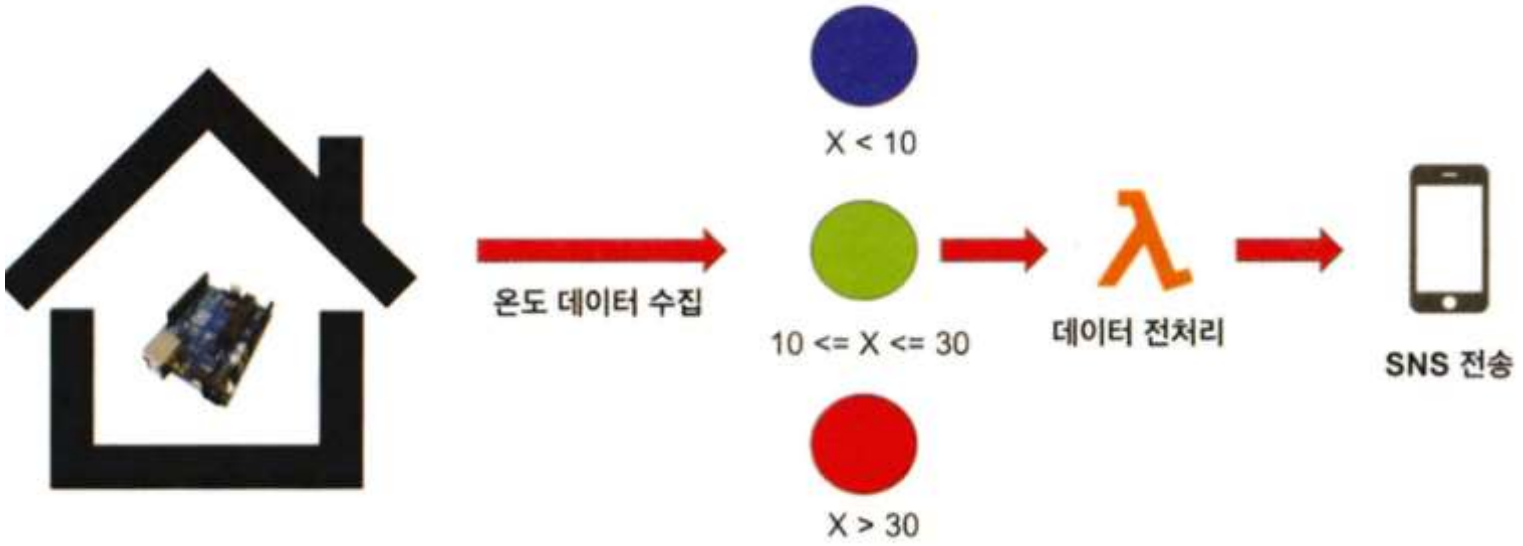
ServerLess

- AWS Lambda 사용 사례
  - ✓ 데이터 전처리



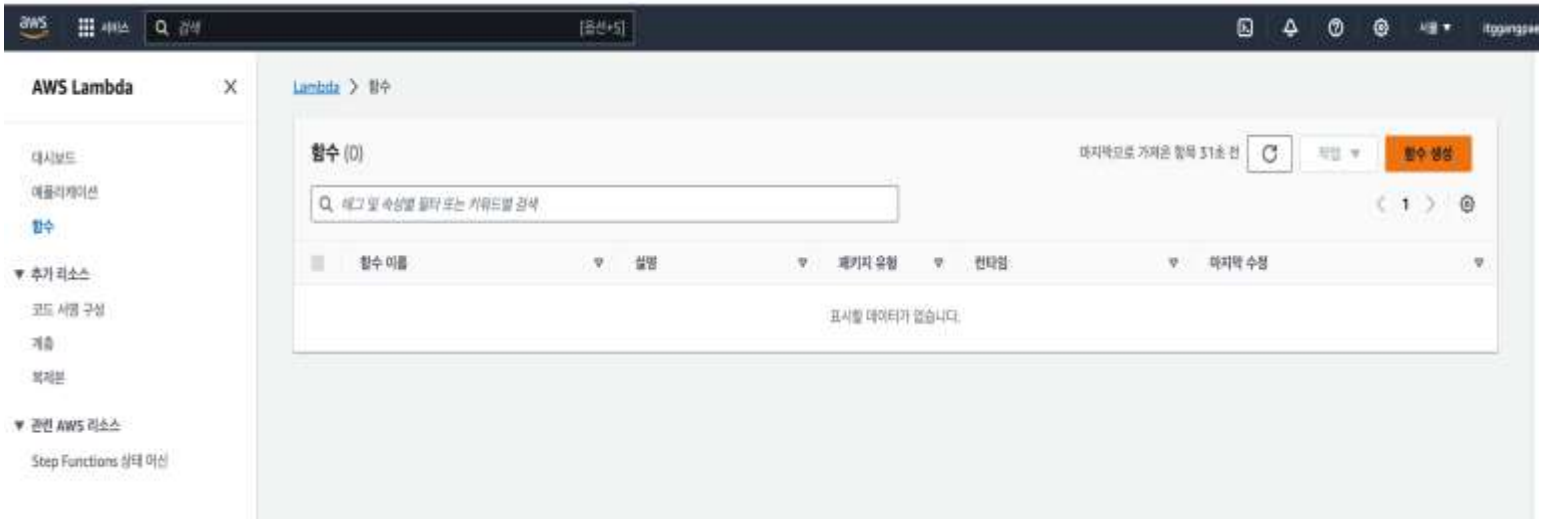
ServerLess

- AWS Lambda 사용 사례
  - ✓ 데이터 전처리



ServerLess

- 람다 함수 만들기
  - ✓ 람다 콘솔에서 오른쪽 상단의 함수 생성 클릭



ServerLess

❖ 람다 함수 만들기

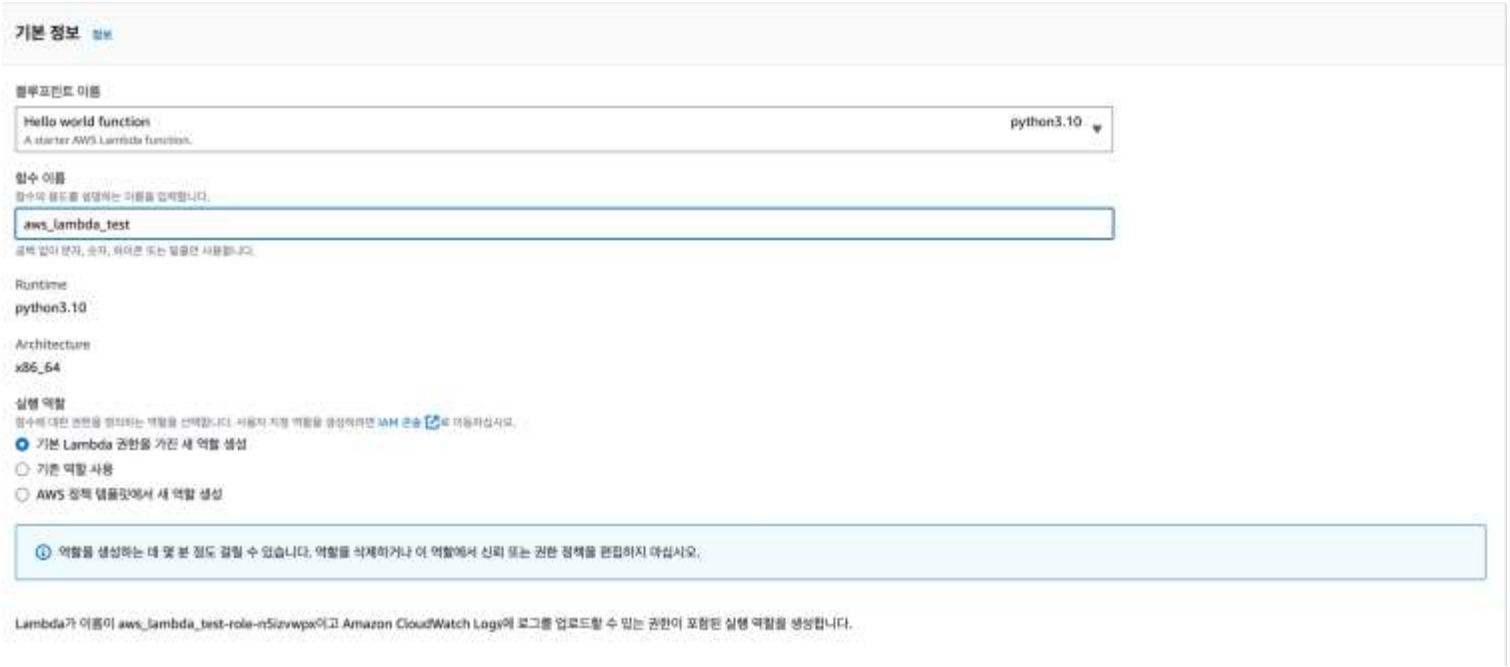
✓ 생성 방법 선택

- 새로 작성: 간단한 로직이 구현된 코드를 기반으로 작성
- 블루프린트 사용: AWS에서 자주 사용되는 기능을 고려하여 개발자를 위한 블루프린트를 만들어서 제공
- 컨테이너 이미지: AWS에서 운영하는 가상 컨테이너 이미지 저장소 경로를 이용해서 배포하고자 하는 경우 사용

ServerLess

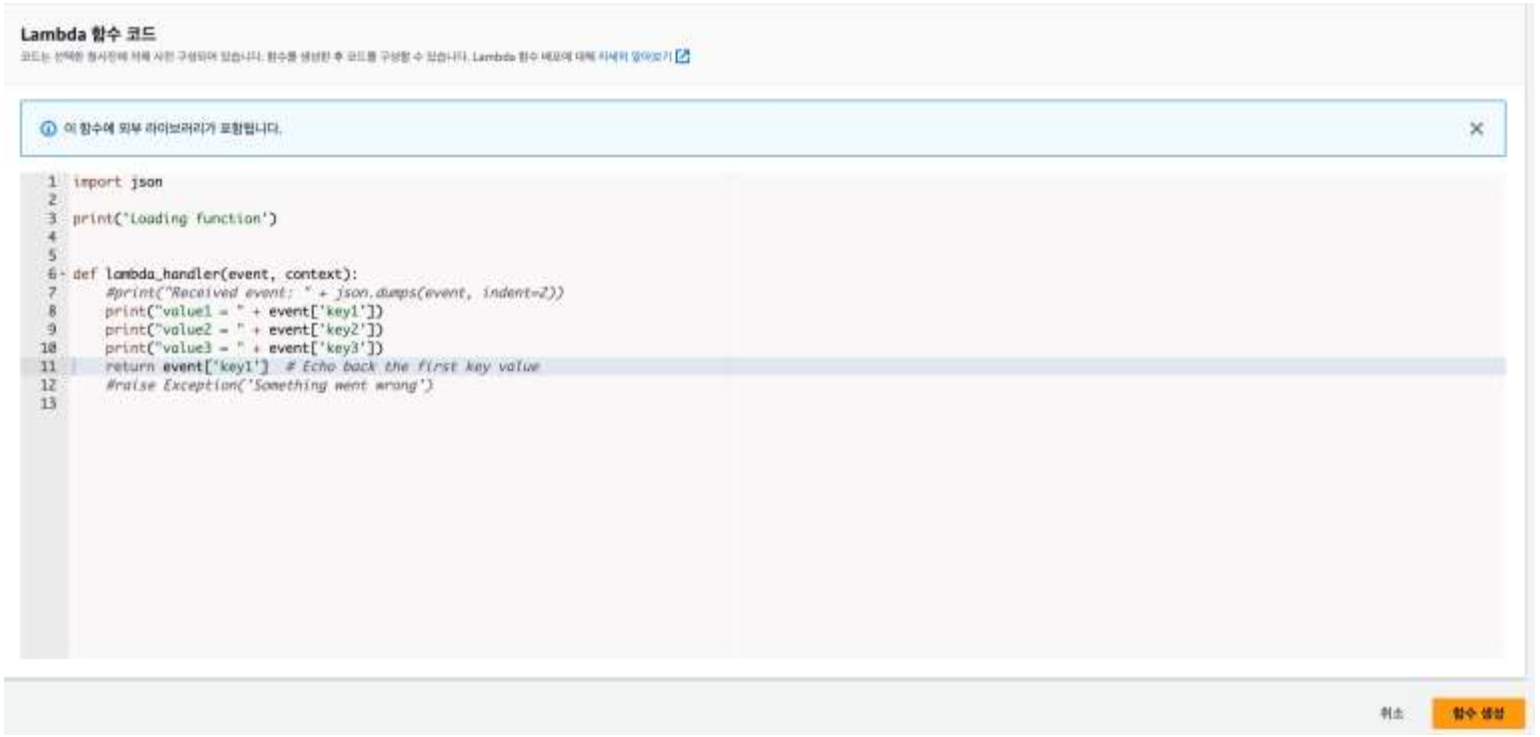
❖ 람다 함수 만들기

✓ 블루프린트를 선택하고 hello-world-python을 선택하고 이름 입력



ServerLess

- 람다 함수 만들기
  - ✓ 함수 코드를 확인하고 함수 생성을 클릭



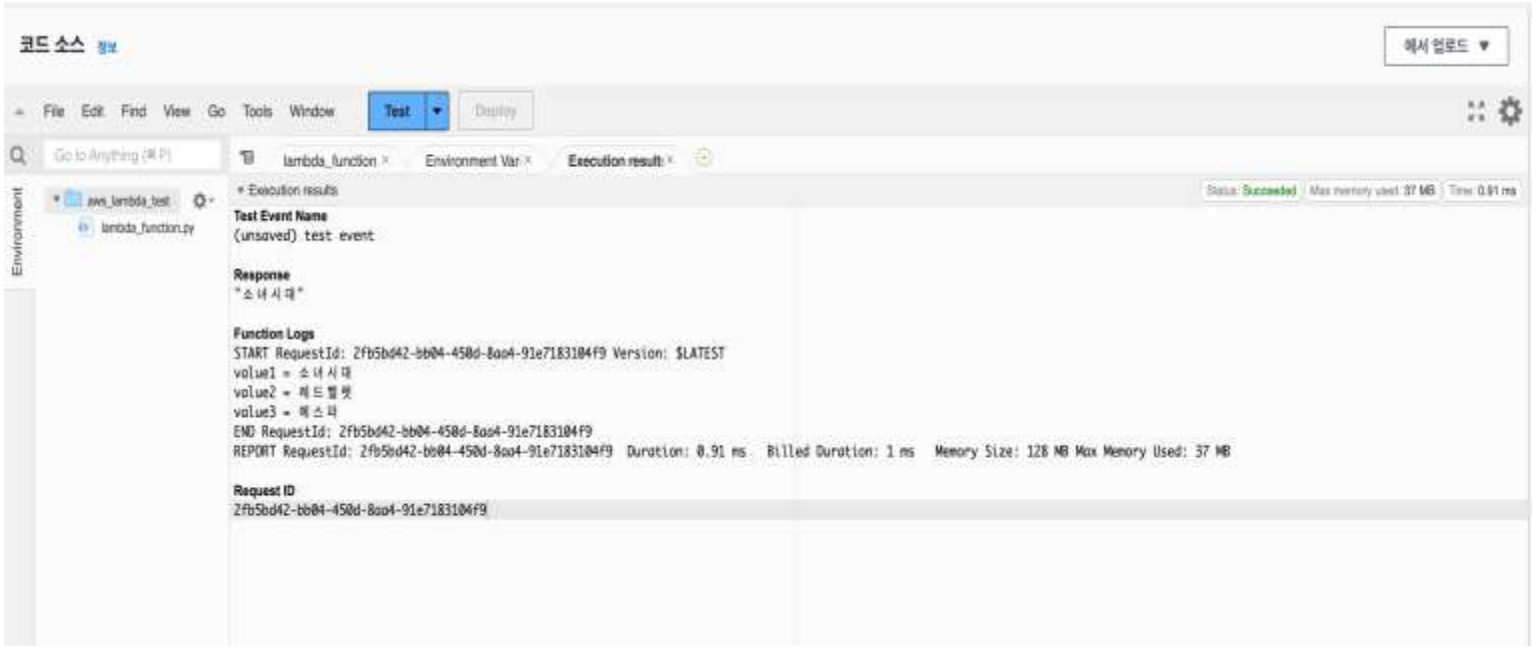
ServerLess

- ❖ 람다 함수 만들기
  - ✓ 코드 소스 창에서 Test를 클릭하고 옵션을 설정한 후 간접 호출 클릭

ServerLess

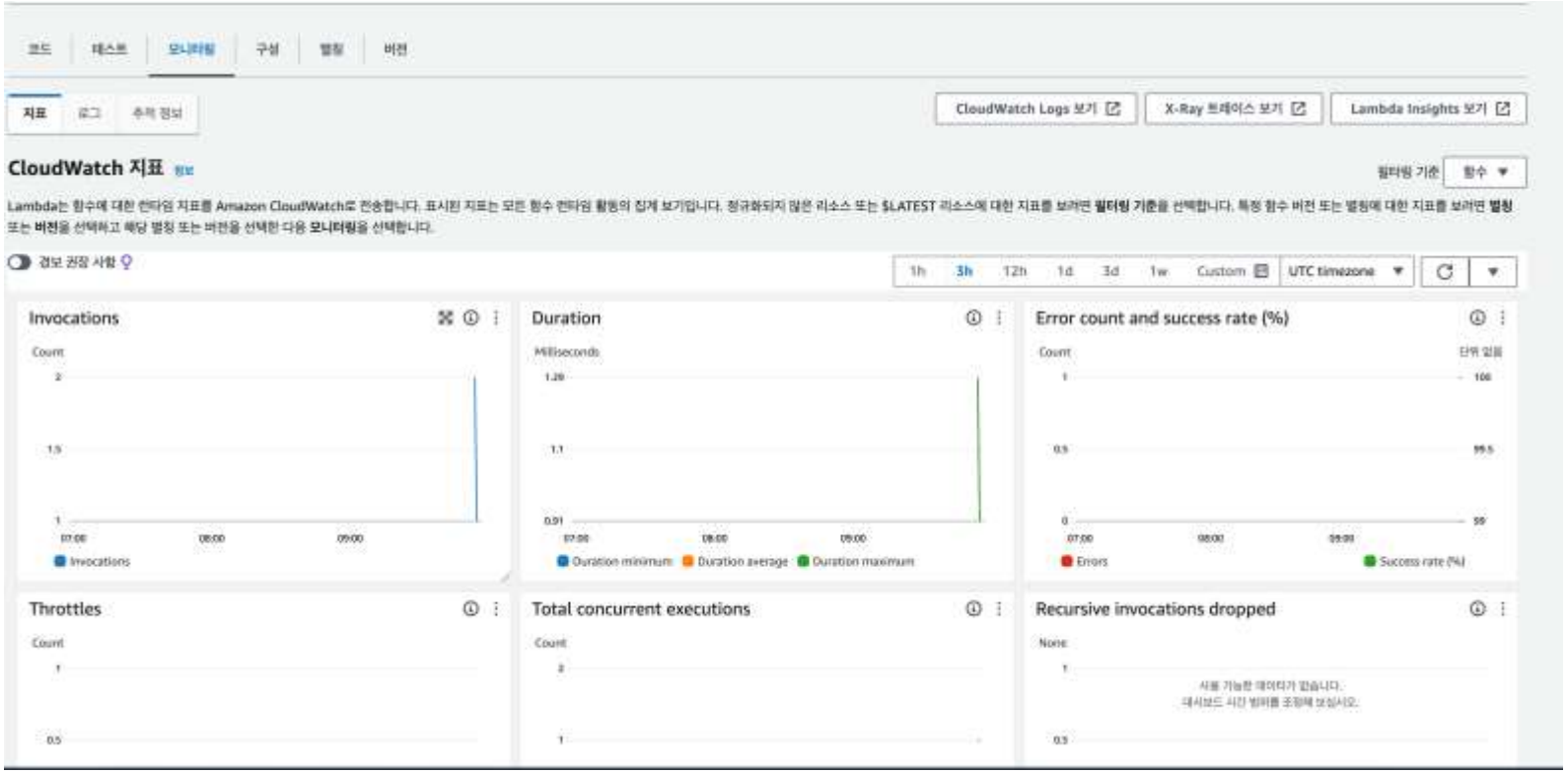
❖ 람다 함수 만들기

✓ 코드 소스 창에서 Test를 클릭하고 옵션을 설정한 후 간접 호출 클릭



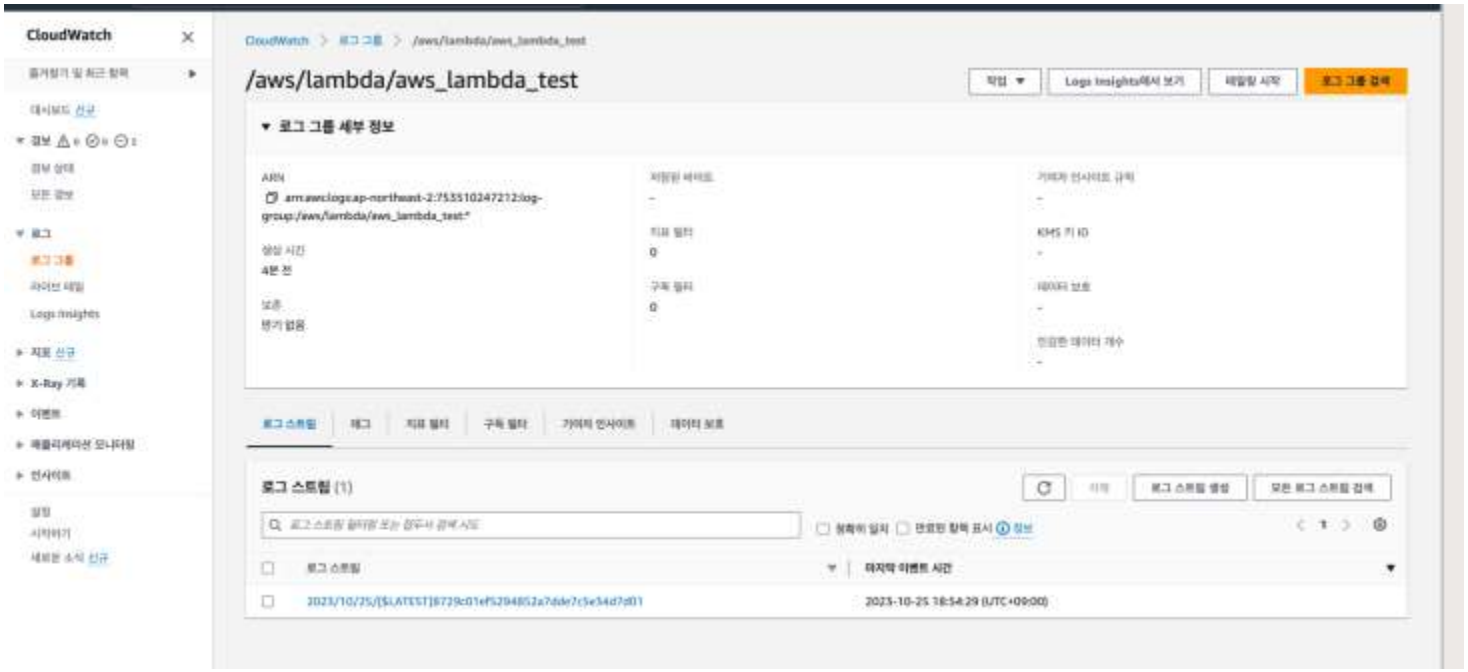
ServerLess

- ❖ 람다 함수 만들기
  - ✓ 클라우드 와치에서 확인
    - ● 모니터링 탭에서 CloudWatch에서 로그 보기를 클릭



ServerLess

- ❖ 람다 함수 만들기
  - ✓ 클라우드 와치에서 확인
    - ● 모니터링 탭에서 CloudWatch에서 로그 보기를 클릭



ServerLess

- ❖ 람다 함수가 람다 함수 호출하기
  - ✓ 람다 함수 2개 생성
    - 첫번째 함수 – lambda_a

```
import json

def lambda_handler(event, context):
    print(event)
    return event
```

ServerLess

- ❖ 람다 함수가 람다 함수 호출하기
  - ✓ 람다 함수 2개 생성
    - ● 두번째 함수 – lambda_b

```
import json
import boto3


def lambda_handler(event, context):
    payload = {}
    payload['hello'] = 'hi'
    lan = boto3.client(service_name='lambda',region_name='ap-northeast-2')
    lan.invoke(FunctionName="lambda_a", InvocationType='Event',
Payload=json.dumps(payload))
    print(payload)
    return payload
```

ServerLess

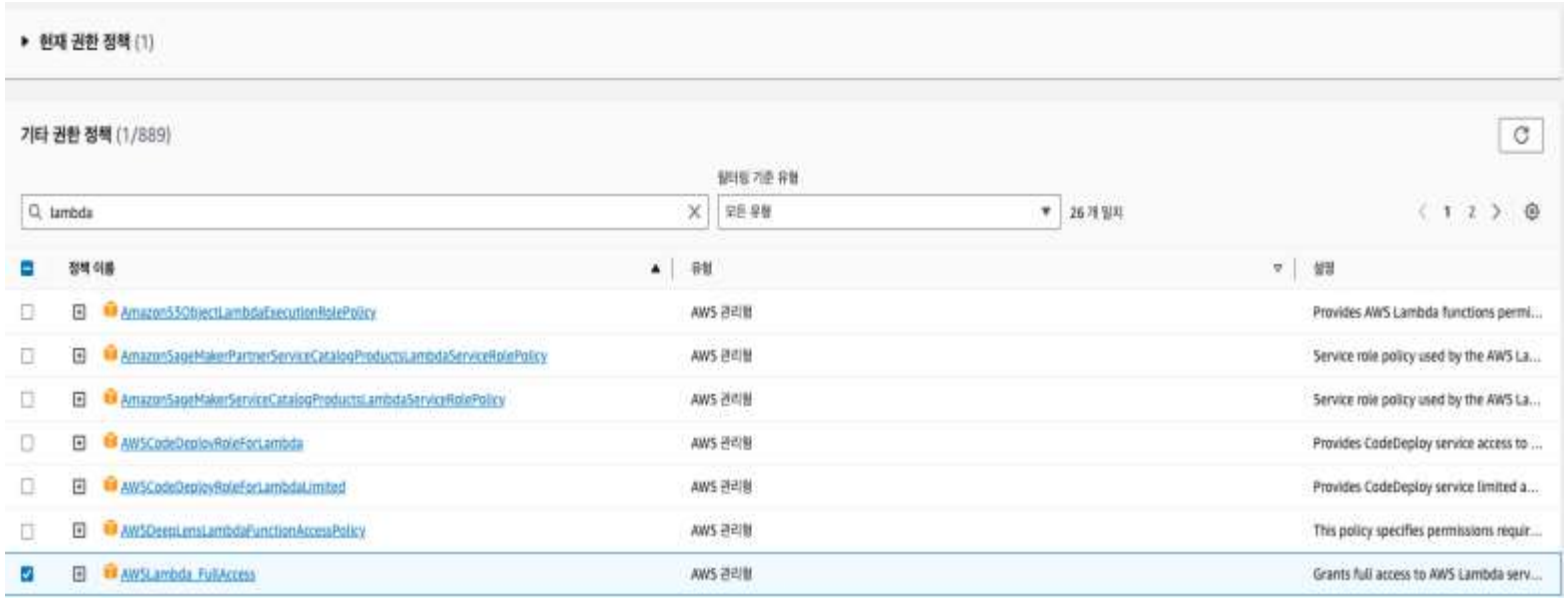
- 람다 함수가 람다 함수 호출하기
  - ✓ lambda_b의 Test를 눌러서 실행 – 권한이 없다는 오류 발생

ServerLess

- 람다 함수가 람다 함수 호출하기
  - ✓ lambda_b에 람다 권한 부여: [구성] – [권한]에서 실행 역할에서 이름을 클릭

ServerLess

- ❖ 람다 함수가 람다 함수 호출하기
  - ✓ lambda_b에 람다 권한 부여: [권한 추가]에서 정책 연결 클릭 후 람다 권한 추가

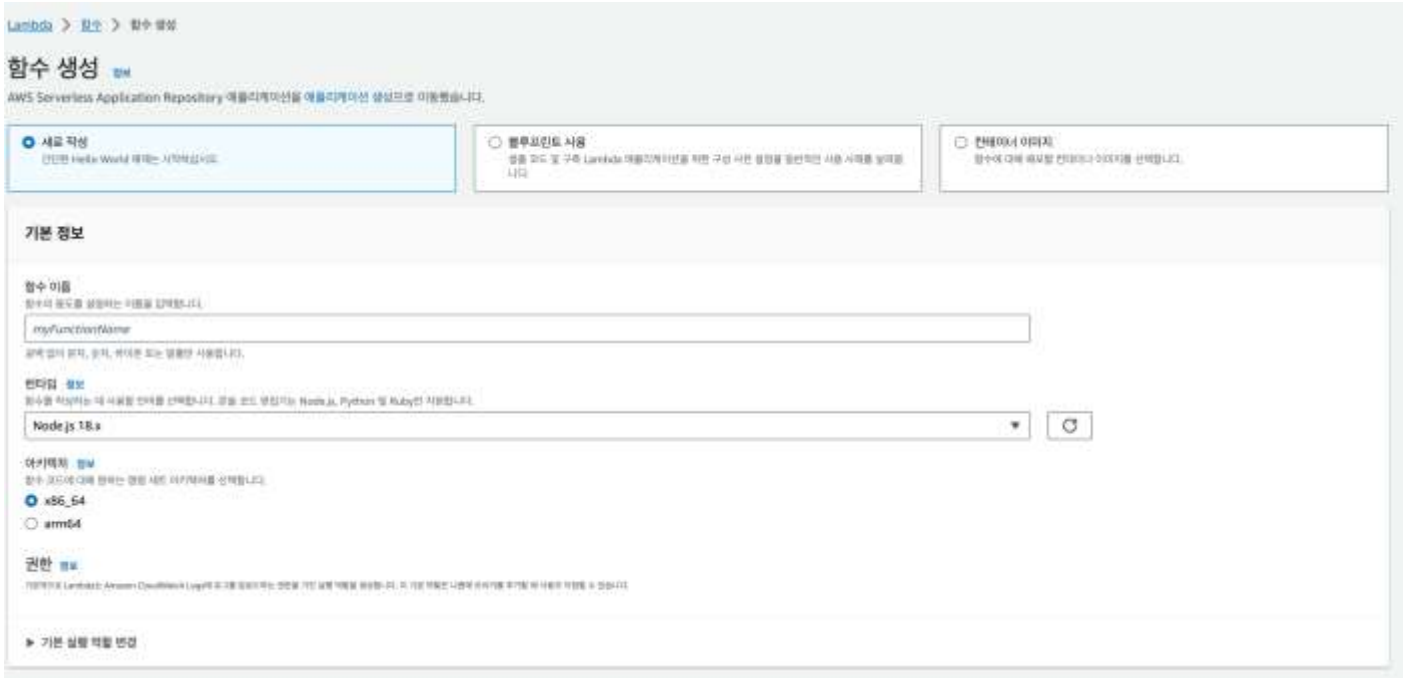


ServerLess

- 람다 함수가 람다 함수 호출하기
  - ✓ lambda_b에 다시 테스트 수행

ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 파이썬에서 실행되는 함수 생성



ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 코드 작성 후 deploy

```
import json
import boto3
from datetime import datetime

client = boto3.client('s3')
```

ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 코드 작성 후 deploy

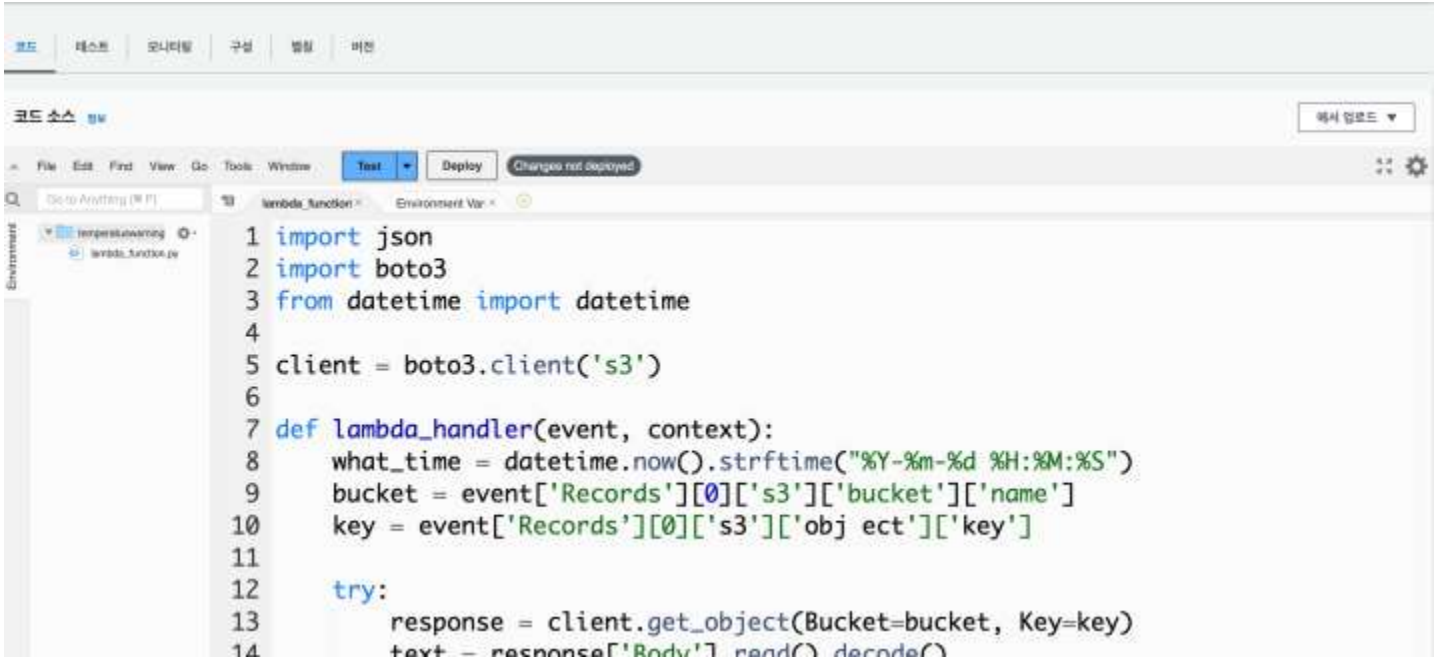
```
def lambda_handler(event, context):
    what_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    try:
        response = client.get_object(Bucket=bucket, Key=key)
        text = response['Body'].read().decode()
        data = json.loads(text)

        if data['temperature'] > 40:
            print(f" {data['temperature']} at {what_time}")
            print("주의하세요 매우 덥습니다.")
        else:
            print("나쁘지 않은 날씨")
    except Exception as e:
        print(e)
        raise e
```

ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 코드 작성 후 deploy



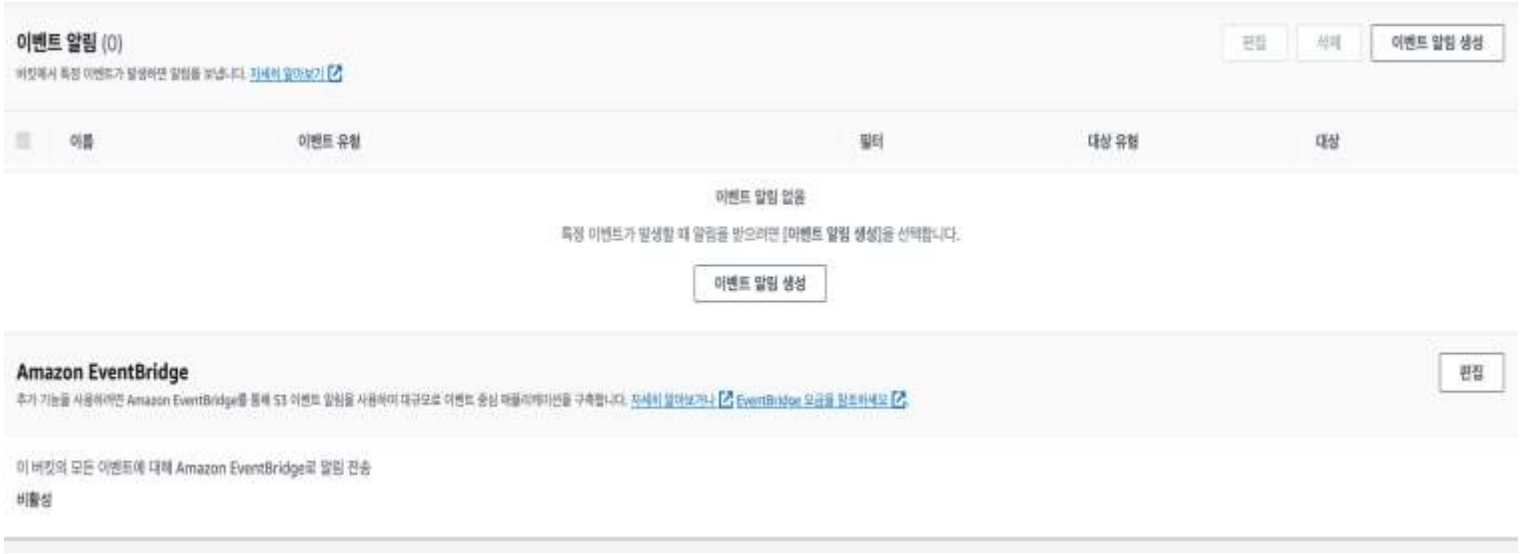
ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ S3 버킷 생성 후 [권한] – [버킷 권한 설정]에서 버킷 정책 추가

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicListGet",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:List*",
                "s3:Get*",
                "s3:Put*",
                "s3:Delete*"
            ],
            "Resource": [
                "arn:aws:s3:::adamsoft",
                "arn:aws:s3:::adamsoft/*"
            ]
        }
    ]
}
```

ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 버킷에서 속성 탭을 열어서 이벤트 알림 생성을 클릭

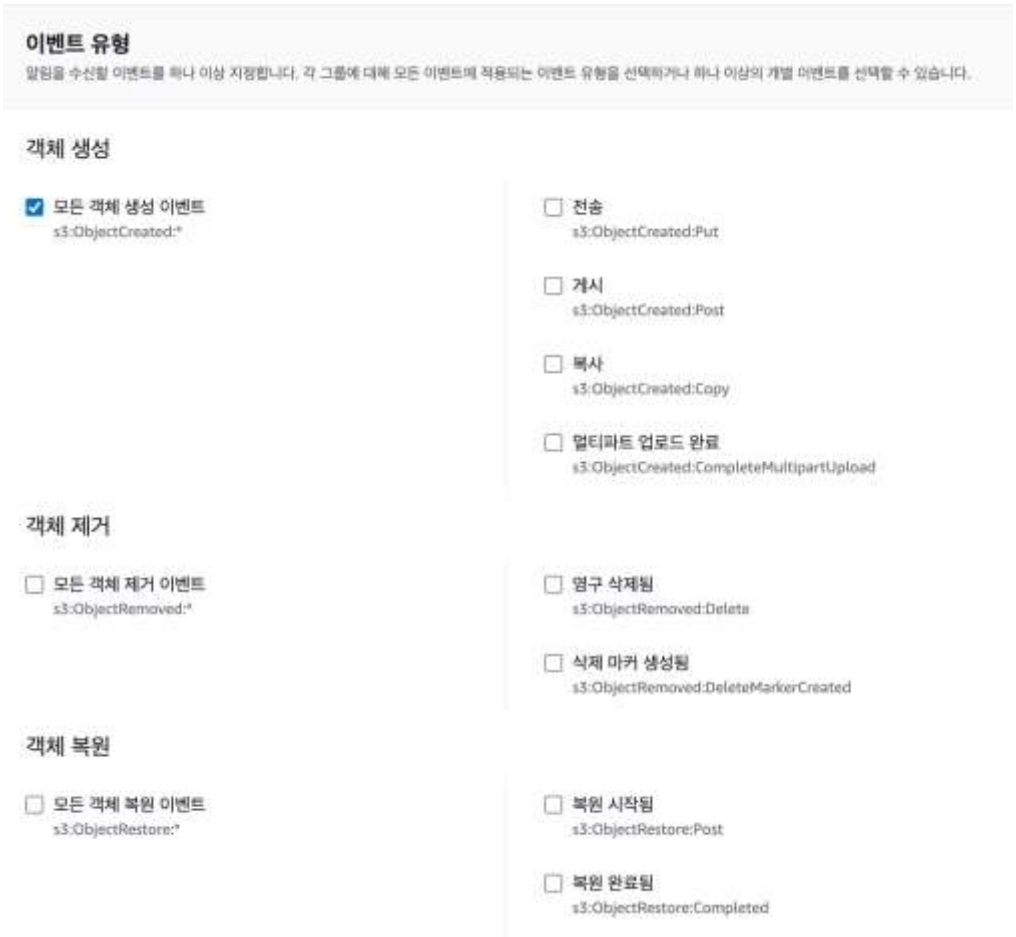


ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 이벤트 설정

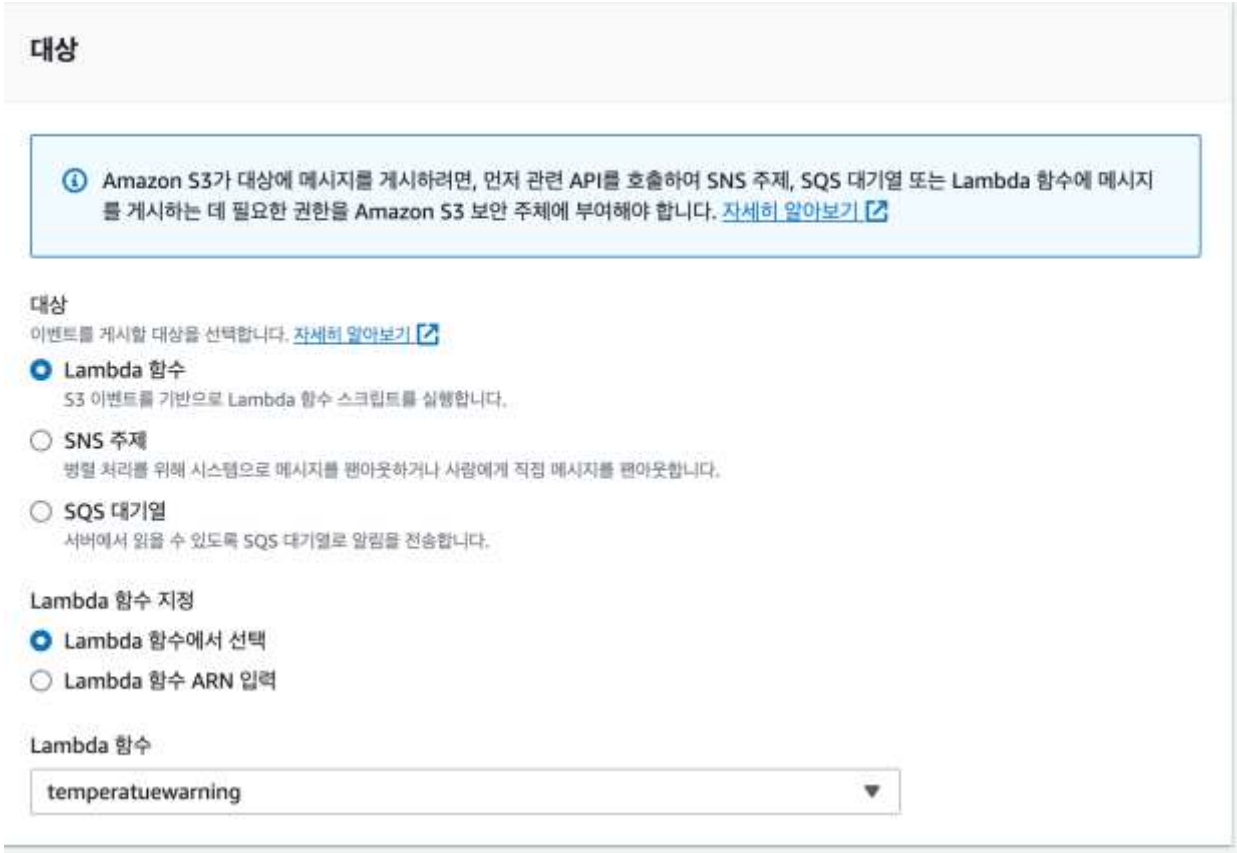
ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 이벤트 설정



ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 이벤트 설정



ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 이벤트 설정

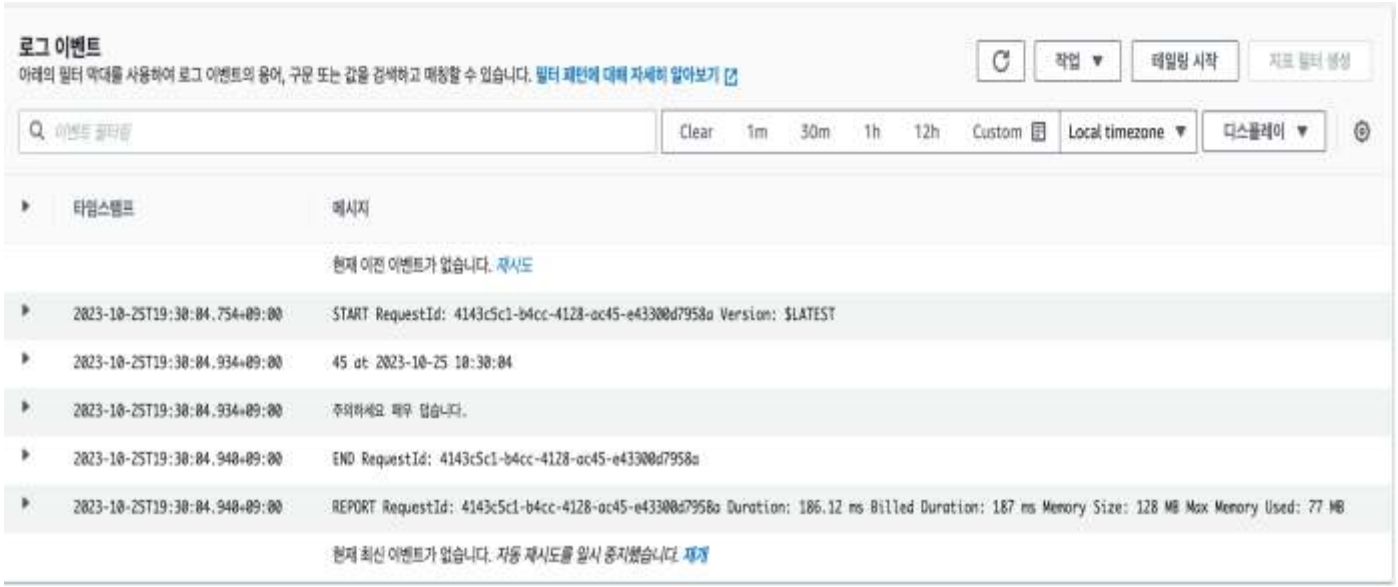
ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 테스트
    - json 파일을 만들어서 업로드

```
{
    "temperature":45
}
```

ServerLess

- S3 버킷에 확장자가 json 데이터가 업로드되면 PutObject 이벤트가 발생하게 되고 이 이벤트가 발생했을 때 데이터 중에서 temperature 값을 읽어서 그 시간 과 메시지를 출력하도록 하기
  - ✓ 테스트
    - json 파일을 만들어서 업로드
      - Cloud Watch에서 로그 확인



ServerLess

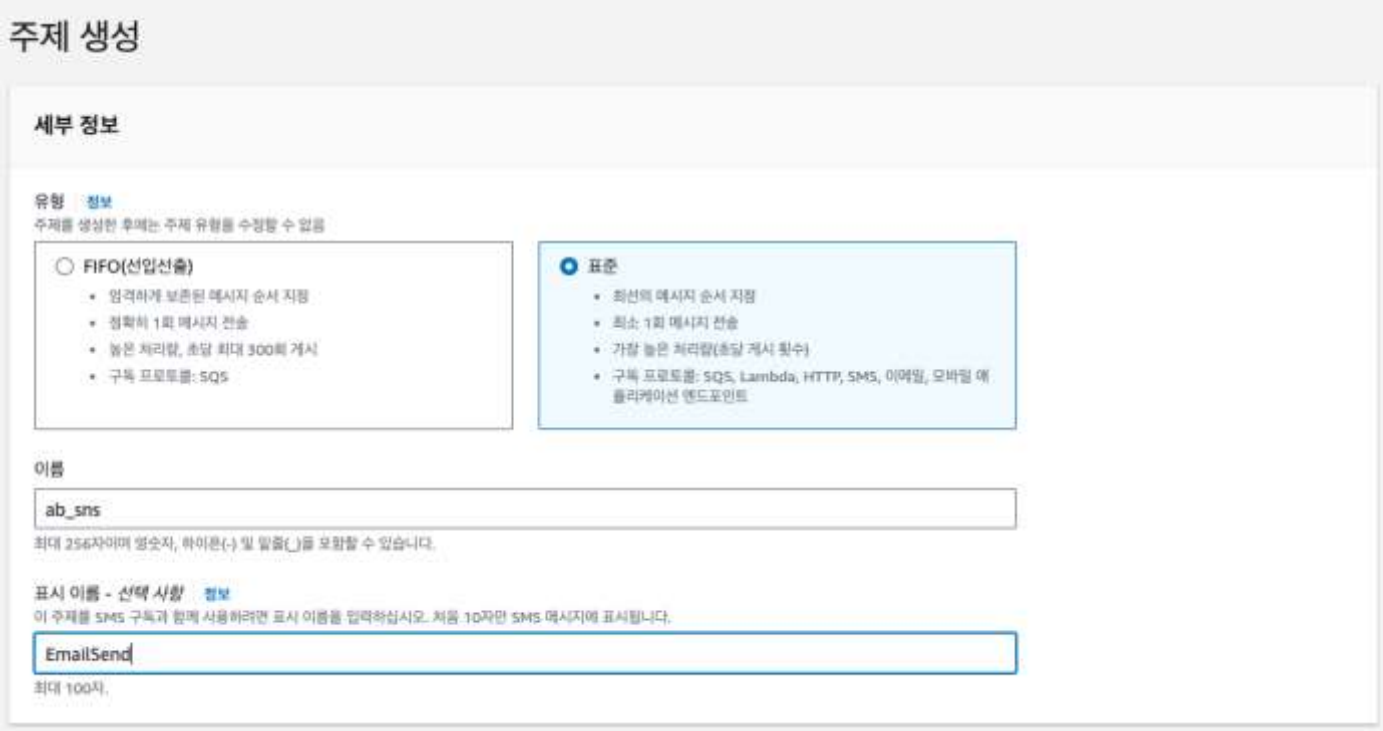
- ❖ Lambda에서 Amazon SNS 실행
  - ✓ 람다 함수 생성
    - ● 메시지를 전송하는 함수 – sns_a

```
import json
import boto3

def lambda_handler(event, context):
    client = boto3.client(service_name='sns')
    response = client.publish(
        TargetArn = '', #'arn:aws:sns:ap-northeast-2:1234567890:ab_sns',
        Message = 'hello',
        Subject = 'sns_a lambda',
        MessageStructure='text'
    )
    print(response)
    return response
```

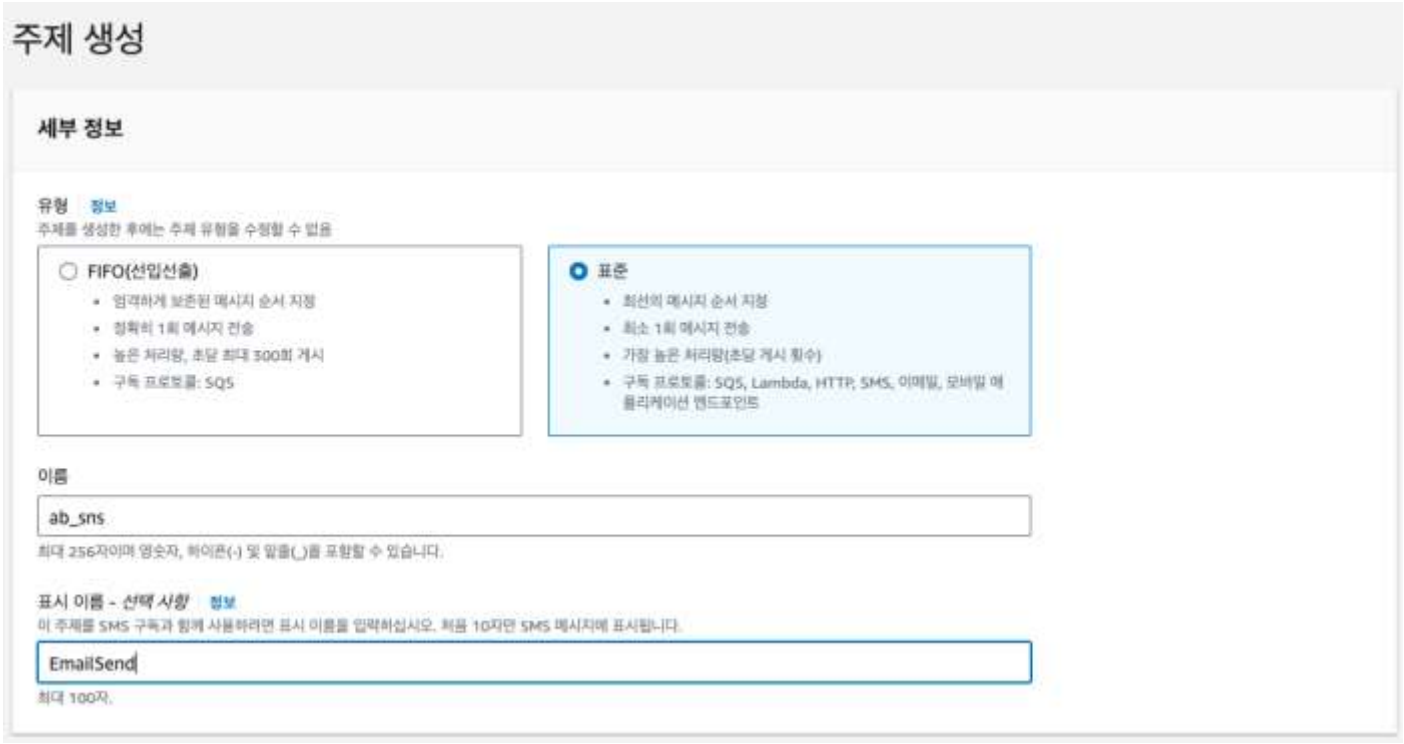
ServerLess

- ❖ Lambda에서 Amazon SNS 실행
  - ✓ SNS 주제 생성
    - ● 주제 설정



ServerLess

- ❖ Lambda에서 Amazon SNS 실행
  - ✓ SNS 주제 생성
    - ● 구독 생성



ServerLess

- Lambda에서 Amazon SNS 실행
  - ✓ SNS 주제 생성
    - 수신을 받는 메일에서 확인
    - 구독 생성 – ARN 복사

ServerLess

❖ Lambda에서 Amazon SNS 실행

✓ SNS 주제 생성

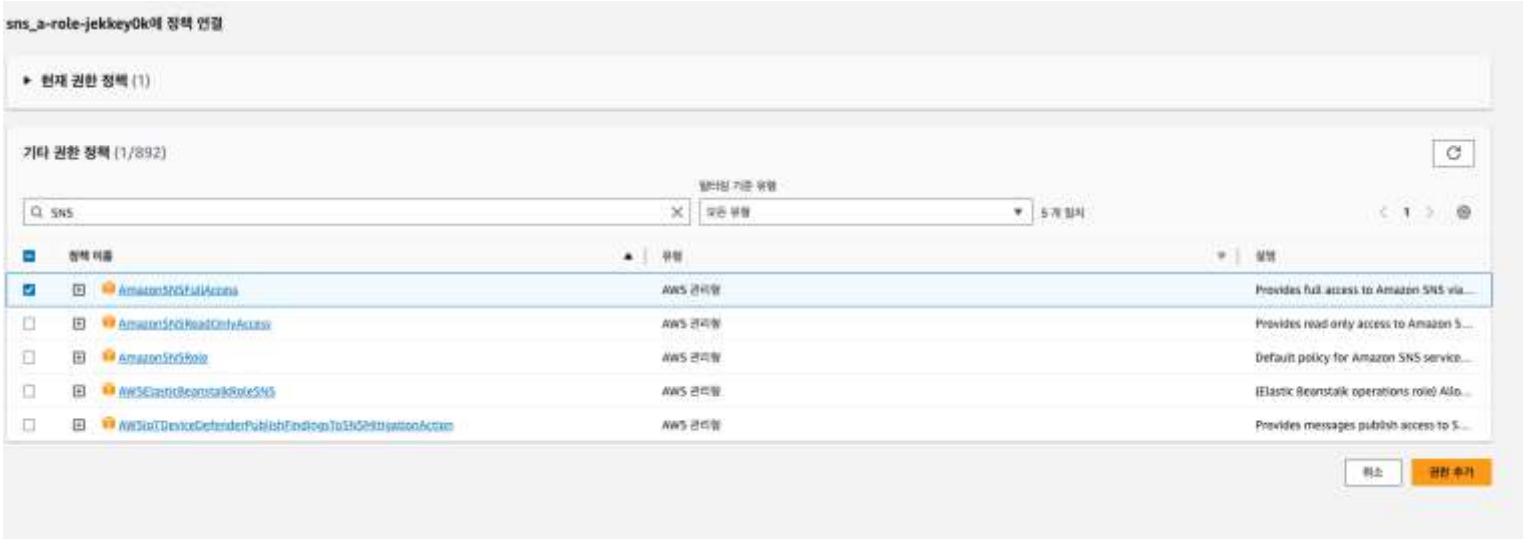
● sns_a 함수 수정

```
import json
import boto3


def lambda_handler(event, context):
    client = boto3.client(service_name='sns')
    response = client.publish(
        TargetArn = 'arn:aws:sns:ap-northeast-2:753510247212:ab_sns',
#'arn:aws:sns:ap-northeast-2:1234567890:ab_sns',
        Message = 'hello',
        Subject = 'sns_a lambda',
        MessageStructure='text'
    )
    print(response)
    return response
```

ServerLess

- ❖ Lambda에서 Amazon SNS 실행
  - ✓ SNS 주제 생성
    - ● sns_a 함수에 SNS 권한 부여

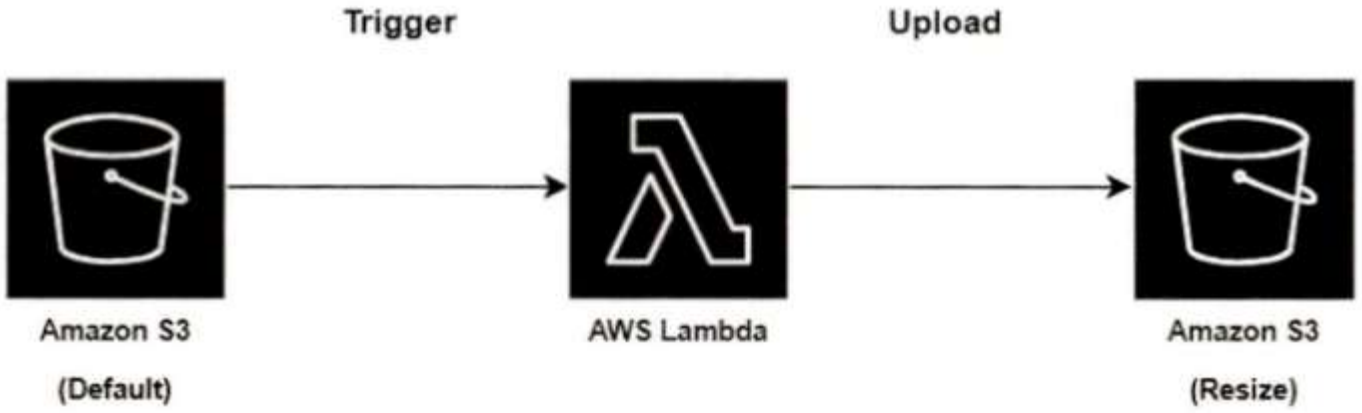


ServerLess

- Lambda에서 Amazon SNS 실행
  - ✓ 함수를 호출한 후 이메일 과 로그 확인

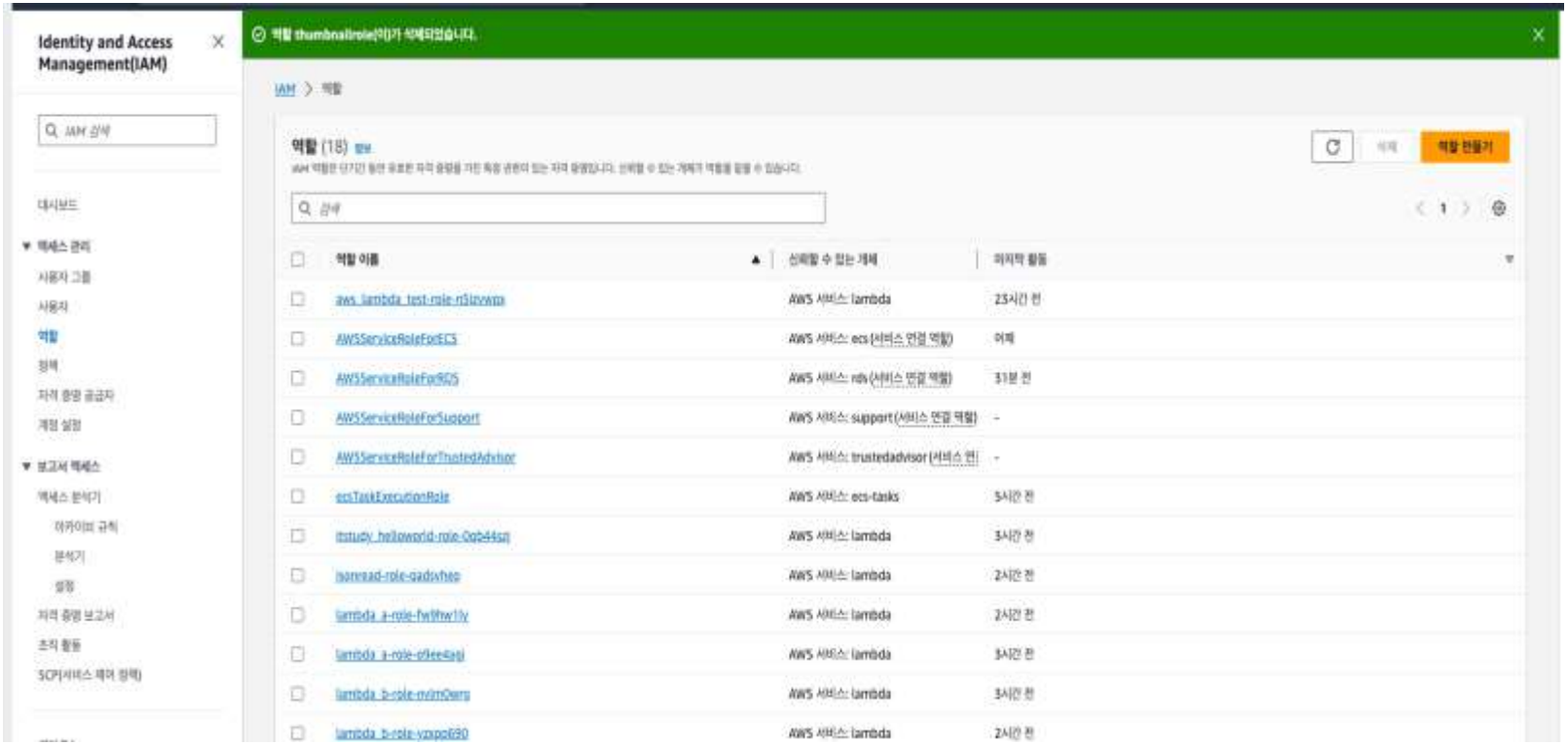
ServerLess

- 썸네일 이미지 만들기
  - ✓ 버킷에 이미지를 업로드하거나 삭제하면 Amazon S3가 AWS Lambda 에 Trigger 하여 다른 S3 버킷에 Thumbnail 이미지를 생성해서 업로드



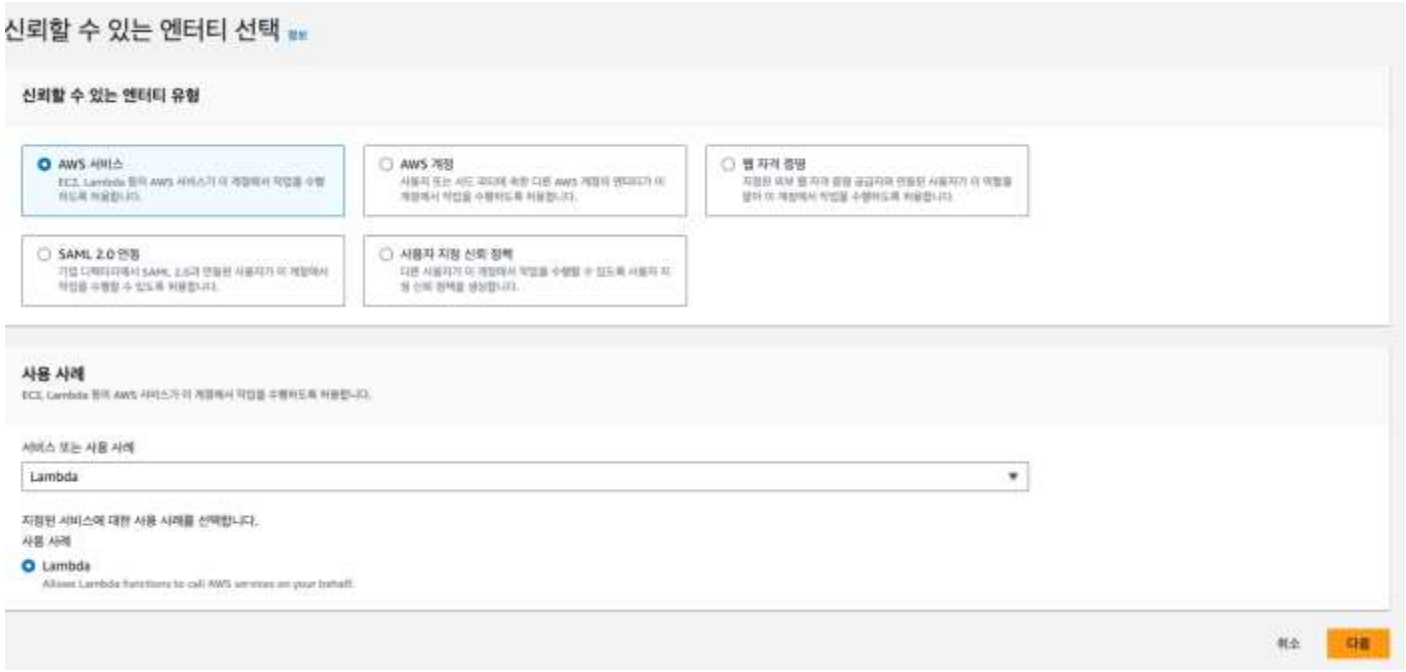
ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 역할 생성 – S3 와 Lambda 실행 권한 소유
    - IAM에서 [엑세스 관리] – [역할] – [역할 만들기]



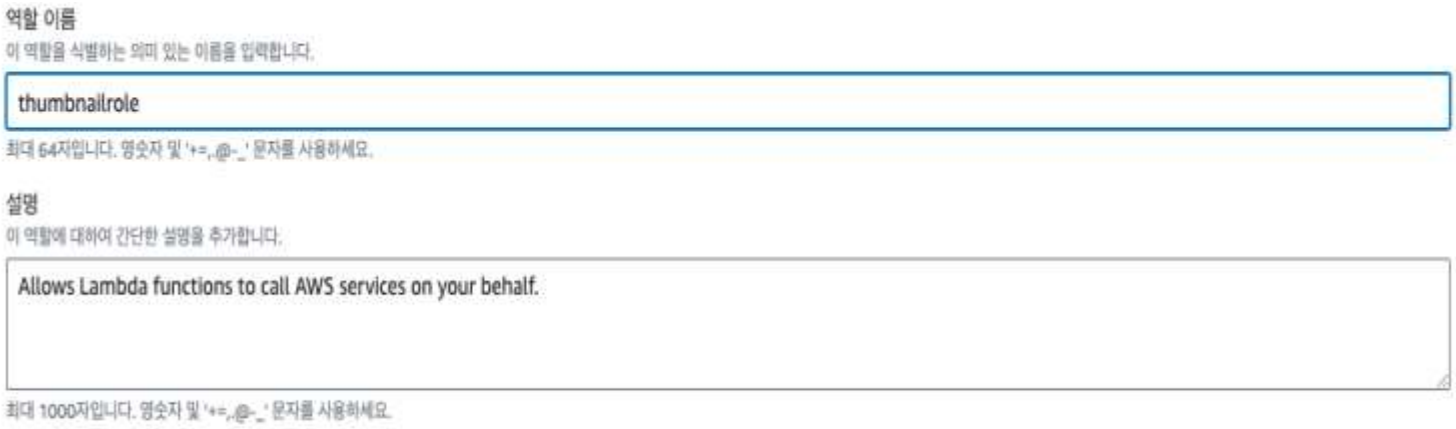
ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 역할 생성 – S3 와 Lambda 실행 권한 소유
    - IAM에서 [엑세스 관리] – [역할] – [역할 만들기]



ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 역할 생성 – S3 와 Lambda 실행 권한 소유
    - IAM에서 [엑세스 관리] – [역할] – [역할 만들기]

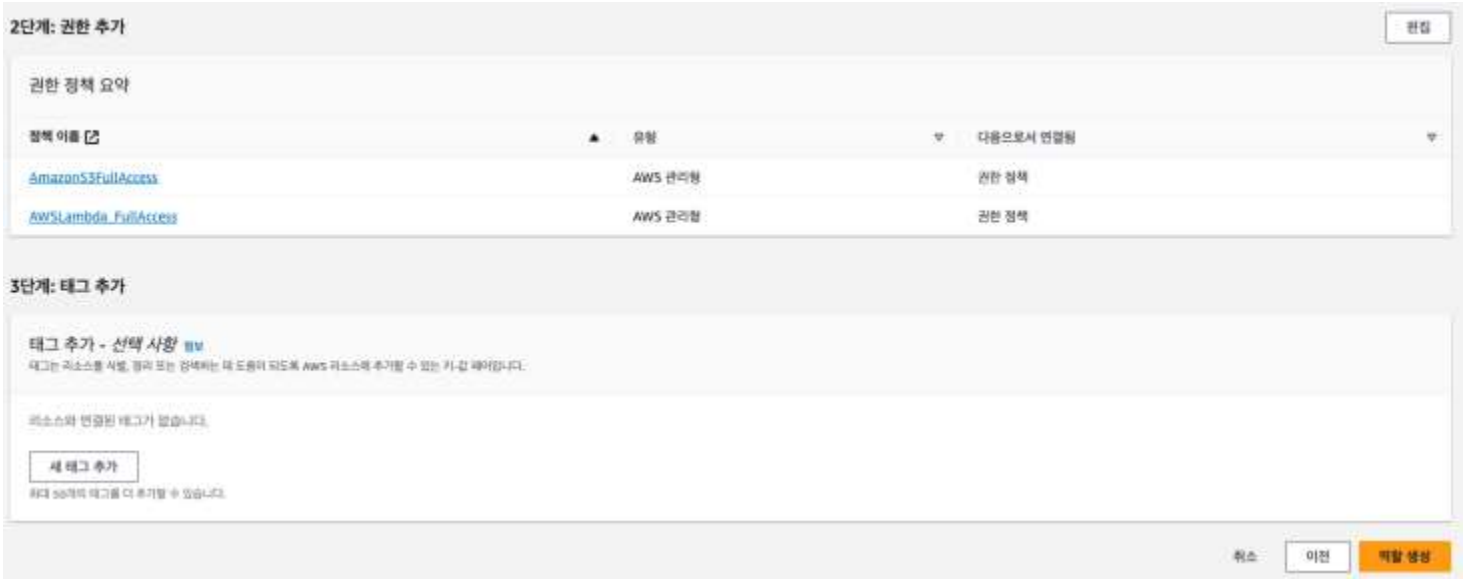


ServerLess

- 썸네일 이미지 만들기
  - ✓ 람다 함수 생성

ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 역할 추가
    - IAM에서 [엑세스 관리] – [역할] – 생성된 람다 함수에 S3 와 Lambda 실행 권한 추가



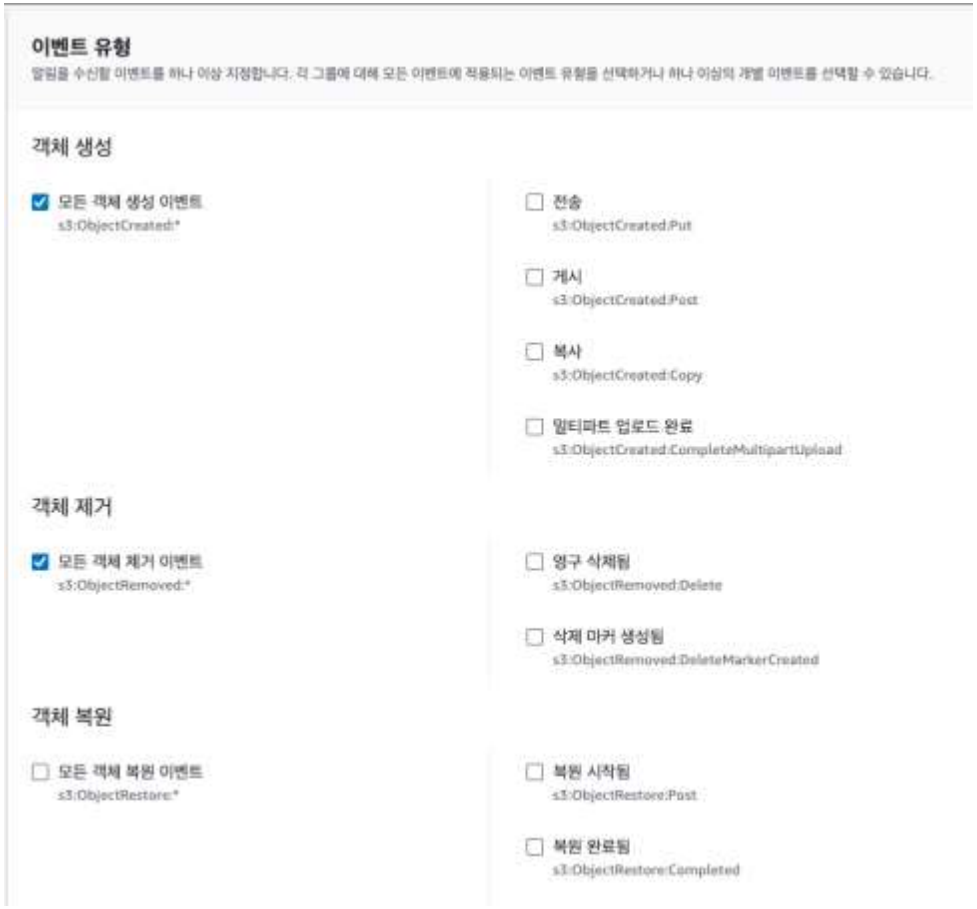
ServerLess

- 썸네일 이미지 만들기
  - ✓ 일반 버킷 생성 – 속성 탭에서 객체가 생성되거나 소멸될 때 람다 함수가 수행되도록 설정

ServerLess

❖ 썸네일 이미지 만들기

✓ 일반 버킷 생성 – 속성 탭에서 객체가 생성되거나 소멸될 때 람다 함수가 수행되도록 설정

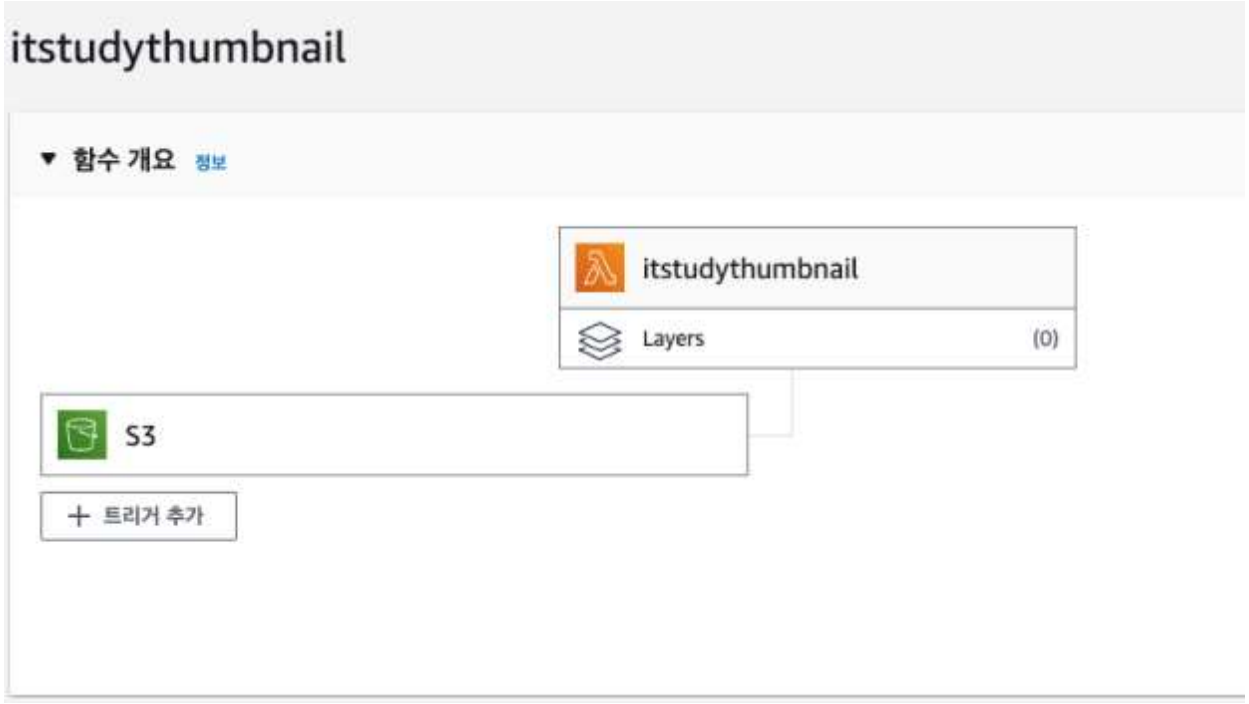


ServerLess

- 썸네일 이미지 만들기
  - ✓ 일반 버킷 생성 – 속성 탭에서 객체가 생성되거나 소멸될 때 람다 함수가 수행되도록 설정

ServerLess

- 썸네일 이미지 만들기
  - ✓ 람다 함수 확인



ServerLess

- 썸네일 이미지 만들기
  - ✓ 이미지를 업로드해서 람다 함수를 실행하고 CloudWatch에서 로그 확인

ServerLess

- 썸네일 이미지 만들기
  - ✓ 썸네일 이미지 버킷 생성 – 이전버킷이름-resized라는 이름으로 생성

ServerLess

- 썸네일 이미지 만들기
  - ✓ 람다 함수 코드 작성

```
import boto3
import os
import sys
import uuid
from urllib.parse import unquote_plus
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)
```

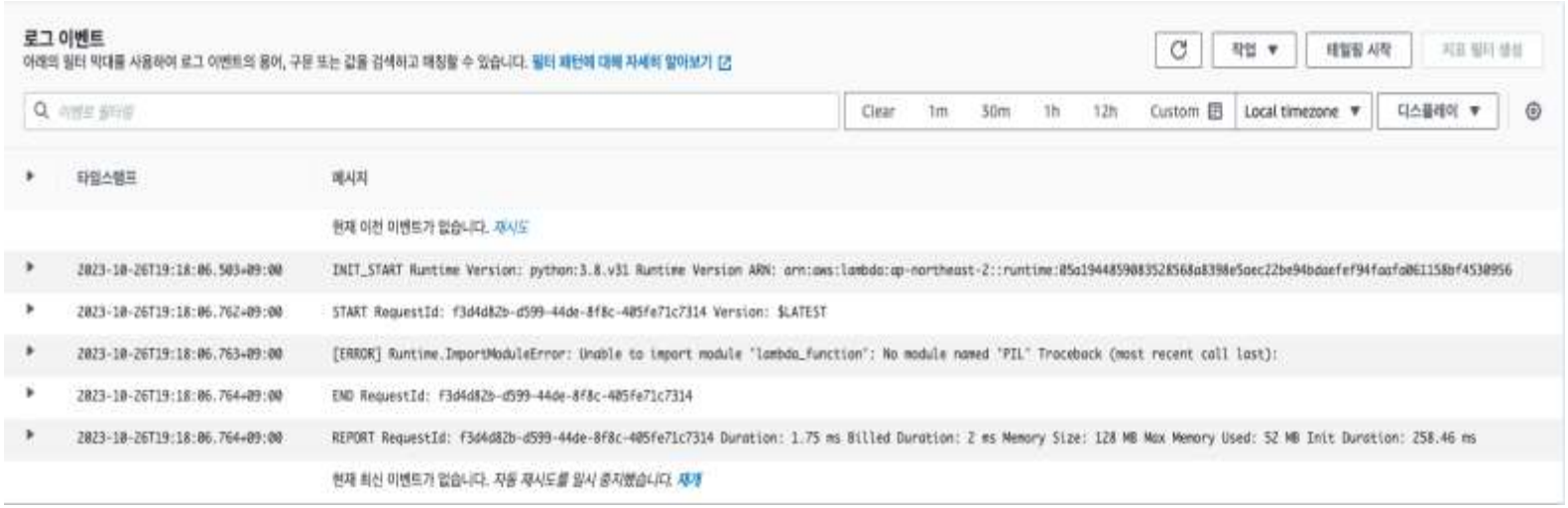
ServerLess

- 썸네일 이미지 만들기
  - ✓ 람다 함수 코드 작성

```
def lambda_handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
        upload_path = '/tmp/resized-{}'.format(tmpkey)
        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}-resized'.format(bucket), key)
```

ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 이미지를 업로드해서 람다 함수를 실행하고 CloudWatch에서 로그 확인
    - ● 모듈이 없다는 에러 발생



ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 도커에서 모듈을 생성해서 추가
    - 도커에서 파이썬 컨테이너를 실행

      docker run --name lambda-img -it lambci/lambda:build-python3.8 bash
    - 배시 셸에서 디렉토리 생성

      mkdir -p opt/python
    - 필요한 패키지 설치

      pip install pillow -t opt/python

      pip install boto3 -t opt/python
    - 필요한 패키지 확인

      cd opt/python

      ls

      exit
    - 패키지를 로컬로 복사

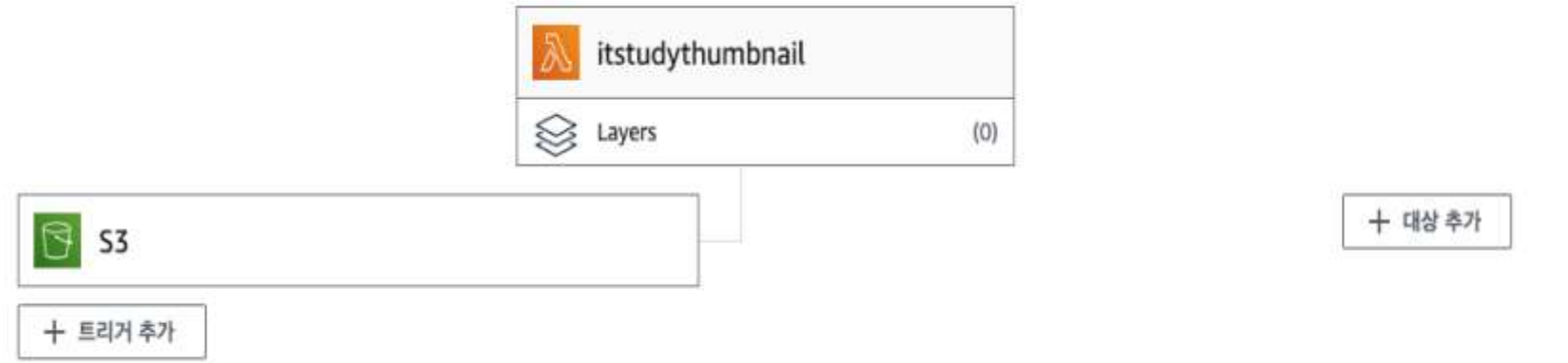
      docker container cp lambda-img:/var/task/opt/python /Users/adam/Documents/dofkerfiledown

ServerLess

- 썸네일 이미지 만들기
  - ✓ 도커에서 모듈을 생성해서 추가
    - 복사된 디렉토리를 zip 파일로 압축

ServerLess

- 썸네일 이미지 만들기
  - ✓ 도커에서 모듈을 생성해서 추가
    - 람다 함수 아래의 Layers를 클릭



ServerLess

- ❖ 썸네일 이미지 만들기
  - ✓ 도커에서 모듈을 생성해서 추가
    - Layer 추가