

DynamoDB+Lambda+S3

- 실습 1 : S3 + Lambda + DynamoDB
  - S3 버킷에 있는 JSON 파일을
  - Lambda가 읽어
  - Dynamo DB에 INSERT
- 실습 2 : CloudWatch log + Lambda + DynamoDB
  - Lambda가
  - CloudWatch log를 조회하여
  - DynamoDB에 저장하기

# 실습1) S3

- 자신의 S3에 json file upload

s3://sgu-202500-3b/dynamo\_data/user\_likes\_data.json

user\_likes\_data.json

```
{  
  "user_id": "haeri05",  
  "product": "워치",  
  "price": 50000,  
  "brand": "애플"  
}
```

```
import json
import boto3
from datetime import datetime
```

```
def lambda_handler(event, context):
    # 클라이언트 및 리소스 생성
    s3 = boto3.client('s3')
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('sgu-202500-user-likes-time')

    # 버킷과 키 지정
    bucket_name = 'sgu-202500-3b'
    object_key = 'dynamo_data/user_likes_data.json'

    try:
        # S3에서 JSON 파일 읽기
        response = s3.get_object(Bucket=bucket_name, Key=object_key)
        content = response['Body'].read().decode('utf-8')
        data = json.loads(content)

        # 타임스탬프 추가
        timestamp = datetime.now().isoformat()
        data['timestamp'] = timestamp

        # DynamoDB 삽입
        table.put_item(Item=data)

    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': 'DynamoDB insert successful!',
            'item': data
        }, ensure_ascii=False)
    }
```

# 실습1) lambda 생성

```
except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({
            'error': str(e)
        }, ensure_ascii=False)
    }
```

# 실습1) 혹시.. 아래와 같은 오류가 나지는 않는지...

Status: Failed  
Test Event Name: hello-world

Response:

```
{  
  "errorType": "Sandbox.Timedout",  
  "errorMessage": "RequestId: 02c16715-384f-4ca7-8428-aba809380543 Error: Task timed out after 3.00 seconds"  
}
```

Function Logs:

START RequestId: 02c16715-384f-4ca7-8428-aba809380543 Version: \$LATEST

END RequestId: 02c16715-384f-4ca7-8428-aba809380543

REPORT RequestId: 02c16715-384f-4ca7-8428-aba809380543 Duration: 3000.00 ms

Billed Duration: 3000 ms Memory Size: 128 MB Max Memory Used: 91 MB Init Duration: 286.35 ms Status: timeout

Request ID: 02c16715-384f-4ca7-8428-aba809380543

# 실습1) 람다 오류 이유는?

- 람다 함수의 기본적인 타임 아웃은 3초!
- S3에서 객체를 읽고! JSON 파싱하고! DynamoDB 쓰는 일련의 과정이 3초 안에 끝나지 않으면 **"Sandbox.Timedout"** 발생!
- 10초로 변경해 보자!

The screenshot shows the AWS Lambda console interface. At the top, there are tabs for '코드' (Code), '테스트' (Test), '모니터링' (Monitoring), '구성' (Configuration), '별칭' (Aliases), and '버전' (Versions). The '구성' (Configuration) tab is selected. On the left sidebar, there are links for '일반 구성' (General configuration), '트리거' (Triggers), '권한' (Permissions), '대상' (Targets), '함수 URL' (Function URL), and '환경 변수' (Environment variables). The main content area is titled '일반 구성 정보' (General configuration information). It contains several fields: '설명' (Description) with a value of '-', '메모리' (Memory) set to '128 MB', '임시 스토리지' (Temporary storage) set to '512 MB', and 'SnapStart' set to 'None'. The '제한 시간' (Timeout) field is set to '0 분 3 초' (0 min 3 sec) and is highlighted with a red box. An '편집' (Edit) button is located in the top right corner of the configuration section and is also highlighted with a red box.

일반 구성	메모리	임시 스토리지
설명 -	128 MB	512 MB
제한 시간 0 분 3 초	SnapStart None	

# 실습1) 람다 제한 시간을 10초로 변경!

## 기본 설정 편집

### 기본 설정 [정보](#)

#### 설명 - 선택 사항

#### 메모리 [정보](#)

구성된 메모리에 비례하는 CPU가 함수에 할당됩니다.

MB

메모리를 128MB~10240MB 범위로 설정

#### 임시 스토리지 [정보](#)

함수에 대해 최대 10GB의 임시 스토리지(/tmp)를 구성할 수 있습니다. [요금 보기](#)

MB

임시 스토리지(/tmp)를 512MB~10240MB 사이로 설정합니다.

#### SnapStart [정보](#)

함수가 초기화된 후 Lambda가 함수의 스냅샷을 캐싱하도록 설정해 시작 시간을 단축합니다. 함수 코드가 스냅샷 작업에 대한 복원력이 있는지 여부를 평가하려면 [SnapStart 호환성 고려 사항](#)을 검토하세요.

None ▼

지원되는 런타임: Java 11, Java 17, Java 21.

#### 제한 시간

분  초

#### 실행 역할

함수에 대한 권한을 정의하는 역할을 선택합니다. 사용자 지정 역할을 생성하려면 [IAM 콘솔](#)로 이동하십시오.

- ☒ 기존 역할 사용  
☐ AWS 정책 템플릿에서 새 역할 생성

#### 기존 역할

생성된 기존 역할 중에 이 Lambda 함수와 함께 사용할 역할을 선택합니다. 이 역할에는 Amazon CloudWatch Logs에 로그를 업로드할 수 있는 권한이 있어야 합니다.

▼



IAM 콘솔에서 [SafeRoleForUser-sgu-202500](#) 역할을 확인합니다.

[취소](#)

[저장](#)

# 실습 1) DynamoDB 결과 확인!

<input type="checkbox"/>	<a href="#">haeri05</a>	2025-05-11T21:00:00	소니	디지털	블랙	89000	무선 이어폰	4.7	최근 블루투스 기기 검색
<input type="checkbox"/>	<a href="#">haeri05</a>	2025-05-26T10:20:23.230371	애플			50000	워치		

—



## 실습1, 번외) 특정 contents만 parsing 하고 싶다면?

- 필요한 key만 파싱하여 insert 하기!
- S3에 아래의 json file 업로드!
- user\_id, product만 insert!

```
{  
  "user_id": "haeri05",  
  "product": "갤럭시워치",  
  "price": 40000,  
  "color": "화이트",  
  "brand": "삼성"  
}
```

```
import json
import boto3
from datetime import datetime
```

```
def lambda_handler(event, context):
    s3 = boto3.client('s3')
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('sgu-202500-user-likes-time')

    bucket_name = 'sgu-202500-3b'
    object_key = 'dynamo_data/user_likes_data_2.json'

    try:
        # 1. S3에서 파일 가져오기
        response = s3.get_object(Bucket=bucket_name, Key=object_key)
        content = response['Body'].read().decode('utf-8')
        raw_data = json.loads(content)

        # 2. 필요한 필드만 추출
        user_id = raw_data.get('user_id')
        product = raw_data.get('product')

        if not user_id or not product:
            return {
                'statusCode': 400,
                'body': '필수 데이터 누락: user_id 또는 product가 없습니다.'
            }

        # 3. DynamoDB에 저장할 항목 구성
        item = {
            'user_id': user_id,
            'timestamp': datetime.now().isoformat(),
            'product': product
        }
```

# 실습1, 번외) lambda 함수

```
# 4. insert
    table.put_item(Item=item)

    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': '선택 필드만 Insert 완료',
            'item': item
        }, ensure_ascii=False)
    }

except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({'error': str(e)}, ensure_ascii=False)
    }
```

# 실습1, 번외2) JSON이 배열 형태로 들어 있다면?

```
[
  {
    "user_id": "haeri01",
    "product": "갤럭시워치",
    "price": 40000,
    "brand": "삼성"
  },
  {
    "user_id": "minji02",
    "product": "아이패드",
    "price": 800000,
    "brand": "애플"
  },
  {
    "user_id": "jihoon03",
    "product": "버즈",
    "price": 120000,
    "brand": "삼성"
  }
]
```

```

import json
import boto3
from datetime import datetime

def lambda_handler(event, context):
    s3 = boto3.client('s3')
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('sgu-202500-user-likes-time')

    bucket_name = 'sgu-202500-3b'
    object_key = 'dynamo_data/user_likes_data.json'

    try:
        # 1. S3에서 파일 읽기
        response = s3.get_object(Bucket=bucket_name, Key=object_key)
        content = response['Body'].read().decode('utf-8')
        data_list = json.loads(content) # JSON 배열로 파싱

        # 2. 각 항목 반복 처리
        for record in data_list:
            user_id = record.get('user_id')
            product = record.get('product')

            if not user_id or not product:
                print(f"user_id 또는 product 누락: {record}")
                continue

            item = {
                'user_id': user_id,
                'timestamp': datetime.now().isoformat(),
                'product': product
            }

            table.put_item(Item=item)
            print(f"Inserted: {item}")

    return {
        'statusCode': 200,
        'body': json.dumps('전체 레코드 Insert 완료', ensure_ascii=False)
    }

```

# 실습1, 번외2) 람다

```

except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({'error': str(e)}, ensure_ascii=False)
    }

```

# 실습2), CloudWatch log + Lambda + DynamoDB

s3에 json 조회했던 람다의 클라우드와치 확인하기!

sgu-202500-lambda-s3

조절

ARN 복사

작업 ▼

함수 개요 정보

다이어그램 | 템플릿



sgu-202500-lambda-s3



Layers

(0)

+ 트리거 추가

+ 대상 추가

Infrastructure Composer로 내보내기

다운로드 ▼

설명

-

마지막 수정

16시간 전

함수 ARN

arn:aws:lambda:ap-northeast-2:443370697536:function:sgu-202500-lambda-s3

함수 URL 정보

-

코드

테스트

모니터링

구성

별칭

버전

Monitor 정보

CloudWatch Logs 보기

애플리케이션 신호 보기

X-Ray 트레이스 보기

Lambda Insights 보기

# 실습2), cloudwatch log

- log 확인

[/aws/lambda/squ-202500-lambda-s3](#) > 2025/05/26/[\$LATEST]f67616bbac654f9b87a97fbd73a29057

## 로그 이벤트

아래의 필터 막대를 사용하여 로그 이벤트의 용어, 구문 또는 값을 검색하고 매칭할 수 있습니다. [필터 패턴에 대해 자세히 알아보기](#)

Q 이벤트 필터링 - Enter 키를 눌러 검색

▶ 타임스탬프	메시지
	현재 이전 이벤트가 없습니다. <a href="#">재시도</a>
▶ 2025-05-26T19:20:19.546+09:00	INIT_START Runtime Version: python:3.13.v40 Runtime Version ARN: arn:aws:lambda:ap-northeast-2::runtime:67df0ac22722a9d5069c1cdb0b24c86bdfd11f9738985ac6f79319e3e026ffa8
▶ 2025-05-26T19:20:19.842+09:00	START RequestId: 030ce44e-e985-4c64-870a-40c2cc68f0ec Version: \$LATEST
▶ 2025-05-26T19:20:23.482+09:00	END RequestId: 030ce44e-e985-4c64-870a-40c2cc68f0ec
▶ 2025-05-26T19:20:23.482+09:00	REPORT RequestId: 030ce44e-e985-4c64-870a-40c2cc68f0ec Duration: 3639.23 ms Billed Duration: 3640 ms Memory Size: 128 MB Max Memory Used: 93 MB Init Duration: 292.29 ms
	현재 최신 이벤트가 없습니다. 자동 재시도를 일시 중지했습니다. <a href="#">재개</a>

## 실습2) cloudwatch json 구조

- cloudwatch 로그 이벤드 구조는 기본 3개의 필드

```
{  
  "timestamp": 1716728459247,      // 로그 발생 시각 (Epoch 밀리초)  
  "message": "로그 본문 내용",    // 로그 메시지 (1줄 문자열)  
  "ingestionTime": 1716728460000  // CloudWatch가 수집한 시간  
}
```

필드명	타입	설명
timestamp	number	로그가 생성된 시각 (밀리초 단위)
message	string	로그 본문 1줄 텍스트 (사용자 print(), 시스템 로그 등)
ingestionTime	number	로그가 CloudWatch로 전송된 시간

## 실습2) CloudWatch 조회 용 Lambda 생성

- 람다 : sgu-계정-cloudwatch
- 런타임 : python 3.13
- 제한시간 : 10초 (이유는 알 것이라 생각함)
- DynamoDB 저장
  - user\_id= cloudwatch
  - timestamp=현재시간
  - product=cloudwatch message



## 실습2) Lambda코드

```
import boto3
import json
from datetime import datetime

def lambda_handler(event, context):
    logs = boto3.client('logs')
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('sgu-202500-user-likes-time')

    log_group = '/aws/lambda/sgu-202500-lambda-s3'

    try:
        # 1. 최근 로그 스트림 가져오기
        streams = logs.describe_log_streams(
            logGroupName=log_group,
            orderBy='LastEventTime',
            descending=True,
            limit=1
        )['logStreams']

        if not streams:
            return { 'statusCode': 404, 'body': 'No log streams found' }

        stream_name = streams[0]['logStreamName']

        # 2. 로그 이벤트 가져오기
        events = logs.get_log_events(
            logGroupName=log_group,
            logStreamName=stream_name,
            limit=20,
            startFromHead=True
        )['events']
```

```
# 3. 로그 이벤트를 하나씩 DynamoDB에 저장
for e in events:
    # CloudWatch timestamp → ISO 형식 변환
    event_time = datetime.fromtimestamp(e['timestamp'] / 1000).isoformat()
    message = e['message'].strip()

    item = {
        'user_id': 'cloudwatch',
        'timestamp': event_time,    # CloudWatch 로그 시각으로 저장
        'product': message
    }

    table.put_item(Item=item)
    print(f" Inserted: {item}")

return {
    'statusCode': 200,
    'body': json.dumps('CloudWatch 로그 → DynamoDB 저장 완료', ensure_ascii=False)
}

except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({'error': str(e)}, ensure_ascii=False)
    }
```