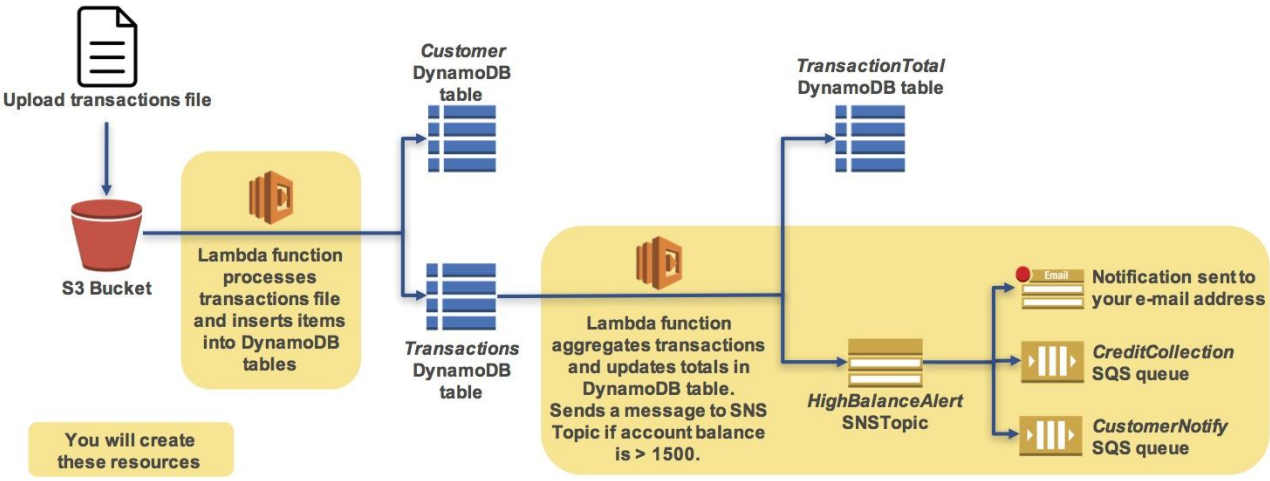
AWS 관리형 서비스로 서버 없는 아키텍처 구현

본 실습에서는 AWS 관리형 서비스를 사용하여 서버 없는 아키텍처를 구현합니다.

시스템이 트랜잭션 파일을 수신한 뒤 자동으로 그 콘텐츠를 데이터베이스에 로드하고 알림을 보냅니다. 이러한 작업이 Amazon EC2 서버를 전혀 사용하지 않고 이루어집니다.

시나리오

다음의 다이어그램이 실습 시나리오를 보여줍니다.



시나리오의 워크플로는 다음과 같습니다.

- *트랜잭션 파일*을 Amazon S3 버킷으로 **업로드**합니다.
- **AWS Lambda 함수를 트리거**합니다. AWS Lambda 함수는 파일을 읽고 기록을 두 개의 **Amazon DynamoDB 테이블**에 삽입합니다.
- 계정 잔액이 1,500 USD 이상이면 고객 총계를 계산하고 **Amazon Simple Notification Service(SNS) 주제로 메시지를 전송**하는 **또 다른 AWS Lambda 함수를 트리거**합니다.
- 그런 다음 Amazon SNS가 **이메일 알림을 전송**하고 **Amazon Simple Queue Service(SQS) 대기열에 메시지를 저장**하여 고객과 채권 추심 부서에 알립니다.

목표

본 실습을 완료하면 다음과 같은 작업을 수행할 수 있습니다.

- AWS 관리형 서비스를 사용하여 서버 없는 아키텍처 구현
- Amazon S3 및 Amazon DynamoDB로부터 AWS Lambda 함수 트리거

소요 시간

본 실습에는 약 **45분**이 소요됩니다.

AWS Management Console 액세스

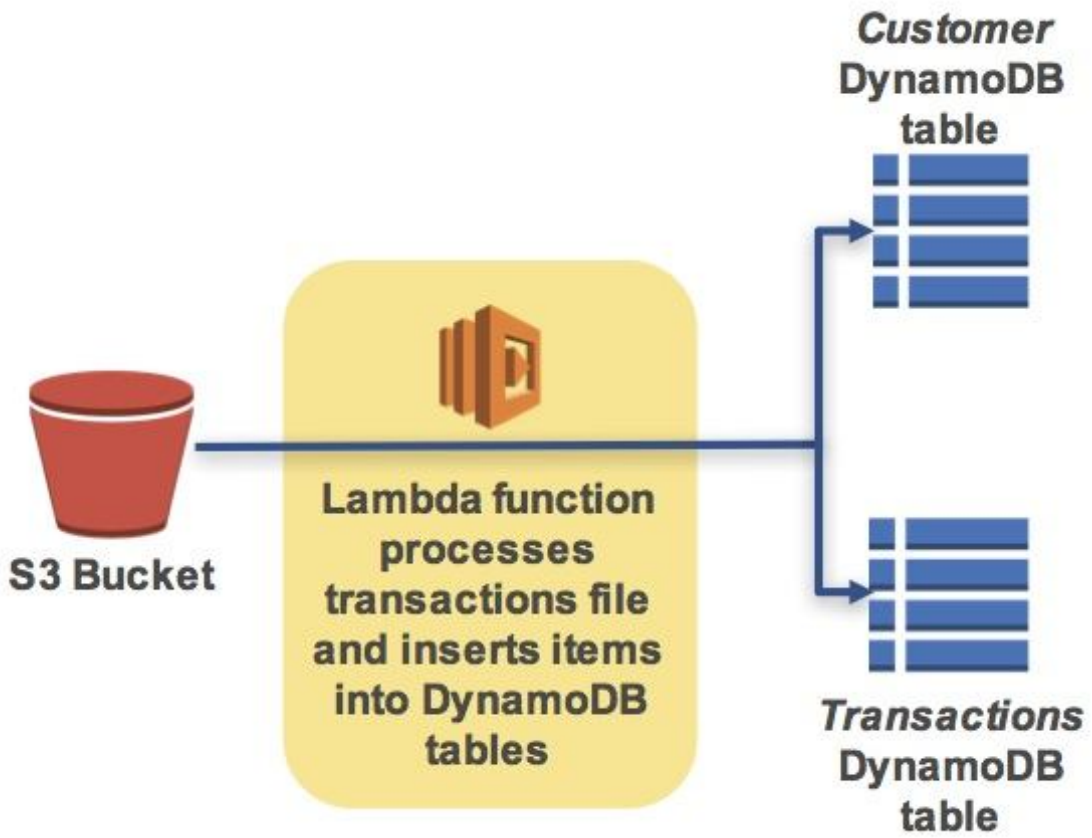
1. [1]실습 제목 오른쪽에서 **Start Lab**을 클릭하여 Qwiklabs를 시작합니다.

1. [2]Qwiklabs 페이지의 **Connect** 탭에서 **Password**를 클립보드로 복사한 후 **Open Console**을 클릭합니다.

1. [3]다음 단계에 따라 AWS Management Console에 로그인합니다.

- **User Name**에 'awsstudent'를 입력합니다.
- **Password**에 클립보드에서 복사한 암호를 붙여넣습니다.
- **Sign In**을 클릭합니다.

작업 1: Lambda 함수를 생성하여 트랜잭션 파일 처리



이번 작업에서는 *AWS Lambda 함수**를 생성하여 트랜잭션 파일을 처리합니다. Lambda 함수는 파일을 읽고 *Customer*와 *Transactions* DynamoDB 테이블에 정보를 삽입합니다.

1. [4] **AWS Management Console**의 **Services** 메뉴에서 **Lambda**를 클릭합니다.
2. [5] **Get Started Now**를 클릭합니다.

Blueprints는 Lambda 함수 기록용 코드 템플릿입니다. 청사진은 Alexa skills 생성이나 Amazon Kinesis Firehose 스트림 처리와 같은 표준 Lambda 트리거용으로 제공됩니다. 이 실습에서는 사전에 작성된 Lambda 함수를 제공하므로 청사진을 사용하지 않습니다.

1. [6] **Select blueprint** 페이지에서 **Blank Function**을 클릭합니다.
2. [7] **Configure triggers** 페이지에서 Lambda 아이콘 왼쪽의 빈칸을 클릭합니다.
3. [8] 드롭다운 목록이 나타나면 **S3**(아래로 스크롤해야 보일 수 있음)를 클릭합니다.

Amazon S3 버킷에 파일이 생성될 때마다 Lambda 함수가 실행됩니다.

1. [9] **Configure triggers** 페이지에서 다음과 같이 설정합니다(목록에 없는 설정은 모두 무시).

Bucket	**inputs3** 단어가 포함된 버킷을 선택합니다. *qls-xxxx-xxxx-inputs3...* 형식으로 되어 있습니다. (버킷 위로 마우스를 가져가면 전체 이름을 확인할 수 있습니다.)
Event type	**Object Created (All)** *Object Removed*를 선택하지 마십시오.
Enable trigger	상자 선택(체크)

1. [10] **Next**를 클릭합니다.
2. [11] **Configure function** 페이지에서 다음과 같이 설정합니다(목록에 없는 설정은 모두 무시).

Name	'TransactionProcessor'
Description	'Process data and send to DynamoDB tables'
Runtime	**Python 2.7** 선택

1. [12] **Code entry type** 섹션 아래에 나타나는 **모든 코드를 삭제**하여 필드를 비웁니다.
2. [13] 아래의 **Copy Code Block**을 클릭하고 코드 필드에 붙여넣습니다.

```
#-*- coding: utf-8 -*-
# TransactionProcessor Lambda 함수
#
# 이 함수는 Amazon S3 버킷에서 생성되는 객체가 트리거합니다.
# 파일이 다운로드되고 각 줄이 DynamoDB 테이블에 삽입됩니다.

from __future__ import print_function
import json, urllib, boto3, csv

# S3와 DynamoDB에 연결
s3 = boto3.resource('s3')
dynamodb = boto3.resource('dynamodb')

# DynamoDB 테이블에 연결
customerTable     = dynamodb.Table('Customer');
transactionsTable = dynamodb.Table('Transactions');

# Lambda 함수가 트리거될 때마다 실행되는 핸들러
def lambda_handler(event, context):

    # 디버깅 로그에 수신 이벤트 표시
    print("Event received by Lambda function: " + json.dumps(event, indent=2))

    # 이벤트에서 버킷과 객체 키 획득
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf8')
    localFilename = '/tmp/transactions.txt'

    # S3에서 로컬 파일 시스템으로 파일 다운로드
    try:
        s3.meta.client.download_file(bucket, key, localFilename)
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. 버킷이 존재하고이 버킷이 이 함수와 같은 리
        raise e

    # 트랜잭션 CSV 파일을 읽음 Delimiter is the '|' character
    with open(localFilename) as csvfile:
        reader = csv.DictReader(csvfile, delimiter='|')

        # 파일의 각 행을 읽음
        rowCount = 0
        for row in reader:
            rowCount += 1

            # 디버깅 로그에 행 표시
            print(row['customer_id'], row['customer_address'], row['trn_id'], row['trn_date'], ro

            try:
                # 고객 ID와 주소를 고객 DynamoDB 테이블에 삽입
                customerTable.put_item(
                    Item={
                        'CustomerId': row['customer_id'],
                        'Address':    row['customer_address']})
```

```
            # 트랜잭션 세부 사항을 트랜잭션 DynamoDB 테이블에 삽입
            transactionsTable.put_item(
                Item={
                    'CustomerId':        row['customer_id'],
                    'TransactionId':     row['trn_id'],
                    'TransactionDate':   row['trn_date'],
                    'TransactionAmount': int(row['trn_amount'])})

        except Exception as e:
             print(e)
             print("Unable to insert data into DynamoDB table".format(e))

    # 완료!
    return "%d transactions inserted" % rowCount
```

코드를 검사합니다. 다음 단계를 수행합니다.

- 이벤트를 트리거한 Amazon S3에서 파일 다운로드
- 파일 내 각 줄 반복
- 데이터를 *고객* 및 *트랜잭션* DynamoDB 테이블에 삽입

1. [14] **Existing Role**을 아래로 스크롤하고 ***S3LambdaDynamoDBRole*** 텍스트가 포함된 역할을 선택합니다. (버킷 위로 마우스를 가져가면 전체 이름을 확인할 수 있습니다.)

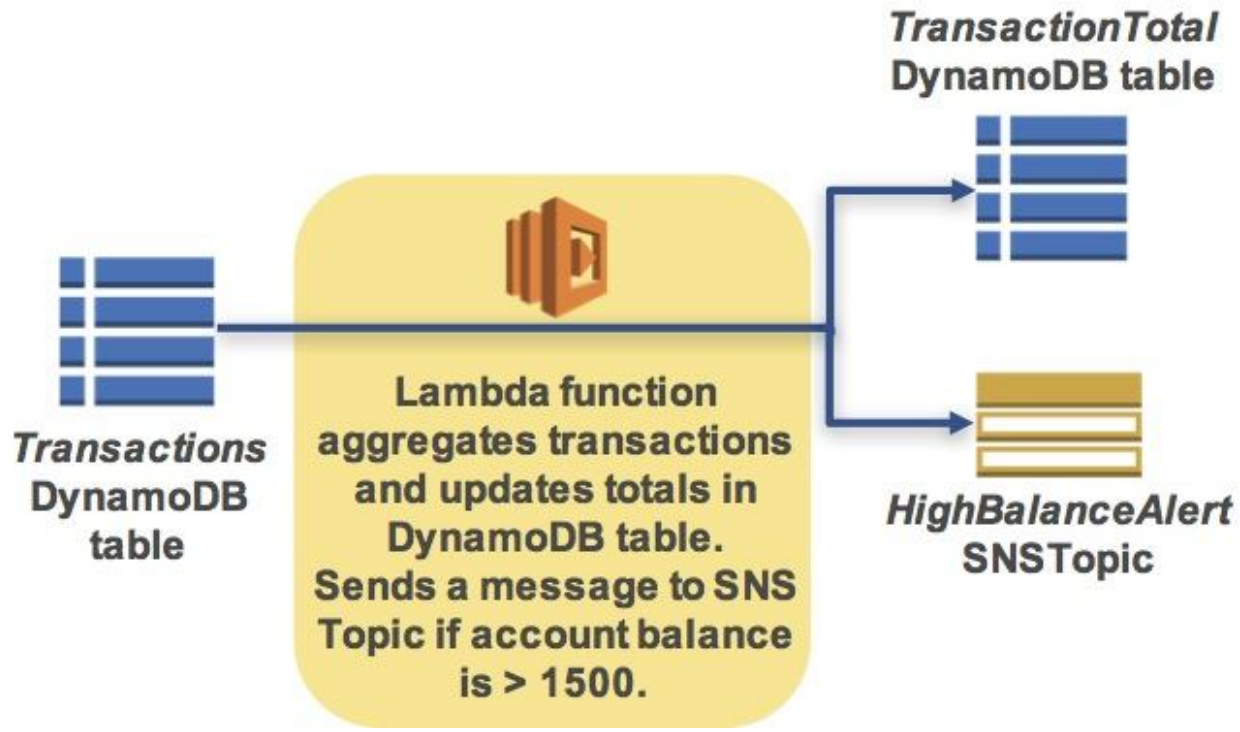
"SNS"가 포함된 역할은 선택하지 *마십시오*.

이 역할은 Lambda 함수에 실행 권한을 부여하므로 Amazon S3 및 Amazon DynamoDB에 액세스할 수 있습니다.

1. [15] **Advanced settings**를 클릭합니다.
2. [16] **Timeout** 값을 **0 min 20 sec**로 변경합니다.
3. [17] 다른 값은 기본 설정 그대로 두고 **Next**를 클릭합니다.
4. [18] 함수 설정을 검토하고 **Create function**을 클릭합니다.

이제 선택한 Amazon S3 버킷에 파일이 업로드될 때마다 이 Lambda 함수가 실행됩니다. 함수는 업로드된 파일에서 데이터를 읽고 찾아낸 데이터를 **DynamoDB**의 *Customer* 및 *Transactions* 테이블에 저장합니다.

작업 2: Lambda 함수를 생성하여 트랜잭션 총계를 계산하고 계정 잔액이 높으면 알림



이번 작업에서는 **AWS Lambda 함수**를 생성하여 트랜잭션 총계를 계산하고 계정 잔액이 1,500 USD를 초과하면 Simple Notification Service 알림을 전송합니다.

1. [19] 탐색 창에서 **Functions**를 클릭합니다.
2. [20] **Create a Lambda function**을 클릭합니다.
3. [21] **Select blueprint** 페이지에서 **Blank Function**을 클릭합니다.
4. [22] **Configure triggers** 페이지에서 Lambda 아이콘 왼쪽의 빈칸을 클릭합니다.
5. [23] 드롭다운 목록이 나타나면 **DynamoDB**를 클릭합니다.

지정된 DynamoDB 테이블에 데이터가 추가될 때마다 Lambda 함수가 실행됩니다.

1. [24] **Configure triggers** 페이지에서 다음과 같이 설정합니다(목록에 없는 설정은 모두 무시).

DynamoDB table	**Transactions**
Starting position	**Latest**
Enable trigger	상자 선택(체크)

*Starting Position*은 DynamoDB에 추가된 최신 정보를 처리할지 아니면, 사용 가능한 가장 빠른 데이터(*Trim Horizon*이라고 함)로 되돌릴지를 Lambda에 알려줍니다. 이 함수는 *최신* 정보를 사용합니다.

1. [25]**Next**를 클릭합니다.
2. [26]**Configure function** 페이지에서 다음과 같이 설정합니다(목록에 없는 설정은 모두 무시).

Name	'TotalNotifier'
Description	'Update total, send notification for balance exceeding $1500'
Runtime	**Python 2.7**

1. [27]**Code entry type** 섹션 아래에 나타나는 **모든 코드를 삭제**하여 필드를 비웁니다.
2. [28]아래의 **Copy Code Block**을 클릭하고 코드 필드에 붙여넣습니다.

```
#-*- coding: utf-8 -*-
# TotalNotifier Lambda 함수
#
# 트랜잭션 DynamoDB 테이블에 값이 삽입되면 이 함수가 트리거됩니다.
# 트랜잭션 총계를 계산하고 한도를 초과하면 SNS로 알림을 보냅니다.

from __future__ import print_function
import json, boto3

# SNS에 연결
sns = boto3.client('sns')
alertTopic = 'HighBalanceAlert'
snsTopicArn = [t['TopicArn'] for t in sns.list_topics()['Topics'] if t['TopicArn'].endswith(':' +

# DynamoDB에 연결
dynamodb = boto3.resource('dynamodb')
transactionTotalTableName = 'TransactionTotal'
transactionsTotalTable = dynamodb.Table(transactionTotalTableName);

# Lambda 함수가 트리거될 때마다 실행되는 핸들러
def lambda_handler(event, context):

    # 디버깅 로그에 수신 이벤트 표시
    print("Event received by Lambda function: " + json.dumps(event, indent=2))

    # 트랜잭션이 추가될 때마다 새 트랜잭션 총계 계산
    for record in event['Records']:
        customerId = record['dynamodb']['NewImage']['CustomerId']['S']
        transactionAmount = int(record['dynamodb']['NewImage']['TransactionAmount']['N'])

        # TransactionTotal DynamoDB 테이블에 고객 총계 업데이트
        response = transactionsTotalTable.update_item(
            Key={
                'CustomerId': customerId
            },
            UpdateExpression="add accountBalance :val",
            ExpressionAttributeValues={
                ':val': transactionAmount
            },
            ReturnValues="UPDATED_NEW"
        )

        # 최근 계정 잔액 검색
        latestAccountBalance = response['Attributes']['accountBalance']
        print("Latest account balance: " + format(latestAccountBalance))

        # 잔액이 1,500 USD를 초과하면 SNS로 메시지 전송
        if latestAccountBalance  >= 1500:

            # 전송할 메시지 구성
            message = '{"customerID": "' + customerId + '", ' + '"accountBalance": "' + str(lates
            print(message)

            # SNS로 메시지 전송
```

```
            sns.publish(
                TopicArn=snsTopicArn,
                Message=message,
                Subject='Warning! Account balance is very high',
                MessageStructure='raw'
            )

    # 완료!
    return 'Successfully processed {} records.'.format(len(event['Records']))
```

코드를 검사합니다. 다음 단계를 수행합니다.

- Amazon SNS와 Amazon DynamoDB에 연결
- 트랜잭션 총계를 계산하고 *TransactionTotal* DynamoDB 테이블에 저장
- 트랜잭션 총계가 1,500 USD 이상이면 Amazon SNS로 알림 전달

1. [29] **Existing Role**을 아래로 스크롤하고 ***SNSLambdaDynamoDBRole*** 텍스트가 포함된 역할을 클릭합니다.

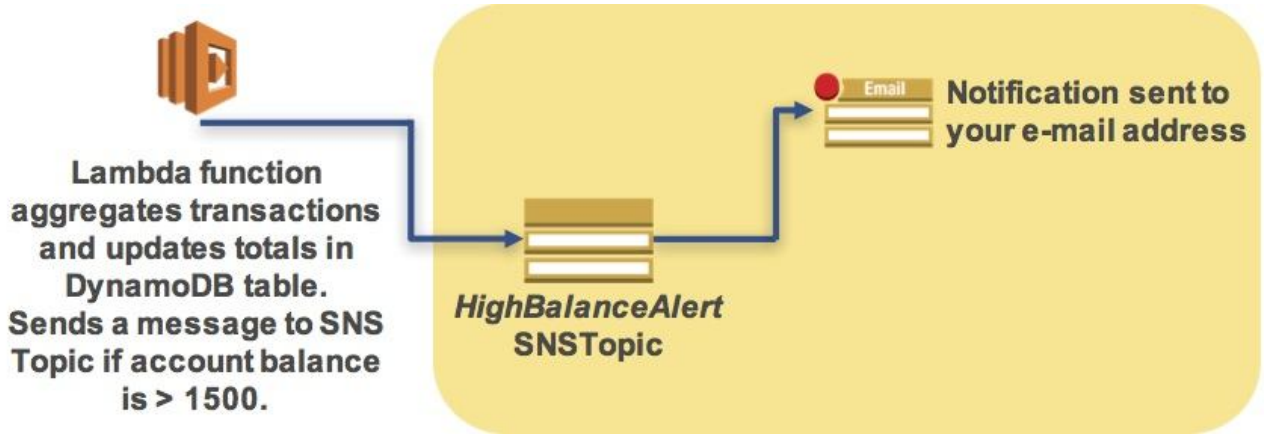
"S3"가 포함된 역할은 선택하지 *마십시오*.

이 역할은 Amazon DynamoDB 및 Amazon Simple Notification Service(SNS)에 액세스하는 데 필요한 실행 권한을 AWS Lambda에 부여합니다.

1. [30] **Advanced settings**를 클릭합니다.
2. [31] **Timeout** 값을 **0 min 20 sec**로 변경합니다.
3. [32] 다른 값은 기본 설정 그대로 두고 **Next**를 클릭합니다.
4. [33] 함수 설정을 검토하고 **Create function**을 클릭합니다.

이제 *Transactions* DynamoDB 테이블이 업데이트될 때마다 이 함수가 각 고객의 트랜잭션 총계를 계산하고 *TransactionTotal* DynamoDB 테이블에 저장합니다. 총계가 1,500 USD를 초과하면 Simple Notification Service 주제에 메시지를 전송하여 고객과 채권 추심 부서에 알립니다.

작업 3: Simple Notification Service(SNS) 주제 생성



이번 작업에서는 계정 잔액이 1,500 USD를 초과하면 Lambda 함수로부터 알림을 받을 **Simple Notification Service(SNS) topic**을 생성합니다. 또한, 이메일 주소나 SMS(선택 사항)로 주제를 구독합니다.

1. [34] **Services** 메뉴에서 **Simple Notification Service**를 클릭합니다.
2. [35] **Get started**를 클릭합니다.
3. [36] **Create topic**을 클릭합니다.
4. [37] **Create new topic 대화 상자에서 다음과 같이 설정합니다.

Topic name	'HighBalanceAlert' 이 이름을 정확하게 입력해야 Lambda 함수가 알림을 트리거할 수 있습니다.
Display name	'HighAlert'

1. [38] **Create topic**을 클릭합니다.
2. [39] **Create subscription**을 클릭합니다.
3. [40] **Create subscription** 대화 상자에서 다음과 같이 설정합니다(목록에 없는 설정은 모두 무시).

Protocol	**Email**
Endpoint	자주 사용하는 이메일 주소를 입력합니다. 회사 또는 개인 이메일 주소를 사용하면 됩니다. 실습에서 생성한 Simple Notification Service 주제가 입력한 이메일로 알림을 전송합니다.

1. [41] **Create subscription**을 클릭합니다.

확인 요청이 입력한 이메일 주소로 발송됩니다. 알림을 받으려면 구독 확인을 해야 합니다.

1. [42] 방금 제공한 이메일 계정에서 **HighAlert**으로부터 받은 새 이메일을 확인합니다. 전송에 일 분 정도 걸릴 수 있습니다.
2. [43] 이메일이 도착하면, 본문에 있는 **Confirm subscription** 링크를 클릭합니다.

이제 Simple Notification Service 주제가 메시지를 받을 때마다 이메일로 알림을 전송합니다.

선택 사항: SMS를 통해 휴대폰으로 메시지를 수신할 수도 있습니다. SMS로 구독하려면 다음 단계를 진행하십시오.

- **Create subscription** 클릭
- **Protocol**에서 **SMS** 선택
- **Endpoint**에 국제 번호 형식(예: +14155557000, +917513200000 등)으로 휴대폰 번호 입력
- **Create subscription** 클릭

작업 4: 두 개의 Simple Queue Service 대기열 생성

이번 작업에서는 두 개의 **Simple Queue Service(SQS) 대기열**을 생성합니다. 이 대기열들은 방금 생성한 Simple Notification Service(SNS) 주제를 구독합니다. 이러한 설정을 *팬아웃 시나리오*라고 합니다. 여러 명의 구독자에게 각 SNS 알림이 전송되고 해당 구독자들이 각자의 대기열에서 독립적으로 메시지를 소비할 수 있기 때문입니다.

먼저, 고객 알림용 대기열을 생성합니다.

1. [44] **Services** 메뉴에서 **Simple Queue Service**를 클릭합니다.
2. [45] **Get Started Now**를 클릭합니다.
3. [46] **Create New Queue** 대화 상자의 **Queue Name**에 'CustomerNotify'를 입력합니다.

리전이 선입선출(FIFO) 대기열을 지원하는지 아닌지에 따라 인터페이스가 달라집니다.

1. [47] "어떤 유형의 대기열이 필요하십니까?"라는 메시지가 표시되면 **Standard Queue**를 기본 선택으로 두고 맨 아래로 스크롤을 내립니다.
2. [48] 나머지 설정을 기본값으로 두고 **Create Queue**나 **Quick-Create Queue**를 클릭합니다.

완전한 애플리케이션 환경에서는 Lambda 함수 또는, 다른 애플리케이션을 사용하여 이 대기열의 메시지를 읽고, 잔액이 높은 사용자를 알려줄 수 있습니다.

다음으로 채권 추심 부서 알림용 대기열을 생성합니다.

1. [49] **Create New Queue**를 클릭합니다.
2. [50] **Create New Queue** 대화 상자의 **Queue Name**에 'CreditCollection'를 입력합니다.
3. [51] 나머지 설정을 기본값으로 두고 **Create Queue**나 **Quick-Create Queue**를 클릭합니다.

완전한 애플리케이션 환경에서는 Lambda 함수 또는 다른 애플리케이션을 사용하여 이 대기열의 메시지를 읽고, 해당 계정을 모니터링하도록 채권 추심 부서에 알려줄 수 있습니다.

1. [52] 두 대기열의 확인란을 *모두* 선택합니다.
2. [53] **Queue Actions**를 클릭한 다음, **Subscribe Queues to SNS Topic**을 클릭합니다.
3. [54] **Subscribe to a Topic** 대화 상자의 **Choose a Topic**에서 **HighBalanceAlert**을 클릭한 다음, **Subscribe**를 클릭합니다.
4. [55] **Topic Subscription Result** 대화 상자에서 **OK**를 클릭합니다.

이제 두 개의 대기열이 Simple Notification Service 주제를 구독합니다. 해당 주제로 전송되는 메시지를 모두 자동으로 수신합니다.

작업 5: 트랜잭션 파일을 업로드하여 서버 없는 아키텍처 테스트

이번 작업에서는 트랜잭션 파일을 검색하여 S3 버킷으로 업로드합니다. 그런 다음 서버 없는 아키텍처를 테스트합니다.

작업 5.1: 트랜잭션 파일 업로드

1. [56] 다음 링크를 클릭하여 transactions.txt 파일을 로컬 컴퓨터로 다운로드합니다.

https://us-west-2-aws-staging.s3.amazonaws.com/awsu-ilt/AWS-100-ARC/v5.2/lab-4-serverless/scripts/transactions.txt

1. [57] 파일이 다운로드되지 않고, 웹 브라우저에 파일이 *표시*되는 경우, 웹 페이지를 **우클릭**한 다음, 페이지를 텍스트 파일로 **저장**합니다.
2. [58] **Services** 메뉴에서 **S3**를 클릭합니다.
3. [59] *qls-xxxx-xxxx-inputs3bucketfortransact-xxx* 형식의 이름으로 된 버킷을.클릭합니다.
4. [60] **Upload**를 클릭합니다.
5. [61] **Add files**를 클릭한 다음 다운로드 받은 *transactions.txt* 파일을 선택합니다.
6. [62] **Upload**를 클릭합니다.

Amazon S3로 이 파일을 업로드하면 실습에서 생성한 첫 번째 Lambda 함수가 즉시 트리거되어 텍스트 파일을 읽고 데이터를 **Customer** 및 **Transactions** DynamoDB 테이블에 저장합니다.

작업 5.2: DynamoDB 테이블 확인

이제 데이터가 DynamoDB 테이블에 로드되었는지 확인하여 트랜잭션 파일이 올바르게 처리되었는지 확인할 수 있습니다.

1. [63] **Services** 메뉴에서 **DynamoDB**를 클릭합니다.
2. [64] 탐색 창에서 **Tables**를 클릭합니다.
3. [65] **Customer**를 클릭합니다.
4. [66] **Items** 탭에서 2명의 고객에 대한 고객 ID 및 주소 항목이 있는지 확인합니다.

데이터가 보이지 않는다면, Lambda 함수에 오류가 발생했거나 실행되지 않은 것입니다. 강사의 지원을 받아 구성을 확인하십시오.

1. [67] **Transactions**를 클릭합니다.
2. [68] **Items** 탭에서 일부 트랜잭션이 존재하는지 확인합니다. 목록에 총 24개의 항목이 있을 것입니다.

Transactions 테이블에 정보가 추가되면 두 번째 DynamoDB 함수가 자동으로 트리거됩니다. 이 함수는 각 계정의 트랜잭션 총계를 계산하고 *TransactionTotal* 테이블에 총계를 저장합니다. 이제 프로세스가 제대로 작동했는지 확인할 수 있습니다.

1. [69] **TransactionTotal**을 클릭합니다.
2. [70] **Items** 탭에서 2명의 고객에 대한 고객 ID 및 계정 잔액이 있는지 확인합니다. *C2* 고객 계정 잔액이 1,500 USD를 초과한다는 점을 기억하십시오.

데이터가 보이지 않는다면 두 번째, Lambda 함수에 오류가 발생했거나 실행되지 않은 것입니다. 강사의 지원을 받아 구성을 확인하십시오.

작업 5.3: SQS 대기열 확인

지금쯤 *HighAlert*에서 C2 고객의 계정 잔액이 높다는 내용의 알림 이메일을 새로 받았을 것입니다. 또한, 같은 메시지가 2개의 Simple Queue Service 대기열에 전송되었으며, 다른 프로세스에 의한 처리를 준비합니다.

휴대폰 번호를 제공했다면, SMS 알림도 받았을 것입니다. 지금까지 Amazon Simple Notification Service가 사람이나 시스템에 알림을 보내는 수많은 방법 중 몇 가지 예를 알아봤습니다.

1. [71] **Services** 메뉴에서 **Simple Queue Service**를 클릭합니다.

Lambda 함수가 제대로 작동했다면 양쪽 대기열 모두 **Messages Available** 열에 하나의 메시지를 표시해야 합니다. 메시지가 보이지 않는다면 강사의 지원을 받아 구성을 확인하십시오.

1. [72] **CreditCollection**을 선택합니다.
2. [73] **Queue Actions**를 클릭한 다음 **View/Delete Messages**를 클릭합니다.
3. [74] **Start Polling for Messages**를 클릭합니다.
4. [75] 표시된 메시지에서 **More Details**를 클릭합니다.
5. [76] 메시지 본문에 * C2 * 고객에 대한 경고가 표시되는지 확인합니다. 다음과 같은 메시지가 표시되어야 합니다.

```
{
"Type" : "Notification",
"MessageId" : "eb0d030d-5f2d-5695-8f22-4c68d0335c0b",
"TopicArn" : "arn:aws:sns:us-east-1:123456789:HighAccountBalanceAlertSNSTopic",
"Subject" : "Warning! Account balance is very high",
"Message" : "{₩₩"customerID₩₩": ₩₩"C2₩₩", ₩₩"accountBalance₩₩":₩₩"1750₩₩"}",
...
}
```

1. [77] **Close**를 클릭합니다.

2. [78] **Close**를 다시 클릭하여 대기열 목록으로 돌아갑니다.

3. [79] **선택 사항: CustomerNotify** 대기열에서도 메시지를 확인합니다. **CreditCollection** 대기열에 있는 메시지와 똑같은 메시지를 포함해야 합니다.

작업 5.4: Lambda 함수 확인

Lambda 함수는 자동으로 로그와 지표를 기록합니다. 이 정보를 확인하여 함수가 제대로 실행되었는지 확인할 수 있습니다.

1. [80] **Services** 메뉴에서 **Lambda**를 클릭합니다.
2. [81] **TransactionProcessor**를 클릭(이름을 직접 클릭)한 다음, **Monitoring** 탭을 클릭하여 Lambda 함수에 대한 CloudWatch 지표를 확인합니다.

지표에서 Lambda 함수가 호출되었고 오류가 발생하지 않았다는 사실을 확인할 수 있어야 합니다.

1. [82] 탐색 창에서 **Functions**를 클릭합니다.
2. [83] **TotalNotifier**를 클릭(이름을 직접 클릭)한 다음, **Monitoring** 탭을 클릭하여 Lambda 함수에 대한 CloudWatch 지표를 확인합니다.

로그 파일은 Lambda 함수 디버깅에 유용합니다. Amazon CloudWatch 로그에 저장됩니다.

1. [84] **View logs in CloudWatch**를 클릭하여 함수 로그를 확인합니다. 표시된 링크를 클릭하고 로그를 검토합니다.

실습 완료

축하합니다! 실습을 완료했습니다.

본 실습용 qwikLABS 페이지로 돌아가 **End**를 클릭하여 실습 환경을 종료해 주십시오.