Architecting on AWS - 실습 6 AWS 관리형 서비스로 서버리스 아키텍처 구현

실습 개요

전통적으로 애플리케이션은 서버에서 실행됩니다. 이는 물리적 서버 또는 물리적 서버 외에 실행 중인 가상 환경일 수 있지만 서버 구매와 서버 프로비저닝 및 용량 관리가 필요합니다. 대신 AWS Lambda 는 서버를 미리 할당할 필요 없이 서버리스 코드를 실행할 수 있습니다. 코드를 제공하고 트리거를 정의하기만 하면 됩니다. 함수는 필요할 때마다(주당 1 회 또는 초당 수백 번) 실행할 수 있으며 사용한 만큼만 비용을 지불합니다.

이 실습에서는 파일을 Amazon Simple Storage Service(Amazon S3)에 업로드할 때 Lambda 함수를 트리거하는 방법을 설명합니다. 파일은 Amazon DynamoDB 테이블에 로드되며, DynamoDB 에서 데이터를 직접 가져오는 대시보드 페이지에서 데이터를 볼 수 있습니다. 이 솔루션은 완전한 서버리스이고 자동으로 확장 가능하며 비용이 매우 저렴합니다.

시스템에서 Amazon Elastic Compute Cloud(Amazon EC2)를 사용하지 *않습니다*. 시스템은 사용할 때는 자동으로 확장되고 사용하지 않을 때는 거의 비용이 발생하지 *않습니다*(데이터 스토리지에 불과 몇 센트).

목표

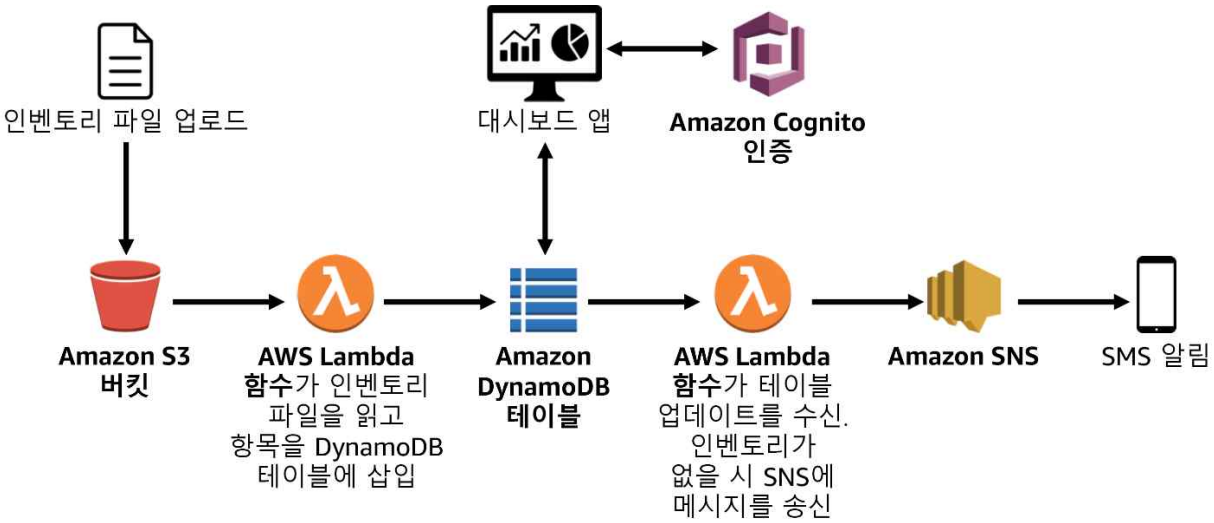
이 실습을 완료하면 다음을 할 수 있게 됩니다.

- Lambda 함수 생성
- Amazon S3 및 DynamoDB 에서 Lambda 함수 트리거
- 알림을 전송하도록 Amazon SNS 를 구성

시나리오

현재 재고 추적 시스템을 만드는 중입니다. 전 세계 스토어에서 재고 파일을 Amazon S3 에 업로드합니다. 팀에서는 재고 수준을 보고 재고 수준이 낮을 때 알림을 보낼 수 있기를 바랍니다.

다음 다이어그램은 이 워크플로를 보여줍니다.



시나리오 워크플로는 다음과 같습니다.

- 재고 파일을 Amazon S3 버킷에 업로드합니다.
- 그러면 Lambda 함수가 트리거되어 파일을 읽고 항목을 DynamoDB 테이블에 삽입합니다.
- 서버리스 웹 기반 대시보드 애플리케이션이 Amazon Cognito 를 사용하여 AWS 에 인증한 다음 DynamoDB 테이블에 대한 액세스 권한을 얻어 재고 수준을 표시합니다.
- 다른 Lambda 함수가 DynamoDB 테이블에서 업데이트를 수신하고 재고품이 떨어지면 Amazon Simple Notification Service(Amazon SNS) 주제에 메시지를 보냅니다,
- 그런 다음 Amazon SNS 가 사용자에게 SMS 또는 이메일 알림을 보내 추가 재고를 요청합니다.

소요 시간

이 실습을 완료하는 데는 약 **40 분**이 소요됩니다.

실습 시작

1. 이 링크를 마우스 오른쪽 버튼으로 클릭한 다음 자신의 컴퓨터로 arc_lab6_template.json 을 다운로드합니다.(실습 배포 파일사용)
2. AWS Management Console 의 서비스 메뉴에서 **Management & Governance > CloudFormation** 을 클릭합니다.

3. **Create stack** 을 클릭하고 아래 단계에 따라 스택을 생성합니다.

   1 단계: 템플릿 지정

- **Template source:** Upload a template file 을 선택합니다.
- **Upload a template file:** Choose file 을 클릭하고 다운로드한 **arc_lab6_template.json** 파일을 선택합니다.
- Next 를 클릭합니다.

2 단계: 스택 세부 정보 지정

- **Stack name:** arclab6
- Next 를 클릭합니다.

3 단계: 스택 옵션 구성

- Next 를 클릭합니다.

4 단계: 검토

- **I acknowledge that...** 의 체크박스에 체크합니다.
- Create stack 을 클릭합니다.

AWS CloudFormation 에서는 이제 템플릿을 사용하여 리소스의 *스택* 을 생성합니다.

Stack info 탭을 클릭합니다.

- **Status** 가 CREATE_COMPLETE 로 변경될 때까지(약 1 분) 대기합니다.

  참고 필요한 경우 새로 고침 아이콘을 15 초마다 클릭하면 화면이 업데이트됩니다.

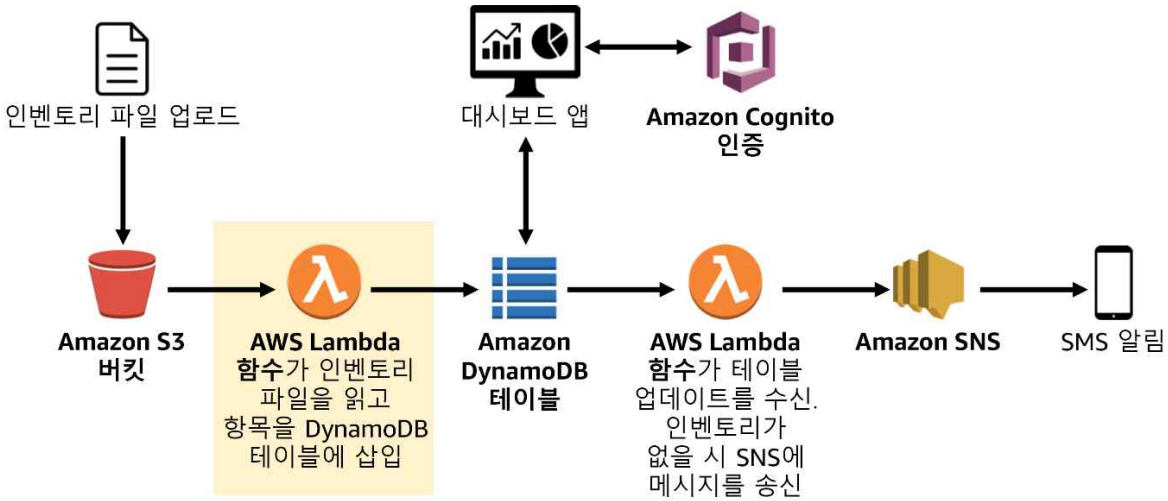
4. **Outputs** 탭을 클릭합니다.

AWS CloudFormation 스택에서 지정된 리소스 ID 및 리소스 링크와 같은 *출력 정보* 를 제공할 수 있습니다.

- **Dashboard:** 매장별 재고 내역을 볼 수 있는 대시보드 주소입니다.
- **IdentityPoolId:** Cognito 서비스에 생성 된 Identity Pool 의 ID 입니다.
- **Region:** 생성된 리소스들의 리전 코드입니다.

작업 1: Lambda 함수를 생성하여 데이터 로드

이 작업에서는 Lambda 함수를 생성하여 재고 파일을 처리합니다. Lambda 함수는 다음 다이어그램에 표시된 것과 같이 파일을 읽고 DynamoDB 테이블에 정보를 삽입합니다.



5. AWS Management Console 의 **Services** 메뉴에서 **Compute > Lambda** 를 클릭합니다.

6. **Create function** 을 클릭합니다.

참고 *블루프린트*는 Lambda 함수 작성에 사용하는 코드 템플릿입니다. 블루프린트는 Alexa 기술 생성 및 Amazon Kinesis Data Firehose 스트림 처리 같은 표준 Lambda 트리거의 경우에 제공됩니다. 본 실습에서는 사전 작성된 Lambda 함수를 제공하므로 처음부터 함수를 작성합니다.

7. 다음을 구성합니다.

- **Create new** 를 선택합니다.
- **Function name:** Load-Inventory
- **Runtime:** *Python 3.9*
- **Change default execution role** 을 확장합니다.
- **Execution role: Use an existing role** 을 선택합니다.
- **Existing role:** *Lambda-Load-Inventory-Role* 으로 입력

이 역할은 Lambda 함수에 실행 권한을 부여하므로 Lambda 함수가 Amazon S3 및 DynamoDB 에 액세스할 수 있습니다.

8. **Create function** 을 클릭합니다.

9. **Code source** 섹션까지 스크롤을 내린 다음, 코드 편집기에서 lambda_function.py 를 더블 클릭해서 열고 표시되는 모든 코드를 삭제합니다.

10. 다음 코드를 복사해 **Code source** 편집기에 붙여 넣습니다.(배포파일 참고)

```
# Load-Inventory Lambda function
#
# This function is triggered by an object being created in an Amazon S3 bucket.
# The file is downloaded and each line is inserted into a DynamoDB table.
```

```
import json, urllib, boto3, csv

# Connect to S3 and DynamoDB
s3 = boto3.resource('s3')
dynamodb = boto3.resource('dynamodb')

# Connect to the DynamoDB tables
inventoryTable = dynamodb.Table('Inventory');

# This handler is executed every time the Lambda function is triggered
def lambda_handler(event, context):

  # Show the incoming event in the debug log
  print("Event received by Lambda function: " + json.dumps(event, indent=2))

  # Get the bucket and object key from the event
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])
  localFilename = '/tmp/inventory.txt'

  # Download the file from S3 to the local filesystem
  try:
    s3.meta.client.download_file(bucket, key, localFilename)
  except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. Make sure they exist and your bucket is
in the same region as this function.'.format(key, bucket))
    raise e

  # Read the Inventory CSV file
  with open(localFilename) as csvfile:
    reader = csv.DictReader(csvfile, delimiter=',')

    # Read each row in the file
    rowCount = 0
    for row in reader:
      rowCount += 1

      # Show the row in the debug log
      print(row['store'], row['item'], row['count'])
```

```
    try:
      # Insert Store, Item, and Count into the Inventory table
      inventoryTable.put_item(
        Item={
          'Store':  row['store'],
          'Item':   row['item'],
          'Count':  int(row['count'])})

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

  # Finished!
  return "%d counts inserted" % rowCount
```

코드에 오타가 있나 잘 확인합니다. 이 코드에서는 다음 단계를 수행합니다.

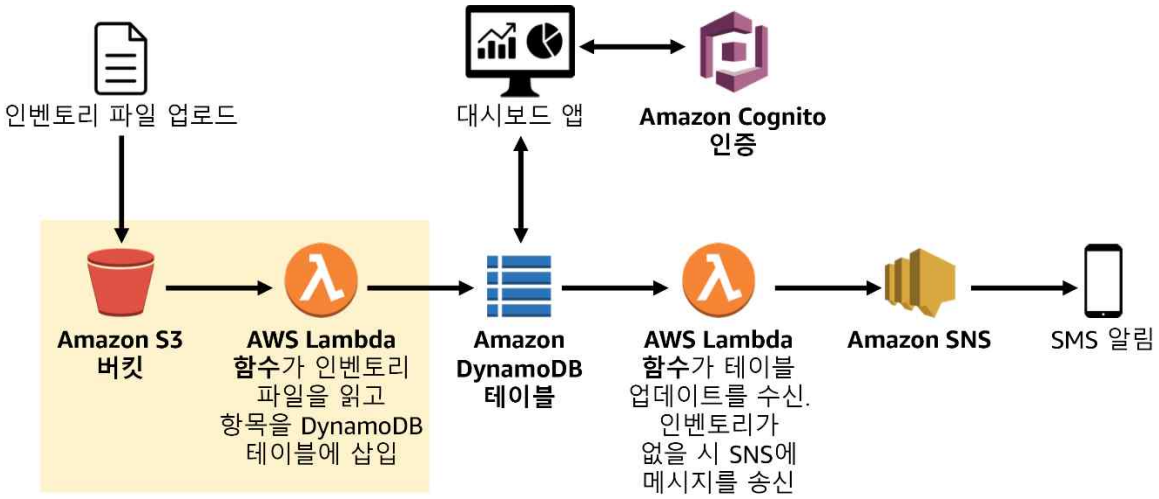
- Amazon S3 에 업로드되어 함수를 트리거한 파일을 다운로드합니다.
- 파일의 각 줄을 루프 처리합니다.
- 데이터를 DynamoDB *Inventory* 테이블에 삽입합니다.

11. 페이지 상단에서 **Deploy** 를 클릭합니다. 람다함수 생성이 완료됩니다

다음으로 파일이 업로드될 때 Lambda 함수를 트리거하도록 Amazon S3 를 구성합니다. AWS 콘솔의 메뉴에서 S3 로 이동합니다

작업 **2: Amazon S3** 이벤트 구성

전 세계 매장이 재고 추적 시스템에 로드 할 재고 파일을 제공합니다. 매장은 FTP 를 통하지 않고 Amazon S3 에 파일을 직접 업로드할 수 있습니다. 이는 웹 페이지나 스크립트 또는 프로그램의 일부를 통해 가능합니다. 다음 다이어그램과 같이 파일이 수신되면 Lambda 함수가 트리거되어 DynamoDB 테이블에 재고 파일을 자동으로 로드합니다.



이 작업에서는 Amazon S3 버킷을 만들고 Lambda 함수를 트리거하도록 구성합니다.

12. **Services** 메뉴에서 **Storage > S3** 를 클릭합니다.

13. **Create bucket** 을 클릭합니다.

각 버킷 이름은 고유해야 하므로 버킷 이름에 난수를 추가합니다. 예: *inventory-123*

14. **Bucket name** 에 `inventory-123` (*123* 을 다른 난수로 교체)을 입력합니다.

15. 맨 아래에서 **Create bucket** 을 클릭 합니다.

참고 동일한 이름의 버킷이 이미 존재한다는 오류가 발생하는 경우 버킷 이름을 변경하고 수락될 때까지 다시 시도하십시오.

이제 파일이 업로드 될 때마다 자동으로 Lambda 함수를 트리거하도록 버킷을 구성합니다.

16. 방금 생성한 이름이 **inventory-xxxx** 인 버킷을 클릭합니다.

17. **Properties** 탭을 클릭합니다.

18. 아래로 스크롤하여 **Event notifications** 세션으로 이동한 다음 **Create event notification** 버튼을 클릭합니다.

19. **General configuration** 항목 아래에 다음과 같이 구성합니다.

    Event name : `Load-Inventory`

20. **Event types** 항목 아래에 다음을 선택합니다.
    All object create events 체크박스에 체크 합니다.

21. 스크롤해서 내려와서 하단의 **Destination** 항목 아래 **Lambda function** 이 선택 되어 있는지 확인 합니다. (만일 선택이 되어 있지 않으면 선택 하십시요.)

Choose from your Lambda functions 가 기본으로 선택 되어 있어야 합니다.

Lambda function 선택 상자 : Load-Inventory 를 선택 합니다.

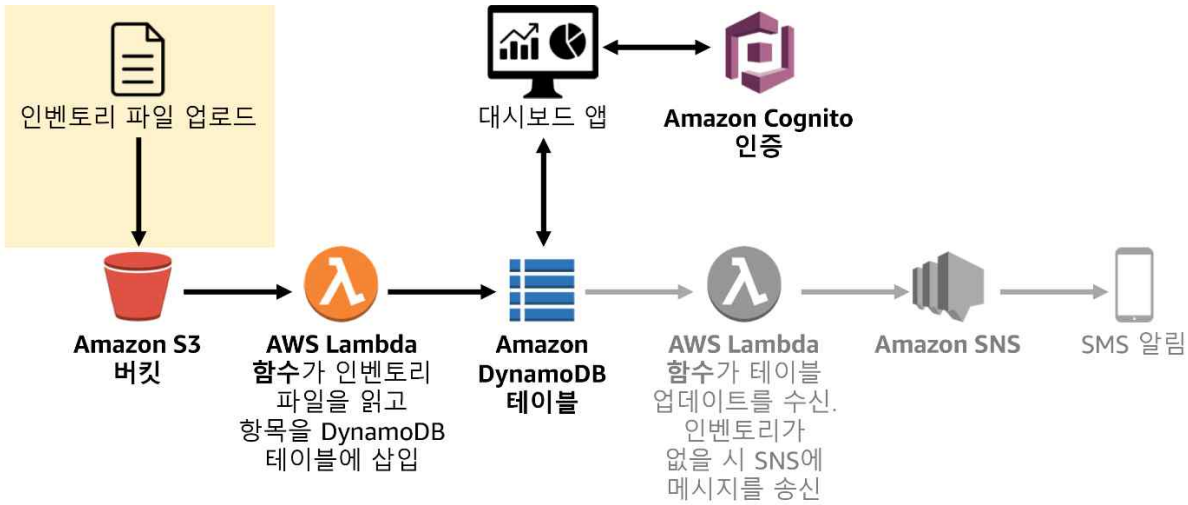
22. 화면 맨 아래 **Save Changes** 버튼을 클릭 합니다.

이렇게 하면 버킷에서 객체가 생성될 때마다 앞서 만든 *Load-Inventory* Lambda 함수를 트리거하라고 Amazon S3 에 알립니다.

이제 버킷이 재고 파일을 수신할 준비가 되었습니다!

작업 3: 로딩 프로세스 테스트

이제 재고 파일 로딩 프로세스를 테스트할 준비가 되었습니다. 재고 파일을 업로드한 다음 성공적으로 로드되었는지 확인합니다. 다음 다이어그램에서는 이러한 단계를 보여줍니다. 이후 작업에서 워크플로의 알림 부분을 테스트합니다.



23. 다음 링크를 마우스 오른쪽 버튼으로 클릭하고 inventory-files.zip 파일을 다운로드합니다.(실습 배포파일 사용)

24. 파일 압축을 풉니다.

이 .zip 파일에는 시스템 테스트에 사용할 수 있는 여러 재고 .csv 파일이 포함되어 있습니다. Berlin 파일에는 다음 데이터가 포함됩니다.

```
store,item,count
Berlin,Echo Dot,12
Berlin,Echo (2nd Gen),19
Berlin,Echo Show,18
Berlin,Echo Plus,0
Berlin,Echo Look,10
Berlin,Amazon Tap,15
```

25. 생성된 S3 버킷 화면에서 **Objects** 탭 아래 **Upload** 버튼을 클릭합니다.

26. **Add files** 버튼을 클릭하고 CSV 파일 중 하나를 버킷에 업로드합니다.

    (윈도우 폴더를 열어 놓고 파일 끌어 놓기를 하면 한 번에 업로드가 가능하다)

27. 화면 맨 아래로 이동하여 **Upload** 버튼을 클릭합니다.

28. 파일이 정상적으로 전송되었으면 화면위에 **Close** 버튼을 클릭합니다.

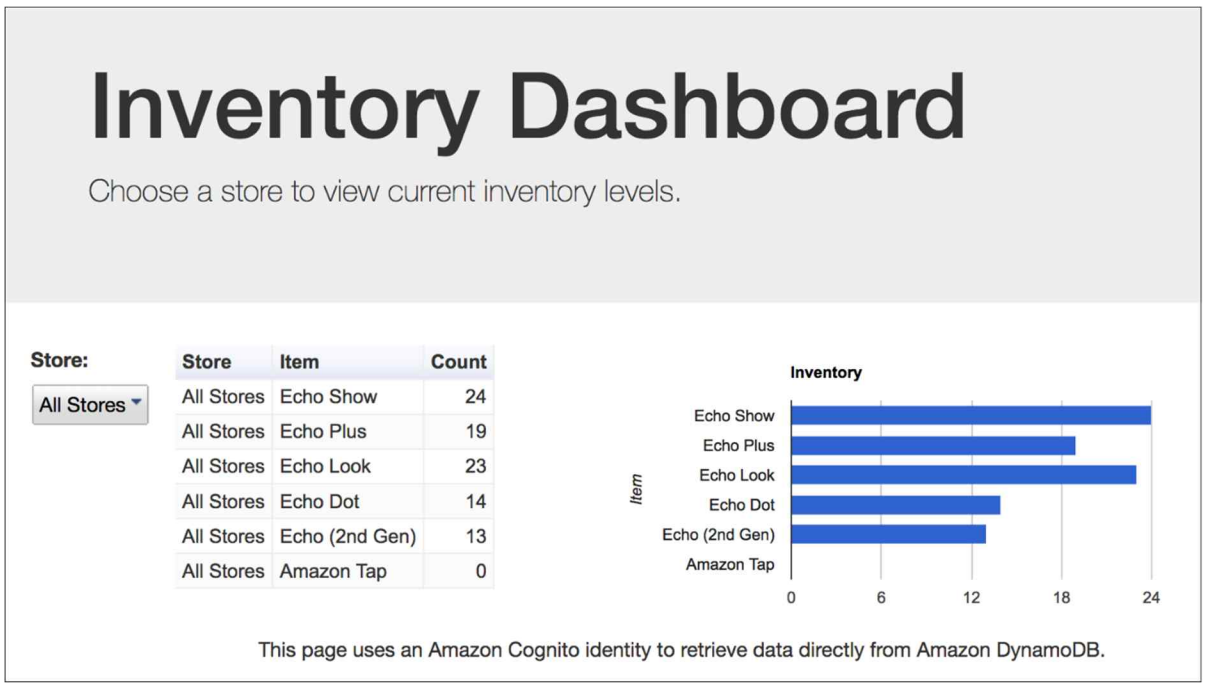
Amazon S3 가 자동으로 Lambda 함수를 트리거 합니다. 그러면 이 함수가 DynamoDB 테이블에 데이터를 로드 합니다.

결과를 볼 수 있도록 서버리스 대시보드 애플리케이션이 제공되어 있습니다.

29. CloudFormation arclab6 Stack 의 Outputs 에서 **Dashboard** URL 을 복사합니다.

30. 새 웹 브라우저 탭을 열어 복사한 URL 을 붙여 넣고 Enter 키를 누릅니다.

대시보드 애플리케이션이 표시되어 버킷에 로드 된 재고 데이터를 보여줍니다. 대시보드가 다음 이미지와 같은 경우 애플리케이션이 DynamoDB 에서 데이터를 검색할 수 있었으며, 이는 Lambda 함수가 성공적으로 트리거 되었음을 입증합니다.



참고 어떤 정보도 표시되지 않는 경우 강사에게 문제 진단을 도와 달라고 요청하십시오.

이 대시보드 애플리케이션은 Amazon S3 에서 정적 웹 페이지로 제공됩니다. 대시보드는 Amazon Cognito 를 통해 *anonymous user* 로 인증됩니다. 이는 대시보드가 DynamoDB 에서 데이터를 검색할 수 있는 충분한 권한을 제공합니다.

DynamoDB 테이블 내에서 데이터를 볼 수도 있습니다.

31. AWS Management Console 브라우저 탭으로 돌아갑니다. **Services** 메뉴에서 **Database > DynamoDB** 를 클릭합니다.

32. 왼쪽 탐색 창에서 **Table** 을 클릭합니다.

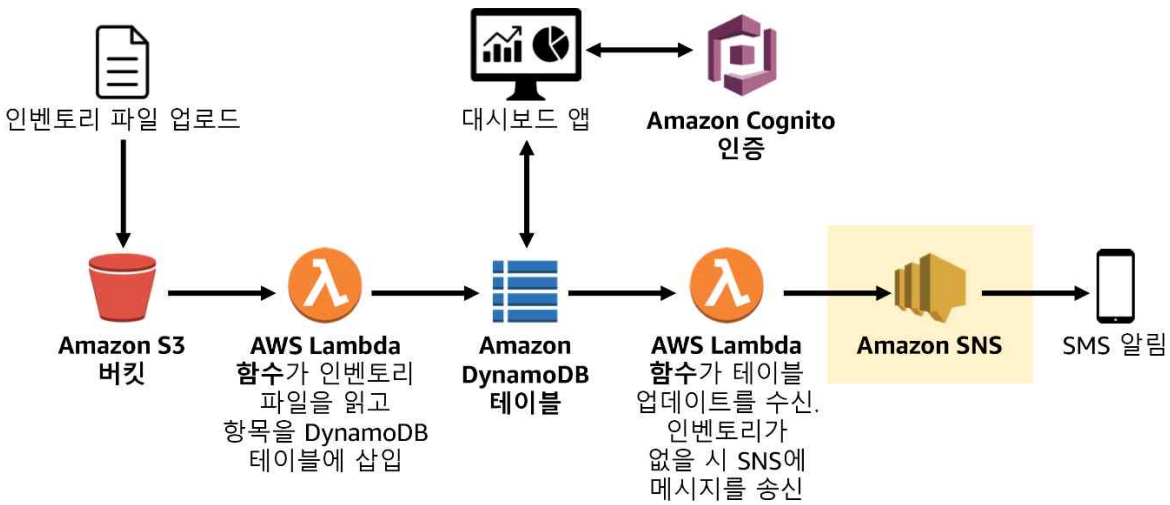
33. **Inventory** 테이블의 이름을 클릭합니다.

34. **Explore table items** 을 클릭합니다.

재고 파일의 데이터가 표시되어 매장, 항목 및 재고 수량을 보여줍니다.

작업 4: 알림 구성

이제 웹 애플리케이션을 통해 재고 파일을 빠르게 볼 수 있으므로 매장에 품목이 부족할 때 재고 관리 담당자에게 알릴 수 있습니다. 이 서버리스 알림 기능에 대해 다음 다이어그램과 같이 Amazon Simple Notification Service(Amazon SNS)를 사용합니다.



Amazon SNS 는 구독 엔드포인트와 및 클라이언트로 메시지를 전달하기 위한 유연한 완전 관리형 게시/구독 메시징 및 모바일 알림 서비스입니다. Amazon SNS 를 사용하면 분산 시스템 및 서비스와 모바일 디바이스와 같은 많은 수의 구독자로 메시지를 팬아웃할 수 있습니다.

35. **Services** 메뉴에서 **Application Integration > Simple Notification Service** 를 클릭합니다.

36. 좌측 패널을 클릭한 후, **Topics** 을 클릭합니다.

37. **Create topic** 을 클릭합니다.

38. Type 은 **Standard** 를 선택합니다.

39. **Create topic** 상자의 **Name** 에 넣고 싶은 고유한 주제 이름을 입력합니다. 예를 들자면 NoStock과 4 자리의 랜덤 숫자의 조합이 될 것입니다.

40. 다음 작업을 위해서 이 주제 이름을 기억합니다.

41. **Create topic** 을 클릭합니다.

알림을 수신하려면 주제를 *구독* 해야 합니다. SMS 및 이메일과 같이 여러 방법을 통해 알림을 수신하도록 구독할 수 있습니다.

42. 페이지의 하단에서, **Create subscription** 을 클릭하여 이메일 또는 SMS 구독 설정을 아래와 같이 진행합니다.

- Amazon SNS 주제를 이메일로 구독하기 위해서:
    - **주제 ARN** 에서 아래와 같은 형식의 생성해 놓은 토픽을 선택해줍니다

      arn:aws:sns:us-east-1:654304825407:NoStock0514 (숫자 다름)

    - **Protocol:** *Email* 을 선택합니다.
    - **Endpoint:** *강의실에서 접근 가능한 유효한 이메일 주소를 입력합니다.*
    - **Create subscription** 을 클릭합니다.

Note: 이메일로 메시지 수신을 시작하기 전에, 구독에 대한 **confirm** 이 필요합니다.

- 구독을 confirm 하기 위해서는
    - 이메일 수신함을 확인하셔서, Amazon SNS 으로부터 온 이메일 본문의 **Confirm subscription** 링크를 클릭하십시오.
    - Amazon SNS 가 웹 브라우저를 열어서 Subscription ID 와 함께 구독 완료 메시지를 보여줄 것입니다.

이제 주제와 구독이 설정되었으므로 SNS 주제로 전송된 모든 메시지는 이메일을 통해 사용자에게 전달됩니다. (아직 메시지 수신 오지 않음)

Note: Amazon SNS 주제를 모바일 텍스트 메시징(SMS)로 구독하고 싶다면 추가 리소스의 이 문서 끝의 추가 리소스 부분 지시대로 진행하십시오.

작업 5: 알림을 전송하는 Lambda 함수 생성

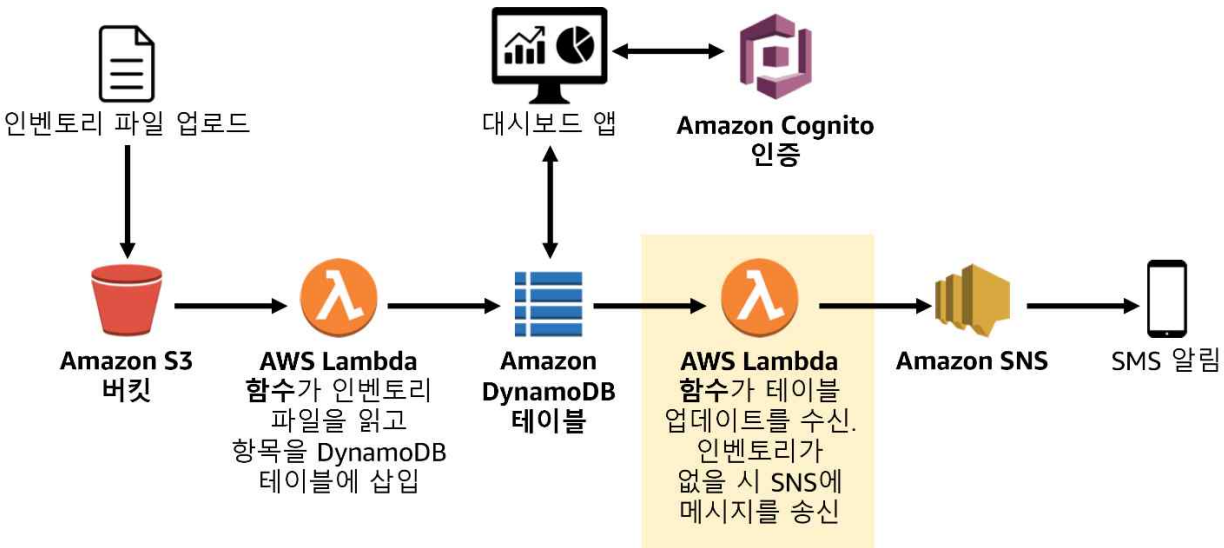
기존의 *Load-Inventory* Lambda 함수를 수정하여 파일이 로드되는 동안 재고 수준을 확인할 수 있지만, 이는 아키텍처 모범 사례가 아닙니다. *Load-Inventory* 함수를 비즈니스 로직으로 과부하시키는 대신, DynamoDB 테이블에 데이터가 로드될 때마다 트리거되는 다른 Lambda 함수를 생성합니다. 이 함수는 *DynamoDB Stream* 으로 트리거됩니다.

이 아키텍처 접근 방식은 몇 가지 이점이 있습니다.

- 각 Lambda 함수는 특정 기능 하나만 수행합니다. 이렇게 하면 코드가 간단해지고 관리가 더 간편해집니다.
- 추가 Lambda 함수를 생성하여 비즈니스 로직을 추가할 수 있습니다. 각 함수는 독립적으로 작동하므로 기존 기능에 영향을 받지 않습니다.

이 작업에서는 DynamoDB 테이블에 로드되는 재고를 모니터링하는 다른 Lambda 함수를 생성합니다. 함수가 품절을 확인하는 경우(수가 0 임) 앞서 생성한 Amazon SNS 주제를 통해 함수에서 알림을 전송합니다.

다음 다이어그램은 이 워크플로를 보여줍니다.



43. **Services** 메뉴에서 **Compute > Lambda** 를 클릭합니다.

44. **Create function** 을 클릭하고 다음을 구성합니다.

- **Create new** 를 선택합니다.
- **Name:** Check-Stock
- **Runtime:** *Python 3.9*
- **Change default execution role** 을 확장합니다.
- **Execution Role:** *Use an existing role*
- **Existing role:** *Lambda-Check-Stock-Role*

- **Create function**을 클릭합니다.

이 역할은 Amazon SNS 로 알림을 보낼 수 있는 권한으로 구성되었습니다.

45. **Code source** 섹션까지 스크롤을 내린 다음, 코드 편집기에서 lambda_function.py 를 더블클릭 해서 열고 표시되는 모든 코드를 삭제합니다.

46. 다음 코드를 복사해 **Code source** 편집기에 붙여 넣습니다.

```
# Stock Check Lambda function
#
# This function is triggered when values are inserted into the Inventory DynamoDB table.
# Inventory counts are checked, and if an item is out of stock, a notification is sent to an SNS topic.


import json, boto3


# This handler is executed every time the Lambda function is triggered
def lambda_handler(event, context):


  # Show the incoming event in the debug log
  print("Event received by Lambda function: " + json.dumps(event, indent=2))


  # For each inventory item added, check if the count is zero
  for record in event['Records']:
    newImage = record['dynamodb'].get('NewImage', None)
    if newImage:


      count = int(record['dynamodb']['NewImage']['Count']['N'])


      if count == 0:
        store = record['dynamodb']['NewImage']['Store']['S']
        item  = record['dynamodb']['NewImage']['Item']['S']


        # Construct message to be sent
        message = store + ' is out of stock of ' + item
        print(message)


        # Connect to SNS
        sns = boto3.client('sns')
        alertTopic = 'NoStock'
        snsTopicArn = [t['TopicArn'] for t in sns.list_topics()['Topics']
```

```
                        if t['TopicArn'].lower().endswith(':' + alertTopic.lower())][0]

        # Send message to SNS
        sns.publish(
          TopicArn=snsTopicArn,
          Message=message,
          Subject='Inventory Alert!',
          MessageStructure='raw'
        )

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

코드에 오타가 있나 잘 확인합니다. 이 코드에서는 다음 단계를 수행합니다.

- 수신 레코드를 루프 처리합니다.
- 재고 수량이 0 인 경우 SNS 주제에 메시지를 전송합니다.

47. 코드 31 번 라인의 *alertTopic* 변수의 값을 NoStock 에서 **작업 4: 알림 구성**의 39 번에서 지정했던 고유한 주제 이름으로 변경하십시오. **(반드시 Topic 을 변경해서 구현한다. 미리 메모장에 복사해 놓고 가져오도록 한다 AWS→SNS→주제로 가서 다시 확인 할 수 있다)**

48. 페이지 상단에서 **Deploy** 를 클릭합니다.

이제 DynamoDB 테이블의 *Inventory* 테이블에 데이터가 추가될 때마다 함수가 트리거 되도록 함수를 구성합니다.

49. 페이지 상단의 **Function overview** 섹션으로 스크롤을 이동합니다.

50. **Add trigger** 를 클릭하고 다음을 구성합니다.

- **Select a trigger:** *DynamoDB*
- **DynamoDB Table:** *Inventory*
- **Add** 를 클릭합니다.

이제 시스템을 테스트할 준비가 되었습니다.

작업 6: 시스템 테스트

이제 Amazon S3 에 재고를 업로드합니다. 그러면 원래의 *Load-Inventory* 함수가 트리거됩니다. 이 함수는 DynamoDB 에 데이터를 로드하고, 그러면 새 *Check-Stock* Lambda 함수가 트리거됩니다. 이 Lambda 함수는 재고가 0 인 품목을 감지하면 SMS 또는 이메일을 통해 Amazon SNS 에 메시지를 전송합니다.

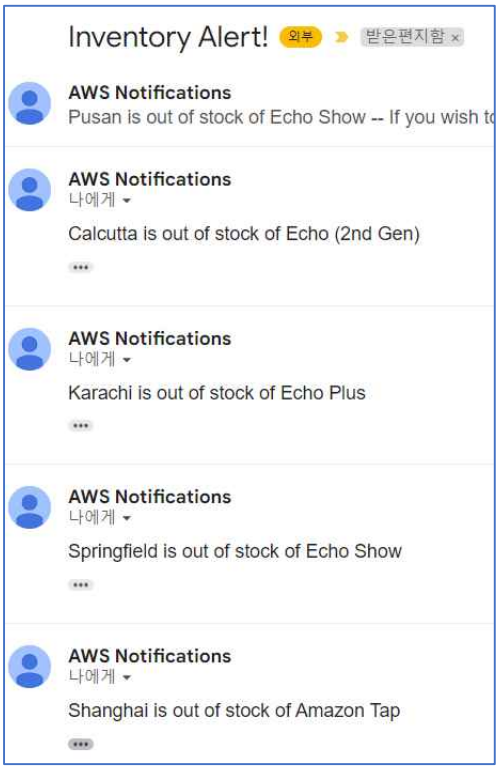
51. **Services** 메뉴에서 **Storage > S3** 를 클릭합니다..
52. 이름이 *inventory-xxxx* 인 버킷을 클릭합니다.
53. 다른 재고 파일 하나를 업로드합니다.
54. Inventory Dashboard 웹 브라우저 탭으로 돌아가서 페이지를 새로 고칩니다.

이제 **Store** 드롭다운 메뉴를 사용해 양쪽 매장의 재고를 볼 수 있습니다.

매장에서 한 품목이 품절임을 나타내는 알림을 SMS 또는 이메일을 통해 수신합니다(모든 재고 파일에는 품절 품목이 하나씩 포함되어 있음).

참고 알림을 받지 못한 경우 잠시 기다렸다가 다른 재고 파일을 업로드해 보십시오. DynamoDB 트리거를 활성화하려면 몇 분 정도 걸릴 수 있습니다. 스팸 방지 설정으로 인해 일부 휴대전화에서 메시지를 수신하지 못할 수 있습니다. 이런 경우 Amazon SNS 주제에 대한 이메일 구독을 추가하십시오. 그러면 확인 이메일이 전송됩니다. 그런 다음 이메일을 열고 **Confirm subscription** 링크를 클릭합니다. 이제 다른 파일을 업로드하여 시스템을 테스트할 수 있습니다.

55. 여러 재고 파일을 동시에 업로드해 보십시오. 어떻게 될 것 같습니까?



추가 리소스

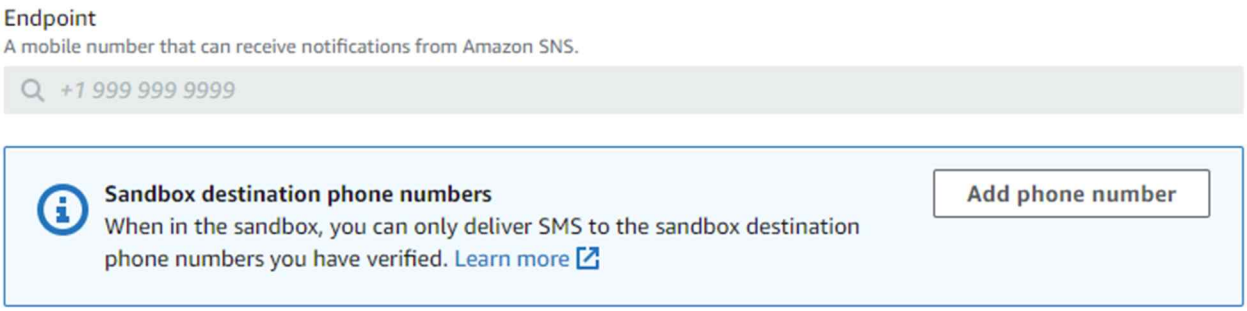
SMS 문자 메시지로 Amazon SNS 주제 구독

- SMS 문자 메시지로 주제를 구독하고 싶다면, SNS 에서 구독에서 **Create subscription** 을 클릭하고 다음을 구성합니다.
- **주제 ARN** 에서 아래와 같은 형식의 생성해 놓은 토픽을 선택해줍니다

  arn:aws:sns:us-east-1:654304825407:NoStock0514 (숫자 다름)

- **Protocol:** *SMS*
- **Endpoint:** 국제 전화번호 형식으로 메세지를 수신할 휴대전화 번호를 입력합니다(예: +82 010 xxxx xxxx). (**아직 입력 할 수 없음**)
- **Create subscription** 을 클릭합니다.

참고 아래 그림과 같이 **Sandbox destination phone numbers** 를 통해서 전화번호를 추가할 수 있는 화면이 보인다면, 아래의 절차대로 진행해 주십시오.



- **Add phone number** 를 클릭합니다.
- 국가를 선택하기 위해서 드롭다운 화살표를 클릭하십시오. 선택하고자 하는 국가를 검색할 수 있는 옵션을 보실 수 있습니다.
- 국제 전화번호 형식으로 메세지를 수신할 휴대전화 번호를 입력합니다(예: +82 010 xxxx xxxx)
- **Verification message language** 에서, 메시지를 수신할 때의 원하는 언어를 선택하십시오. (예: English(United States) 또는 한국어(한국))
- **Add phone number** 을 클릭합니다.
- 휴대전화를 확인하시고, 6 자의 verification code 를 입력합니다.
- **Verify phone number** 을 클릭합니다.
- Topic ARN 항목에, NoStock 으로 끝나는 ARN 을 선택합니다.
- Endpoint 항목에, 방금 전에 추가한 phone number 를 선택합니다.
- **Create subscription** 을 클릭합니다.

참고 SMS 을 통한 메시지를 받지 못한 경우, 강의실에서 액세스할 수 있는 이메일을 제공하여 이메일을 통해 구독할 수 있습니다. 이메일 구독을 생성한 후 확인 이메일을 받게 됩니다. 이메일을 열고 **Confirm subscription** 링크를 클릭합니다.

이제 SNS 주제와 구독 설정을 완료했기 때문에, SNS 주제로 보내진 메시지는 문자로 받으실 수 있을 것입니다.

DynamoDB 의 테이블로 가서 앞에서 와 같이 임의의 항목 1 개의 Count 를 0 으로 수정하면 휴대폰으로 문자 메시지가 수신된다

결론

축하합니다! 다음 작업이 성공적으로 완료되었습니다.

- Lambda 함수 생성
- Amazon S3 및 DynamoDB 에서 Lambda 함수 트리거
- 알림을 전송하도록 Amazon SNS 를 구성

실습 종료

다음 순서 따라 실습 과정에서 생성된 리소스를 정리하십시오.

1. **Lambda:** Check-Stock, Load-Inventory Function 삭제(2 개 동시 삭제 가능)
2. **SNS:** topic 삭제, subscribe 모두 개별 삭제
3. **S3:** Bucket(inventory-로 시작하는 이름)의 파일과 Bucket 삭제
4. **S3:** Bucket(cf-templates 으로 시작하는 이름)의 파일과 Bucket 삭제
5. **CloudFormation:** Stack 삭제 (삭제 완료까지 대기, 수분 이내)
6. **끝.**