5-4 Appendix
S3 バケットにある画像データを削除する

クライアント側の Photo.vue に実装されている deleteImage から呼び出される API、つまり/image/{id}に対する DELETE リクエストで処理されるのはあくまでも DynamoDB のテーブル上にあるレコードの削除処理のみであり、S3 バケットに保存されている実データは削除されないということは先述しました。

そのままでは実データが S3 バケット上に残り続けてしまうため、削除するように実装していきます。ここでは DynamoDB ストリームと AWS Lambda を組み合わせて API リクエストとは非同期に処理を行う形式で実装していきます。

DynamoDB ストリームとは

DynamoDB ストリームを利用すると DynamoDB を利用している場合に、テーブルにあるデータに対する変更処理をリアルタイムに近い状態でキャプチャすることが可能になります。この機能を利用すると、DynamoDB のテーブルに対してアプリケーションが新たなデータを作成するたびにそれを別のデータストアに反映していくといったようなことが可能になります。

例えば、DynamoDB では全テーブルをスキャンするような処理にはあまり向いていません。そのため検索用途では別途、検索エンジンとして Elasticsearch を用いたりしますがその Elasticsearch に対してデータを反映したり、同様にレポーティング用に集計処理が得意なリレーショナル・データベース（RDBMS）へとデータを反映したりといった場合に利用できます。

または、認証処理のデータストアとして DynamoDB を利用する場合などに、新しくデータが作成されたら確認メールを送信するといったユースケースもあり得るでしょう。このように 2 箇所以上のデータストアへの更新や処理をアプリケーション側から行おうとする場合、どちらか一方だけ書き込みが失敗した場合などのハンドリングが必要になるため処理が複雑になりがちですが、DynamoDB ストリームを利用することでそれらを非同期か

つ簡単に行うことが可能になると言えるでしょう。

DynamoDB ストリームはテーブルの項目単位の変更を時系列でシーケンシャルに保持しています。また、それらの情報は最大で 24 時間保存されています。24 時間経過した後は自動的に削除されます。また、変更前と変更後の両方の内容を参照できます。また、その情報はストリームに出現するのは 1 回だけということが保証されており、その順序は実際に項目に対して行った変更の順序と同じになります。ストリームはテーブル単位で有効にできます。

実際の利用イメージは Amazon Kinesis のストリームに近いものになります。DynamoDB ストリームの API は Kinesis Data Streams の API と意図的に似せてデザインされており、シャードといった概念も同様になります（全てのアクションが 100%同じというわけではありません）。DynamoDB ストリームを利用するアプリケーションの開発には Kinesis Adapter 経由で Kinesis Client Library（KCL）を利用することも可能です。また、当然のことながら AWS Lambda を利用することも可能です。

今回はこの DynamoDB ストリームを利用して、既に用意されている削除 API が呼び出されて DynamoDB 上の該当データが削除されたことをストリームによってキャプチャし、Lambda 関数をもちいて S3 バケットに保存されている実際のオブジェクトを削除するという処理を実装します。

DynamoDB ストリームの有効化

今回はこれまで利用している photos テーブルでストリームを有効にします。

```
$ aws dynamodb update-table --table-name photos ¥
--stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES ¥
--region ap-northeast-1
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:ap-northeast-1:111122223333:table/photos",
--- 省略 ---
        "LatestStreamArn":
"arn:aws:dynamodb:ap-northeast-1:111122223333:table/photos/stream/YYYY-MM-DDTHH:MM:S
S.SSS"
```

```
    }
}
```

ここで指定している StreamViewType というのはストリームに書き込まれる情報の種別で以下の値が指定可能です。

指定する値	意味
KEYS_ONLY	変更された項目のキー属性のみ
NEW_IMAGE	変更後の項目に含まれる情報全て
OLD_IMAGE	変更前の項目に含まれる情報全て
NEW_AND_OLD_IMAGES	変更前と変更後、両方の項目に含まれる情報

今回はひとまず NEW_AND_OLD_IMAGES を指定していますが、アプリケーションの特性は必要性に応じて選択してください。

また、コマンド実行結果に含まれる LatestStreamArn の値を控えておいてください。これは後ほど Lambda 関数のトリガーとして設定する際に必要になる値です。

Lambda 関数の作成

まずは任意のディレクトリを作成して移動し、以下の内容を lambda_function.py という名前で作成してください。

```
import boto3
import json
import logging
import os

logger = logging.getLogger()
logger.setLevel(logging.INFO)

s3 = boto3.client('s3', region_name = "ap-northeast-1")

def lambda_handler(event, context):
  for record in event['Records']:
    eventName = record['eventName']
    if eventName == "REMOVE":
```

```
    key = record['dynamodb']['Keys']['photo_id']['S']
    type = record['dynamodb']['OldImage']['type']['S'].split('/')[1]
    object_key = key + '.' + type

    s3.delete_object(
        Bucket = os.getenv('BUCKET_NAME'),
        Key = object_key
    )
    print('Successfully deleted ' + object_key + '.')
return 'Successfully processed {} records.'.format(len(event['Records']))
```

Lambda 関数の内容を見ていきます。まず、DynamoDB のテーブルに対して作成や更新処理を行うとそれら一件一件がレコードとしてストリームに書き込まれます。それを Lambda 関数から取得した結果が event に含まれています。次にレコード件数分処理をしますが、ポイントになるのは eventName です。ここには DynamoDB のテーブルに対して行われた処理が何かを示す値が格納されています。以下がその値となります。

- INSERT - 新しい項目が追加された
- MODIFY - 既存の項目に対して 1 つ以上の属性が変更された
- REMOVE - テーブルから項目が削除された

今回の場合は API により DynamoDB の photos テーブルからデータが削除された場合のみ処理を実行するため、eventName の値が REMOVE だった場合のみ処理を実行しています。実際の処理の内容はシンプルに S3 の delete_object メソッドを実行して指定のバケットから該当のオブジェクトを削除しているだけとなっています。

次に Lambda 関数に紐付ける IAM ロールの作成を行います。今回は通常必要となる CloudWatch Logs へのアクセス権に加えて DynamoDB ストリームに対するポーリングおよびレコードの取得処理と S3 上のオブジェクトの削除を許可する必要があります。

まず、次のファイルを trustpolicy.json として用意します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

次に作成したファイルを利用してIAMロールを作成します。ロール名はdelete-image-roleとします。ファイルの指定はご自身の環境にあわせて適宜置き換えて実行してください。

```
$ aws iam create-role --role-name delete-image-role --assume-role-policy-document
file://trustpolicy.json
{
    "Role": {
--- 省略 ---
        "RoleName": "delete-image-role",
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:role/delete-image-role"
    }
}
```

コマンド実行時のレスポンスに含まれる IAM ロールの ARN を確認し、控えておきます。これは後の Lambda 関数作成時に必要になります。

続いて、IAM ロールにポリシーを適用します。次の内容で permission.json というファイルを作成してください。"Resource"で指定している DynamoDB ストリームの ARN はストリームを有効にした際に確認したストリームの ARN を指定してください。S3 バケットもご自身のものを指定してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:*",
                "dynamodb:ListStreams"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListStreams",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream"
            ],
            "Resource": [
                "<確認した DynamoDB ストリームの ARN>"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::serverless-app-photos/*"
            ]
        }
    ]
}
```

これを先ほど作成した IAM ロールに割り当てます。ポリシー名はここではひとまず delete-image-policy としていますが何でも構いません。

```
$ aws iam put-role-policy --role-name delete-image-role --policy-name
delete-image-policy --policy-document file://permission.json
```

今回もデプロイには AWS SAM を利用することとします。以下の内容でテンプレートファイルを作成してください。サンプルでは template.yaml という名前でファイルを作成したものとします。BUCKET_NAME はご自身の環境に読み替えてください。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: ''
Resources:
  postImages:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: lambda_function.lambda_handler
      Runtime: python3.6
```

```
      Description: ''
      MemorySize: 128
      Timeout: 15
      Role: 'arn:aws:iam::111122223333:role/delete-image-role'
      Environment:
        Variables:
          BUCKET_NAME: serverless-app-photos
```

なお、今回の処理ではイベントソースとなる DynamoDB のテーブルは元々別のステップで作成されているため、このスタックでは設定ができません。したがって、Lambda 関数作成後に CLI をもちいて別途設定します。

作成したらパッケージ化とデプロイを行います。以下のコマンドを実行してください。なお、スタック名はこれまでに指定したものとは別のものにしてください。同じ名前だと全てのリソースが置き換わってしまいます。成果物の保存先として指定している S3 バケットはご自身の環境に置き換えて実行してください。

```
$ aws cloudformation package --template-file template.yaml ¥
--output-template-file template-output.yaml ¥
--s3-bucket serverless-app-sam

$ aws cloudformation deploy ¥
--template-file template-output.yaml ¥
--stack-name delete-image ¥
--capabilities CAPABILITY_IAM --region ap-northeast-1
```

作成完了まで少し待ったのち以下のコマンドを実行して、作成された Lambda 関数の名前を確認します。

```
$ aws cloudformation describe-stacks --stack-name delete-image ¥
--region ap-northeast-1
{
    "Stacks": [
        {
--- 省略 ---
            "Outputs": [
                {
                    "OutputKey": "FunctionName",
                    "OutputValue": "delete-image-deleteImageFromS3-XW5QR0IVJONN"
                }
            ],
--- 省略 ---
```

```
        }
    ]
}
```

確認するのはコマンド実行結果の Outputs に含まれる値になります。OutputValue の値がこのデプロイで作成された Lambda 関数の名前になりますのでこれを控えておいてください。

最後に作成した Lambda 関数のイベントソースとして、DynamoDB ストリームを指定します。Amazon Kinesis Streams 同様にストリームベースのイベントソースなのでイベントソースの設定情報はイベントソースマッピングとして AWS Lambda 側で保有することになります。そのため、作成した Lambda 関数にイベントソースマッピングを作成します。

```
$ aws lambda create-event-source-mapping --event-source- ¥
arn <確認した DynamoDB ストリームの ARN> ¥
--function-name <確認した Lambda 関数名> --enabled ¥
--starting-position LATEST
```

完了したら、画面上から画像を削除していた後、S3 バケットを確認してみてください。実際にオブジェクトが削除されていることが確認できるはずです。