Assignment 6 - Kinesis

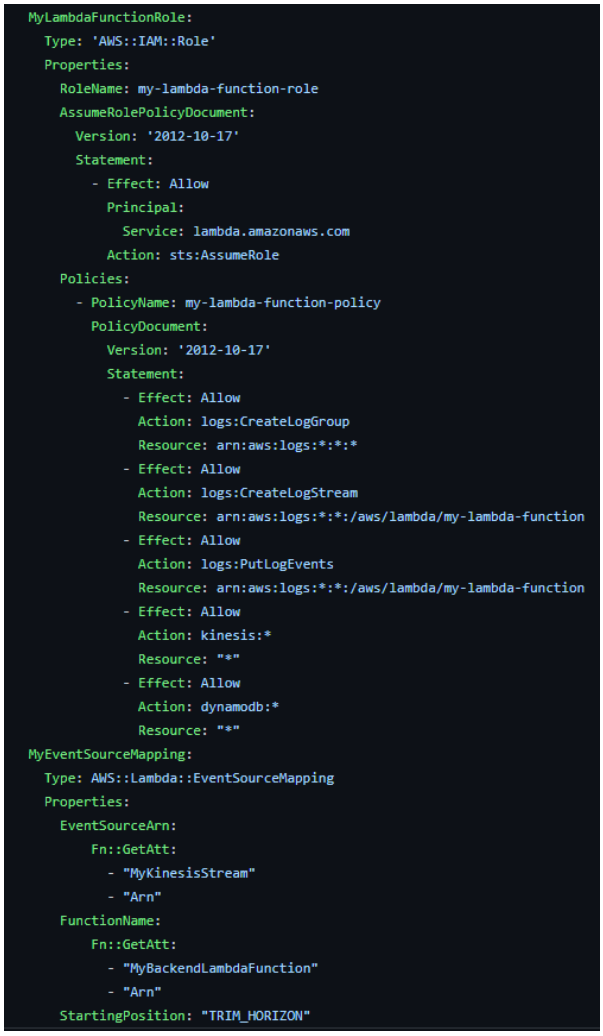
Problem Statement

- Create a Kinesis data stream. set up a mock data generator to generate a stream of data records in JSON format. Write the data records to the Kinesis data stream using the put_record() method of the boto3 client for Kinesis.
- Create a DynamoDB table to store the processed data. Create a Lambda function triggered by the Kinesis data stream. Implement a function in the Lambda function to process the data records, extract relevant information, and store it in the DynamoDB table.
- Ensure that all the resources are created using CloudFormation templates.

Solution

This CloudFormation template creates a Kinesis stream, DynamoDB table, and two Lambda functions. The first Lambda function generates data and puts it in the Kinesis stream, while the second Lambda function triggers when new data is added to the stream and extracts relevant information to store it in the backend DynamoDB.

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  MyKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      Name: my-kinesis-stream
      ShardCount: 1
  MyDynamoDBTable:
    Type: "AWS::DynamoDB::Table"
    Properties:
      AttributeDefinitions:
        - AttributeName: "sequenceNumber"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "sequenceNumber"
          KeyType: "HASH"
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
      TableName: "my-table-name"
  MyLambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      FunctionName: my-lambda-function
      Handler: app.lambda_handler
      Role: !GetAtt MyLambdaFunctionRole.Arn
      Code:
        S3Bucket: test-dg-assign
        S3Key: KinesisCreation.zip
      Runtime: python3.8
  MyBackendLambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      FunctionName: my-backend-lambda-function
      Handler: backend.lambda_handler
      Role: !GetAtt MyLambdaFunctionRole.Arn
      Code:
        S3Bucket: test-dg-assign
        S3Key: KinesisBackend.zip
      Runtime: python3.8
```



```yaml
  MyLambdaFunctionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: my-lambda-function-role
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: my-lambda-function-policy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: logs:CreateLogGroup
                Resource: arn:aws:logs:*:*:*
              - Effect: Allow
                Action: logs:CreateLogStream
                Resource: arn:aws:logs:*:*:/aws/lambda/my-lambda-function
              - Effect: Allow
                Action: logs:PutLogEvents
                Resource: arn:aws:logs:*:*:/aws/lambda/my-lambda-function
              - Effect: Allow
                Action: kinesis:*
                Resource: "*"
              - Effect: Allow
                Action: dynamodb:*
                Resource: "*"
  MyEventSourceMapping:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      EventSourceArn:
          Fn::GetAtt:
            - "MyKinesisStream"
            - "Arn"
      FunctionName:
          Fn::GetAtt:
            - "MyBackendLambdaFunction"
            - "Arn"
      StartingPosition: "TRIM_HORIZON"
```

Lambda function code for generating random data and putting it into the stream.

```python
import json
import boto3
import time

kinesis_client = boto3.client('kinesis')
stream_name = "my-kinesis-stream"
PartitionKey = "111111"

def randomData():
    data_record = {
        'timestamp': int(time.time())
    }
    return data_record

def lambda_handler(event, context):
    data_record = randomData()
    response = kinesis_client.put_record(
        StreamName=stream_name,
        Data=json.dumps(data_record, indent=2).encode('utf-8'),
        PartitionKey=PartitionKey
    )
    return {
        'statusCode': 200,
        'body': json.dumps(response)
    }
```

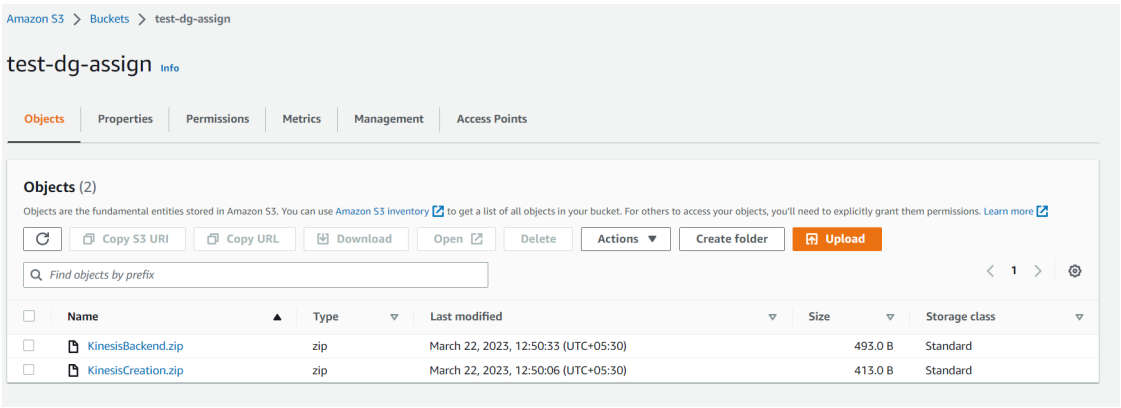
The Lambda function code extracts relevant data from the event and stores it in the DynamoDB table.

```python
import json
import boto3
import base64


dynamodb = boto3.client('dynamodb')


def lambda_handler(event, context):
    for records in event['Records']:
        partitionKey=records['kinesis']['partitionKey']
        sequenceNumber=records['kinesis']['sequenceNumber']
        data=records['kinesis']['data']
        data = base64.b64decode(data).decode('utf-8')
    response = dynamodb.put_item(
        TableName='my-table-name',
        Item={
            "sequenceNumber": {
                "S": sequenceNumber
            },
            "partitionKey": {
                "S": partitionKey
            },
            "data": {
                "S": data
            }
        })
    return {
        'statusCode': 200,
        'body': json.dumps(response)
    }
```

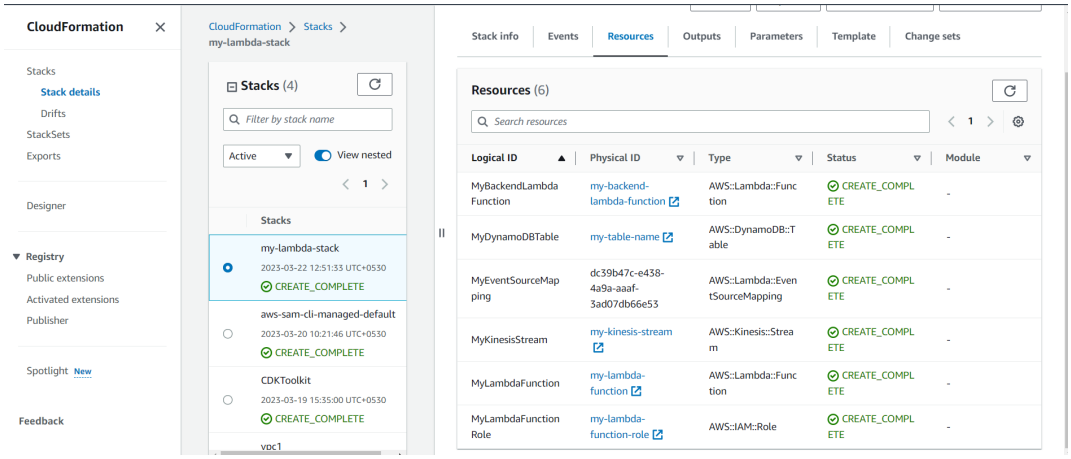
We will compress the two Python files and copy them to the S3 bucket, and then create a stack using CloudFormation.

```
[root@ip-172-31-0-37 cloudformation-ws]# zip  KinesisCreation.zip app.py
  adding: app.py (deflated 47%)
[root@ip-172-31-0-37 cloudformation-ws]# zip KinesisBackend.zip backend.py
  adding: backend.py (deflated 58%)
[root@ip-172-31-0-37 cloudformation-ws]# ls
app.py  backend.py  KinesisBackend.zip  kinesis-creation.yaml  KinesisCreation.zip
[root@ip-172-31-0-37 cloudformation-ws]# aws s3 cp KinesisCreation.zip s3://test-dg-assign/KinesisCreation.zip
upload: ./KinesisCreation.zip to s3://test-dg-assign/KinesisCreation.zip
[root@ip-172-31-0-37 cloudformation-ws]# aws s3 cp KinesisBackend.zip  s3://test-dg-assign/KinesisBackend.zip
upload: ./KinesisBackend.zip to s3://test-dg-assign/KinesisBackend.zip
[root@ip-172-31-0-37 cloudformation-ws]# aws cloudformation create-stack --stack-name my-lambda-stack --template-body file://kinesis-creation.yaml --capabilities CAPABILITY
_NAMED_IAM
{
    "StackId": "arn:aws:cloudformation:ap-south-1:628906266361:stack/my-lambda-stack/2c21a6e0-c882-11ed-b565-0601900d0d0a"
}
[root@ip-172-31-0-37 cloudformation-ws]#
```

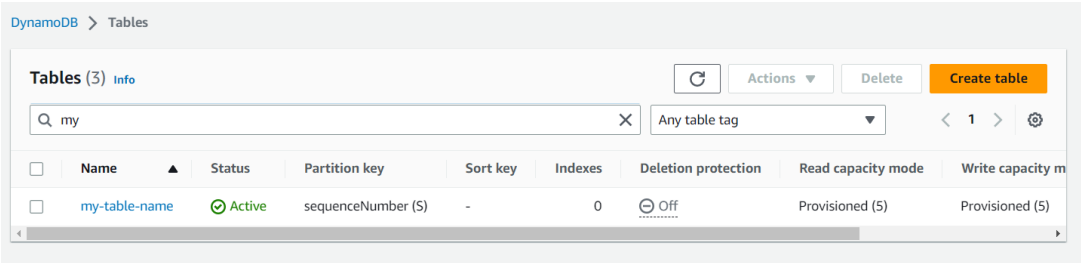
Two compressed files are uploaded to the bucket successfully.



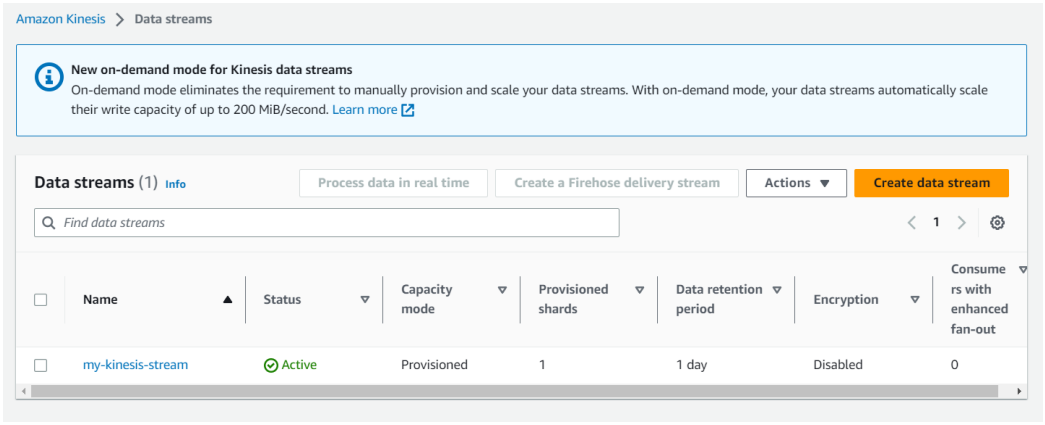
Cloud formation created the resources successfully.



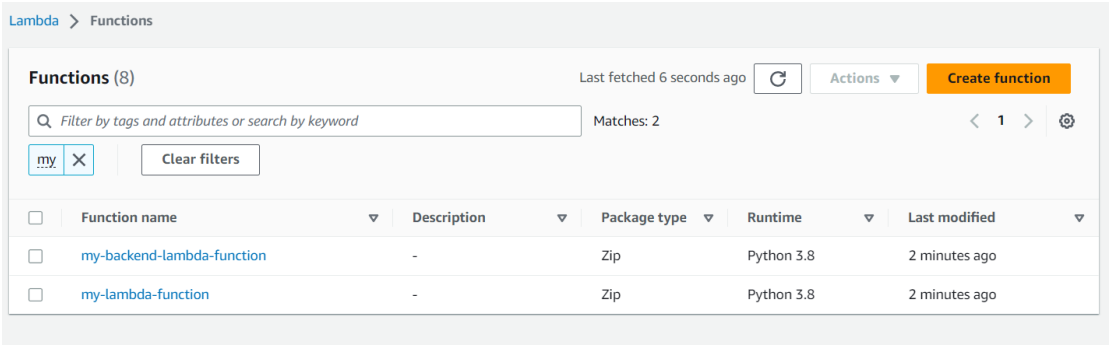
DynamoDB Table



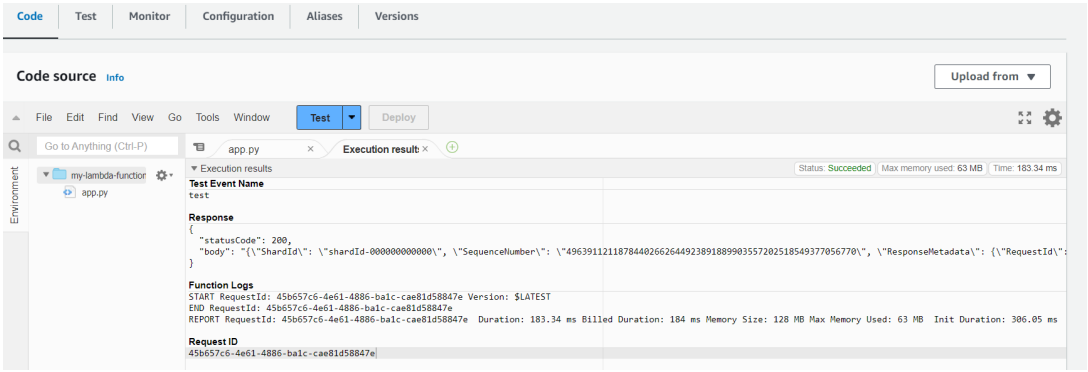
Kinesis Data Stream



Lambda functions



The first Lambda function will successfully execute and put random data into the stream. As soon as data is placed into the stream, a second Lambda function will be triggered and process the data, storing it in DynamoDB.



Data has been successfully stored in the DynamoDB table.