

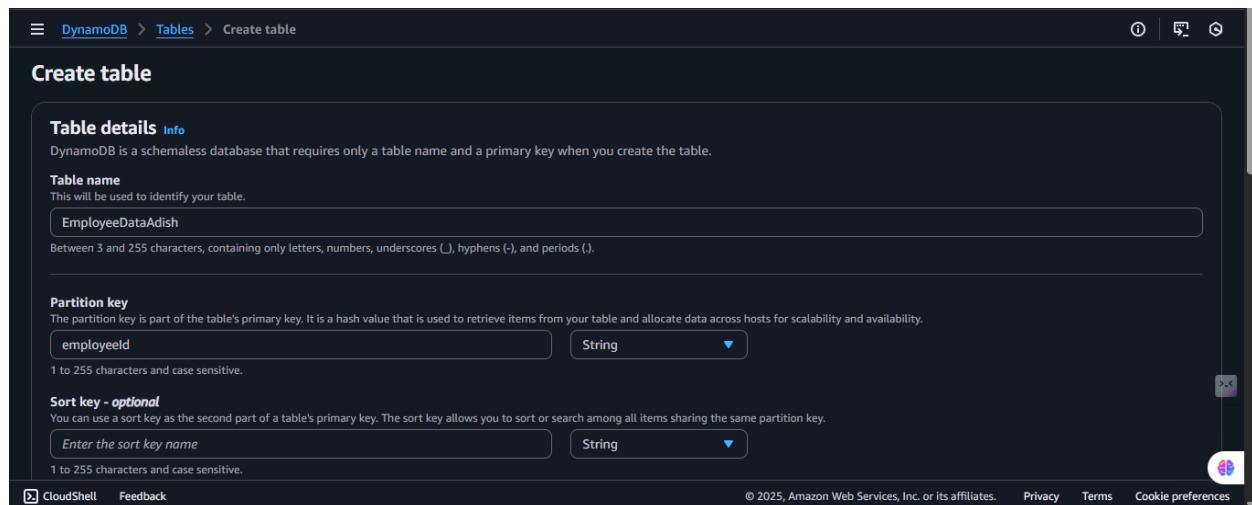
1. DynamoDB Setup

1. Go to **AWS Console** → **DynamoDB** → **Create table**

- **Table name:** EmployeeData
- **Partition key:** employeeId (String)
- Keep defaults for the rest.

2. Add a few test items manually:

```
{  
  "employeeId": "E101",  
  "name": "John Doe",  
  "role": "Developer",  
  "salary": 60000  
}
```



The screenshot shows the AWS DynamoDB 'Create item' interface. At the top, there are navigation links: 'DynamoDB' > 'Explore items: EmployeeDataAdish' > 'Create item'. On the right, there are 'Form' and 'JSON view' buttons. The main area is titled 'Create item' with the sub-section 'Attributes'. A table lists one attribute: 'employeeId - Partition key'. The 'Value' column contains a JSON object:

```
{ "employeeId": "E101", "name": "John Doe", "role": "Developer", "salary": 60000 }
```

The 'Type' column shows 'String'. Below the table are 'Add new attribute' and 'Remove' buttons. At the bottom are 'Cancel' and 'Create item' buttons.

2. SNS Setup

1. Go to **Amazon SNS** → **Topics** → **Create topic**
 - **Type:** Standard
 - **Name:** `dynamodb-update-alerts`
2. After creation, note the **Topic ARN** (e.g. `arn:aws:sns:eu-west-2:123456789012:dynamodb-update-alerts`).
3. **Subscribe your email:**
 - Under the topic → **Create subscription**
 - Protocol: `Email`
 - Endpoint: your email address.
 - Confirm the subscription from your inbox

Amazon SNS > Topics > Create topic

Details

Type [Info](#)
Topic type cannot be modified after topic is created

FIFO (first-in, first-out)
• Strictly-preserved message ordering
• Exactly-once message delivery
• Subscription protocols: SQS

Standard
• Best-effort message ordering
• At-least once message delivery
• Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - optional [Info](#)
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

Maximum 100 characters.

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN X

Protocol
The type of endpoint to subscribe

Endpoint
An email address that can receive notifications from Amazon SNS.

After your subscription is created, you must confirm it. [Info](#)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



AWS Notification - Subscription Confirmation ➔ Inbox



AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾

2:49 PM (0 minutes ago)

☆ ☺ ↻ ⋮

You have chosen to subscribe to the topic:

arn:aws:sns:eu-west-2:975050024946:dynamodb-update-alerts-adish

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)



Write your reply to generate with AI

Yes

No

Follow up

↳ Reply

↗ Forward



AI Reply



Simple Notification Service

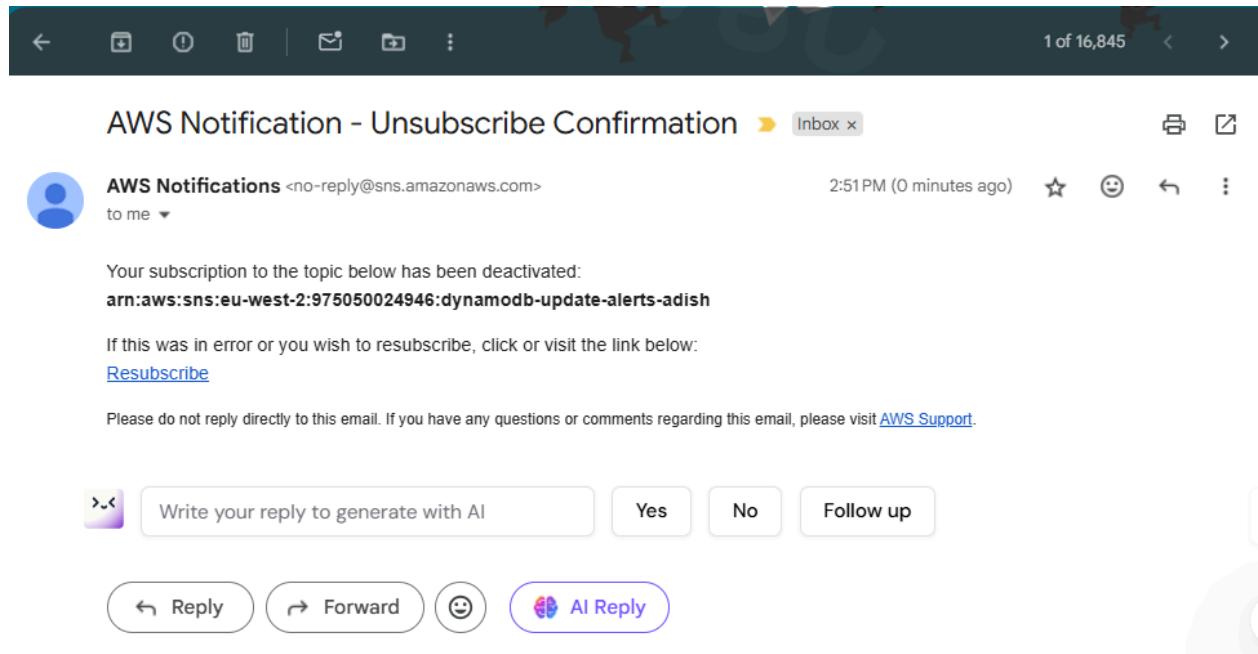
Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

arn:aws:sns:eu-west-2:975050024946:dynamodb-update-alerts-adish:bc8be414-cd23-4462-a6da-554e7ffaa2a5

If it was not your intention to subscribe, [click here to unsubscribe](#).



3. Lambda IAM Role

1. Go to **IAM → Roles → Create role**
 - Trusted entity: **Lambda**
 - Attach these policies:
 - **AmazonDynamoDBFullAccess**
 - **AmazonSNSFullAccess**
 - **AWSLambdaBasicExecutionRole**
2. Name it: **LambdaDynamoDBUpdateRole**

Lambda Function Code

Go to **Lambda → Create function**

- Name: `DynamoDBItemChangeAlert`
- Runtime: `Python 3.13 (or latest)`
- Role: `LambdaDynamoDBUpdateRole`

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The top navigation bar includes 'Lambda', 'Functions', and 'Create function'. The main section is titled 'Create function' with a 'Info' link. It asks to choose one of three options: 'Author from scratch' (selected), 'Use a blueprint', or 'Container image'. Below this is a 'Basic information' section with fields for 'Function name' (set to 'DynamoDBItemChangeAlert'), 'Runtime' (set to 'Python 3.13'), and 'Architecture' (set to 'AWS Lambda'). The bottom of the screen shows standard AWS footer links: CloudShell, Feedback, © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

The screenshot shows the 'Create function' wizard continuing through the 'Permissions' step. The top navigation bar remains the same. The main section is titled 'Permissions' with a 'Info' link. It states that Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. It includes sections for 'Change default execution role' (with options for 'Create a new role with basic Lambda permissions' (selected), 'Use an existing role', and 'Create a new role from AWS policy templates') and 'Existing role' (set to 'aws-lambda-full-access'). At the bottom, there's a 'Additional configurations' section with a note about setting up networking, security, and governance. The footer links are identical to the previous screenshot.

The screenshot shows the AWS Lambda Functions configuration page for a function named 'DynamoDBItemChangeAlert'. The 'Configuration' tab is selected. On the left, a sidebar lists options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, and Monitoring and operations. The 'Environment variables' option is currently selected. In the main panel, under 'General configuration', there is a table with columns for Description, Memory, Timeout, SnapStart, and Ephemeral storage. The 'Description' field is empty. 'Memory' is set to 128 MB, 'Timeout' is set to 1 min 3 sec, 'SnapStart' is set to None, and 'Ephemeral storage' is set to 512 MB. An 'Edit' button is located in the top right corner of this section. At the bottom of the page, there are links for CloudShell, Feedback, and a footer with copyright information and links for Privacy, Terms, and Cookie preferences.

The screenshot shows the same AWS Lambda Functions configuration page as the previous one, but with a green success message at the top: 'Successfully updated the function DynamoDBItemChangeAlert.' The 'Configuration' tab is still selected, and the 'Environment variables' section is visible. The environment variable 'SNS_TOPIC_ARN' is listed with the value 'arn:aws:sns:eu-west-2:975050024946:dynamodb-update-alerts-adish'. A search bar above the table allows for finding environment variables. The rest of the interface is identical to the first screenshot, including the sidebar and footer.

5. DynamoDB Stream Setup

1. Go to your **DynamoDB table** → **Exports and streams** → **Manage Stream**
2. Enable stream and choose **New and old images**.
3. Click **Enable**.
4. Then go to **Lambda** → **Configuration** → **Triggers** → **Add trigger**
 - Select **DynamoDB**
 - Choose your table
 - Batch size: **1**
 - Starting position: **LATEST**
 - Enable trigger

The screenshot shows the AWS DynamoDB console with the path: DynamoDB > Tables > EmployeeDataAdish. The left sidebar includes options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Under DAX, there are Clusters and Subnet groups. The main area displays the 'Tables (2)' section with filters for Any tag key and Any tag value, and a search bar for Find tables. It lists two tables: 'bharathi-table' and 'EmployeeDataAdish'. The 'EmployeeDataAdish' table is selected. The top navigation bar has tabs for Monitor, Global tables, Backups, Exports and streams (which is the active tab), and Permissions. Below the tabs, the 'Exports to S3' section shows a table with columns for Export ARN, Destination S3 bucket, Status, and Export job start time (UTC+0). A button 'Export to S3' is visible. The bottom of the page includes standard AWS footer links: © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

This screenshot continues from the previous one, showing the 'DynamoDB stream details' and 'Amazon Kinesis data stream details' sections. The 'DynamoDB stream details' section shows 'Stream status' as Off and a 'Turn on' button. The 'Amazon Kinesis data stream details' section shows 'Status' as Off and a 'Turn on' button. The rest of the interface is identical to the first screenshot, including the sidebar and footer.

This screenshot shows the 'Turn on DynamoDB stream' configuration dialog. The title is 'Turn on DynamoDB stream'. The 'DynamoDB stream details' section describes capturing item-level changes and pushing them to a DynamoDB stream for access via the Streams API. The 'View type' section allows choosing between New and old images, Key attributes only, New image, and Old image. The 'New and old images' option is selected. The 'New image' option is highlighted in yellow. The dialog includes 'Cancel' and 'Turn on stream' buttons at the bottom right. The footer includes standard AWS links: © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

DynamoDB > Tables > EmployeeDataAdish

DynamoDB

- Dashboard
- Tables**
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations
- Reserved capacity
- Settings

DAX

- Clusters
- Subnet groups

No exports

Export to S3

DynamoDB stream details

Stream status: On

Resource-based policy: Not active

View type: New image

Latest stream ARN: arn:aws:dynamodb:eu-west-2:975050024946:table/EmployeeDataAdish/stream/2025-10-20T09:38:02.489

Turn off

Trigger (0)

Use triggers to invoke an AWS Lambda function every time an item is changed, and then your DynamoDB stream is updated.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lambda > Functions > DynamoDBItemChangeAlert

Code **Test** **Monitor** **Configuration** **Aliases** **Versions**

General configuration

Triggers (0) Info

Find triggers

No triggers

No triggers are configured.

Add trigger

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lambda > Add triggers

Add trigger

Trigger configuration Info

DynamoDB

aws database event-source-mapping nosql polling

DynamoDB table

Choose or enter the ARN of a DynamoDB table.

arn:aws:dynamodb:eu-west-2:975050024946:table/EmployeeDataAdish

Event poller configuration

Activate trigger

Select to activate the trigger now. Keep unchecked to create the trigger in a deactivated state for testing (recommended).

Enable metrics

Monitor your event source with metrics. You can view those metrics in CloudWatch console. Enabling this feature incurs additional costs. [Learn more](#)

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the 'Event poller configuration' section of the AWS Lambda console. It includes fields for activating the trigger, enabling metrics, setting batch size (1), specifying starting position (Latest), and defining a batch window (optional). The status bar at the bottom indicates the page is from 2025 and includes links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

The screenshot shows the 'Configuration' tab of the AWS Lambda function 'DynamoDBItemChangeAlert'. The 'Triggers' section displays one trigger named 'DynamoDB: EmployeeDataAdish', which is associated with the 'EmployeeData' table and is currently enabled. The sidebar on the left lists other configuration options like General configuration, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, and RDS databases. The status bar at the bottom indicates the page is from 2025 and includes links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

6. Testing

1. Go to **DynamoDB** → **Explore table items**
2. Select an existing item (e.g. **E101**) and update a field:

```
{  
  "employeeId": "E101",  
  "name": "John Doe",  
  - "salary": 60000  
  + "salary": 75000  
}  
}
```

DynamoDB > Explore items: EmployeeDataAdish > Edit item

Edit item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Attribute name	Value	Type	Remove
employeeId - Partition key	<pre>{ "employeeId": "E101", "name": "John Doe", "role": "Developer", "salary": 750000 }</pre>	String	

Add new attribute ▾

Cancel Save Save and close

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lambda > Functions > DynamoDBItemChangeAlert

- Code
- Test**
- Monitor
- Configuration
- Aliases
- Versions

Executing function: succeeded ([logs](#))
▶ Details

Test event [Info](#)
To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action
 Create new event Edit saved event

Invocation type
 Synchronous
Executes the Lambda function and blocks until receiving the function's response, with a maximum timeout of 15 minutes. Returns function output or error details directly to the calling application.
 Asynchronous
Enqueues the Lambda function for execution and returns immediately with a request ID. Function processes independently, with results optionally sent to a configured destination like SQS, SNS, or EventBridge.

CloudWatch Logs Live Tail Save Test

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Lambda > Functions > DynamoDBItemChangeAlert

Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional
dynamo

Event JSON

```

1  {
2    "Records": [
3      {
4        "eventID": "1",
5        "eventName": "MODIFY",
6        "eventversion": "1.1",
7        "eventSource": "aws:dynamodb",
8        "awsRegion": "eu-west-2",
9        "dynamodb": {
10          "Keys": {
11            "UserId": { "S": "101" }
12          },
13          "OldImage": {
14            "UserId": { "S": "101" },
15            "version": { "N": "1" }
16          }
17        }
18      }
19    ]
20  }
```

Format JSON

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Testing by lambda testing icon

```
{  
  "Records": [  
    {  
      "eventID": "1",  
      "eventName": "MODIFY",  
      "eventVersion": "1.1",  
      "eventSource": "aws:dynamodb",  
      "awsRegion": "eu-west-2",  
      "dynamodb": {  
        "Keys": {  
          "UserId": { "S": "101" }  
        },  
        "OldImage": {  
          "UserId": { "S": "101" },  
          "Status": { "S": "Pending" },  
          "Score": { "N": "80" }  
        },  
        "NewImage": {  
          "UserId": { "S": "101" },  
          "Status": { "S": "Approved" },  
          "Score": { "N": "90" }  
        },  
      }  
    }  
  ]  
}
```

```

    "StreamViewType": "NEW_AND_OLD_IMAGES",
    "SequenceNumber": "111",
    "SizeBytes": 26
  },
  "eventSourceARN":
  "arn:aws:dynamodb:eu-west-2:123456789012:table/MyDynamoDBTable/stream/2025-10-19T10:00:00.000"
}

]
}

```

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar navigation includes CloudWatch, Favorites and recents, Dashboards, Alarms, Logs (Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights), Metrics, and Application Signals (APM). The main content area is titled 'Log events' and displays a table of log entries. The table has two columns: 'Timestamp' and 'Message'. The messages show the Lambda function executing a query on the DynamoDB table and sending an SNS notification. The interface includes a search bar, filter buttons for time (1m, 1h, UTC timezone), and actions like Actions, Start tailing, and Create metric filter.

Timestamp	Message
2025-10-20T09:49:18.969Z]
2025-10-20T09:49:18.969Z	}
2025-10-20T09:49:18.969Z	Sending SNS notification...
2025-10-20T09:49:19.335Z	SNS alert sent successfully.
2025-10-20T09:49:19.351Z	END RequestId: eb632e43-d615-473d-9e4b-3741bb0f5716
2025-10-20T09:49:19.351Z	REPORT RequestId: eb632e43-d615-473d-9e4b-3741bb0f5716 Duration: 391.47 ms Billed Duration: 392 ms Memory...

The screenshot shows the AWS Lambda Function Editor interface. On the left, the EXPLORER panel lists a function named 'DYNAMODBITEMCHANGEALERT' with a single file 'lambda_function.py'. The main area displays the Python code for the lambda function:

```
from datetime import datetime
sns = boto3.client('sns')
# Hardcode SNS topic ARN for local testing (only for test - remove later)
SNS_TOPIC_ARN = os.environ.get("SNS_TOPIC_ARN", "arn:aws:sns:eu-west-2:975050024946:dynamodb-update")
def lambda_handler(event=None, context=None):
    # Sample DynamoDB MODIFY event
    if event is None or 'Records' not in event:
        event = {
            "statusCode": 200,
            "message": "Processed successfully"
        }
    sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Message=event['message'],
        Subject="DynamoDB Item Change Alert"
    )
    return event
```

The 'TEST EVENTS' section shows a selected event named 'dynamo'. The 'OUTPUT' tab in the bottom right shows the test results:

```
Status: Succeeded
Test Event Name: dynamo
Response:
{
  "statusCode": 200,
  "message": "Processed successfully"
```

The 'Execution Results' tab shows the log output:

```
2025-01-17T10:45:45.000Z INFO [LambdaFunction] Starting execution
2025-01-17T10:45:45.000Z INFO [LambdaFunction] Processing event: dynamo
2025-01-17T10:45:45.000Z INFO [LambdaFunction] Publishing message to SNS topic: arn:aws:sns:eu-west-2:975050024946:dynamodb-update
2025-01-17T10:45:45.000Z INFO [LambdaFunction] Message published successfully
2025-01-17T10:45:45.000Z INFO [LambdaFunction] Returning response: {"statusCode": 200, "message": "Processed successfully"}
```

This screenshot is similar to the first one, but the 'Execution Results' tab shows the log output for the test execution, which includes the SNS notification being sent:

```
Sending SNS notification...
SNS alert sent successfully.
END RequestId: d6e9c3f2-e87a-459d-9fdf-06c6313728ff
REPORT RequestId: d6e9c3f2-e87a-459d-9fdf-06c6313728ff Duration: 304.15 ms Billed Duration: 852 ms Memory Size: 128 MB Max Memory Used: 84 MB Init Duration: 547.12 ms
```

Testing code by lambda function

```
import json

import boto3

import os

from datetime import datetime

sns = boto3.client('sns')
```

```

# Hardcode SNS topic ARN for local testing (only for test - remove later)

SNS_TOPIC_ARN = os.environ.get("SNS_TOPIC_ARN",
"arn:aws:sns:eu-west-2:975050024946:dynamodb-update-alerts-adish")



def lambda_handler(event=None, context=None):

    # Sample DynamoDB MODIFY event

    if event is None or 'Records' not in event:

        event = {

            "Records": [
                {
                    "eventID": "1",
                    "eventName": "MODIFY",
                    "eventVersion": "1.1",
                    "eventSource": "aws:dynamodb",
                    "awsRegion": "eu-west-2",
                    "dynamodb": {
                        "Keys": { "UserId": { "S": "101"} },
                        "OldImage": {
                            "UserId": { "S": "101" },
                            "Status": { "S": "Pending" },
                            "Score": { "N": "80" }
                        },
                        "NewImage": {

```

```

        "UserId": {"S": "101"},

        "Status": {"S": "Approved"},

        "Score": {"N": "90"}


    } ,


    "StreamViewType": "NEW_AND_OLD_IMAGES",
    "SequenceNumber": "111",
    "SizeBytes": 26


},


"eventSourceARN":


"arn:aws:dynamodb:eu-west-2:123456789012:table/MyDynamoDBTable/stream/2025
-10-19T10:00:00.000"
}

]

}

print("Received event:", json.dumps(event, indent=2))

for record in event['Records']:

    event_name = record['eventName'] # INSERT, MODIFY, REMOVE

    table_name = record['eventSourceARN'].split('/')[-1]

    # Only handle updates (MODIFY)

    if event_name == 'MODIFY':

        old_image = record['dynamodb'].get('OldImage', {})


```

```

new_image = record['dynamodb'].get('NewImage', {})

# Convert DynamoDB JSON to readable format

old_item = {k: list(v.values())[0] for k, v in
old_image.items()}

new_item = {k: list(v.values())[0] for k, v in
new_image.items()}

changed_fields = []

for key in new_item.keys():

    old_val = old_item.get(key)

    new_val = new_item.get(key)

    if old_val != new_val:

        changed_fields.append(f'{key}: '{old_val}' →
'{new_val}'')

# Construct message

message = (

    f"🔔 *DynamoDB Item Updated Alert*\n"

    f"Table: {table_name}\n"

    f"Time: {datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S
UTC')}\n\n"

    f"Changes:\n" + "\n".join(changed_fields)

)

```

```
    print("Sending SNS notification...")

    sns.publish (
        TopicArn=SNS_TOPIC_ARN,
        Subject=f"🔔 DynamoDB Update in {table_name}",
        Message=message
    )

    print("SNS alert sent successfully.")

else:
    print(f"Ignored event type: {event_name}")

return {"statusCode": 200, "message": "Processed successfully"}
```



```
# For local test execution

if __name__ == "__main__":
    lambda_handler()
```