

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 Management Console 'Create table' page for DynamoDB. The breadcrumb navigation at the top reads 'DynamoDB > Tables > Create table'. The main heading is 'Create table'. Below this, there's a 'Table details' section with an 'Info' icon. A descriptive text states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The 'Table name' field is labeled 'EmployeeDataAdish' with a note: 'This will be used to identify your table.' Below the field, a validation message says: 'Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).' The 'Partition key' section is labeled 'employeeId' with a dropdown menu set to 'String'. A note explains: 'The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.' Below this, a validation message says: '1 to 255 characters and case sensitive.' The 'Sort key - optional' section has a placeholder 'Enter the sort key name' and a dropdown menu set to 'String'. A note explains: 'You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.' Below this, a validation message says: '1 to 255 characters and case sensitive.' The footer of the console shows 'CloudShell', 'Feedback', '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

Create 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](#)

Attributes

Attribute name	Value	Type	Remove
employeeid - Partition key	<pre>{ "employeeid": "E101", "name": "John Doe", "role": "Developer", "salary": 60000 }</pre>	String	

[Add new attribute](#) [Cancel](#) [Create item](#)

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

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

dynamodb-update-alerts-adish

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.

My Topic

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

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

Protocol

The type of endpoint to subscribe

Email

Endpoint

An email address that can receive notifications from Amazon SNS.

Adishansari786@gmail.com

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



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

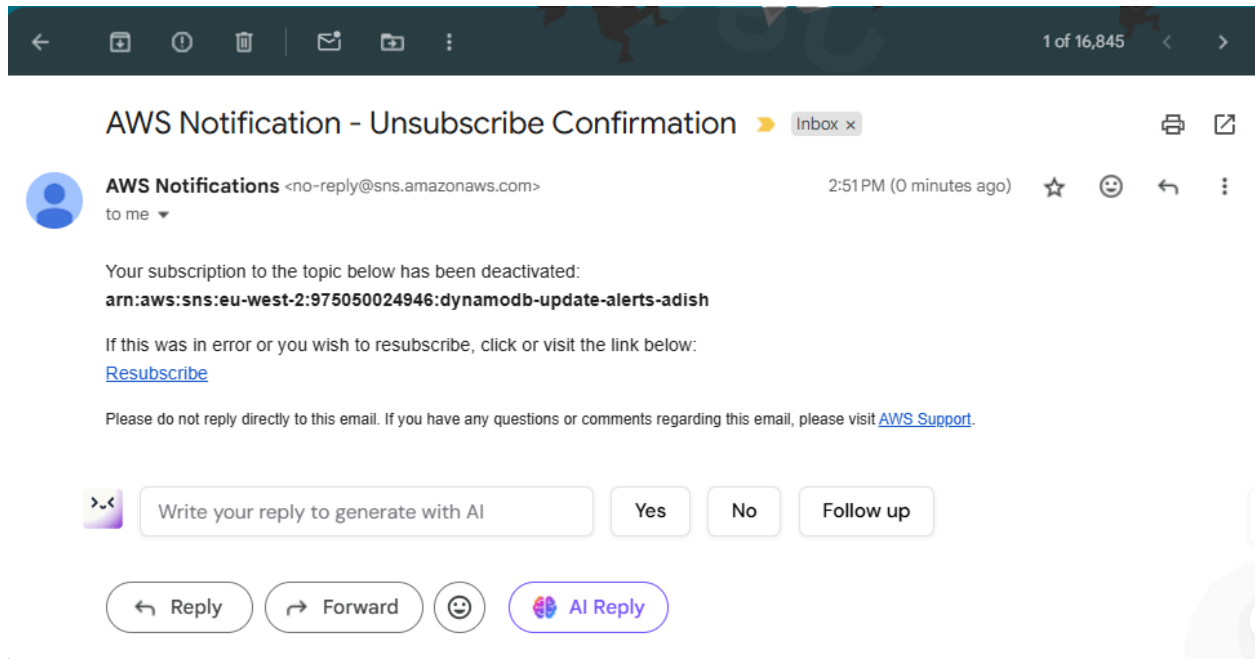
Subscription confirmed!

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' page in the AWS Lambda console. The breadcrumb navigation at the top reads 'Lambda > Functions > Create function'. The main heading is 'Create function' with an 'Info' link. Below this, a subtext says 'Choose one of the following options to create your function.' There are three radio button options: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. The 'Basic information' section contains a 'Function name' field with the value 'DynamoDBItemChangeAlert', a 'Runtime' dropdown set to 'Python 3.13', and an 'Architecture' dropdown. The footer includes 'CloudShell', 'Feedback', and copyright information for Amazon Web Services, Inc.

Create function [Info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**
Start with a simple Hello World example.

☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

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

The screenshot shows the 'Permissions' page in the AWS Lambda console. The breadcrumb navigation at the top reads 'Lambda > Functions > Create function'. The main heading is 'Permissions' with an 'Info' link. Below this, a subtext says 'By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.' There are two sections: 'Change default execution role' and 'Existing role'. The 'Change default execution role' section has three radio button options: 'Create a new role with basic Lambda permissions', 'Use an existing role' (selected), and 'Create a new role from AWS policy templates'. The 'Existing role' section has a dropdown menu with the value 'aws-lambda-full-access'. The footer includes 'CloudShell', 'Feedback', and copyright information for Amazon Web Services, Inc.

Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions

☒ Use an existing role

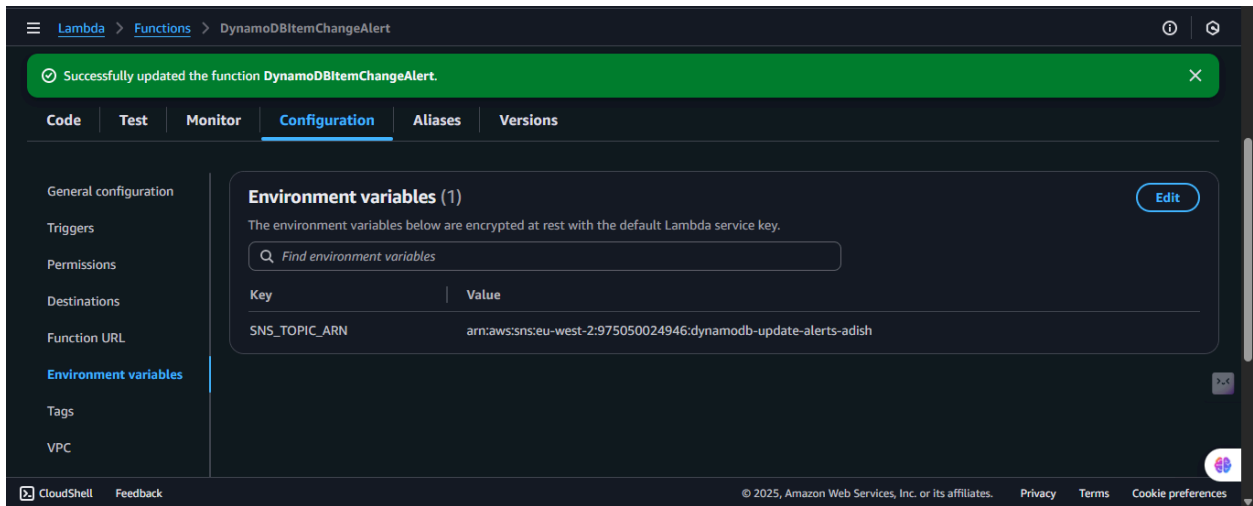
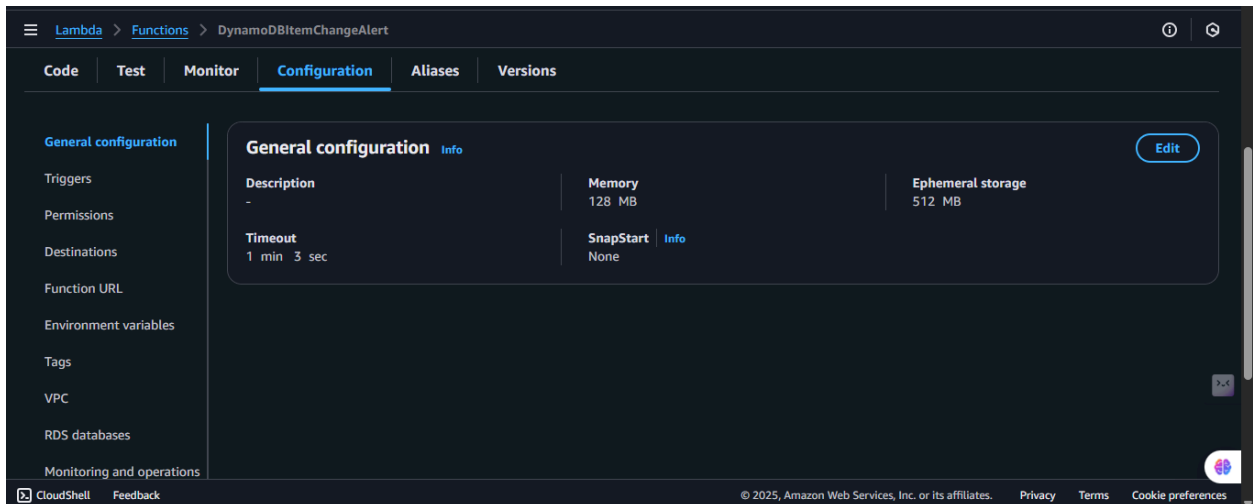
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

[View the aws-lambda-full-access role](#) on the IAM console.

Additional configurations
Use additional configurations to set up networking, security, and governance for your function. These settings help secure and customize your Lambda function deployment.

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



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

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

Tables (2)

Any tag key

Any tag value

Find tables

1

bharathi-table

EmployeeDataAdish

EmployeeDataAdish

Monitor

Global tables

Backups

Exports and streams

Permissions

Exports to S3

View details

Export to S3

Showing all export jobs from the last 90 days.

Find exports

0 matches

Initiated by = User

Clear filters

Export ARN

Destination S3 bucket

Status

Export job start time (UTC+08)

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

DynamoDB stream details

Turn on

Stream status

Off

Trigger

Amazon Kinesis data stream details

Turn on

Amazon Kinesis Data Streams for DynamoDB captures item-level changes in your table, and replicates the changes to a Kinesis data stream. You then can consume and manage the change information from Kinesis. [Learn more](#)

Status

Off

DynamoDB

Tables

EmployeeDataAdish

Turn on DynamoDB stream

Turn on DynamoDB stream

DynamoDB stream details

Capture item-level changes in your table, and push the changes to a DynamoDB stream. You then can access the change information through the DynamoDB Streams API.

View type

Choose which versions of the changed items you would like to push to the DynamoDB stream.

New and old images

Both the new and old images of the changed item.

Key attributes only

Only the key attributes of the changed item.

New image

The entire item as it appears after it was changed.

Old image

The entire item as it appears before it was changed.

Cancel

Turn on stream

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

Turn off

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

Trigger (0)

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

CloudShell

Feedback

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

Cookie preferences

Lambda

Functions

DynamoDBItemChangeAlert

Code

Test

Monitor

Configuration

Aliases

Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

Triggers (0)

Info

Fix errors

Edit

Delete

Add trigger

Find triggers

Trigger

No triggers

No triggers are configured.

Add trigger

CloudShell

Feedback

© 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

CloudShell

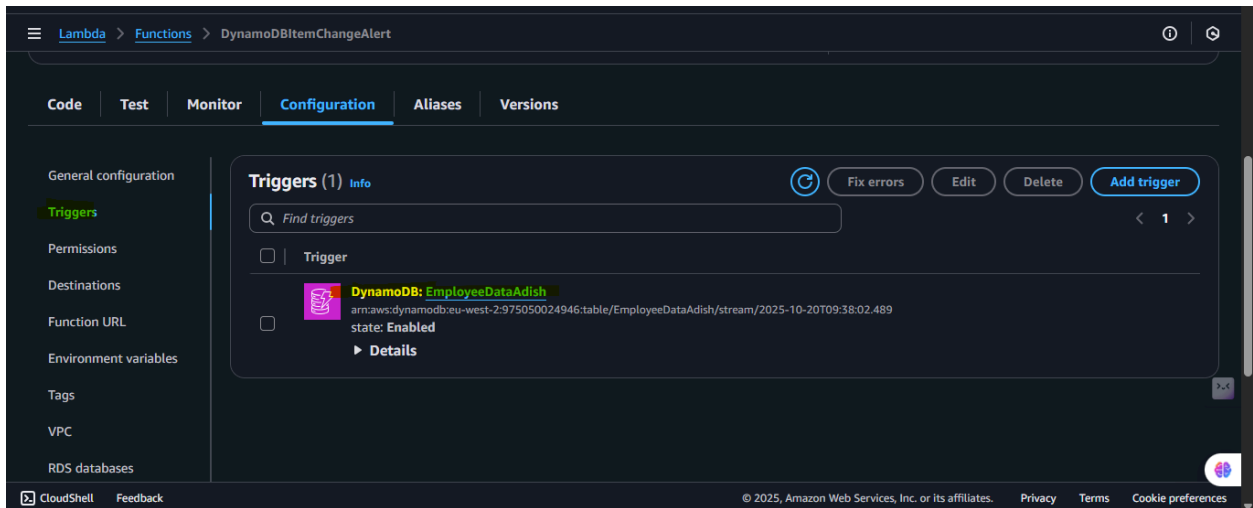
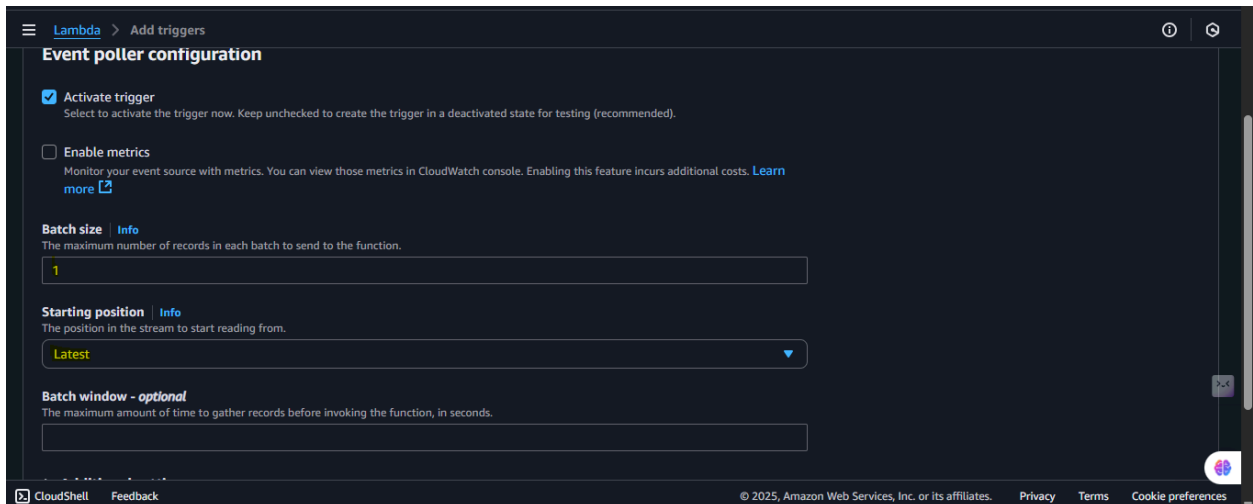
Feedback

© 2025, Amazon Web Services, Inc. or its affiliates.

Privacy

Terms

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

FormJSON view

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](#)

Attributes

Add new attribute

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

Cancel

Save

Save and close

CloudShellFeedback

© 2025, Amazon Web Services, Inc. or its affiliates. PrivacyTermsCookie preferences

Lambda > Functions > DynamoDBItemChangeAlert

Code

Test

Monitor

Configuration

Aliases

Versions

Executing function: succeeded (logs)

Details

Test event

Info

CloudWatch Logs Live Tail

Save

Test

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

Asynchronous

CloudShellFeedback

© 2025, Amazon Web Services, Inc. or its affiliates. PrivacyTermsCookie preferences

Lambda > Functions > DynamoDBItemChangeAlert

Private

Shareable

Template - optional

dynamo

Event JSON

Format 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 *          "FirstName": { "S": "JohnDoe" }
16 *        }
17 *      }
18 *    }
19 *  ]
20 * }
```

CloudShellFeedback

© 2025, Amazon Web Services, Inc. or its affiliates. PrivacyTermsCookie 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"

}

]

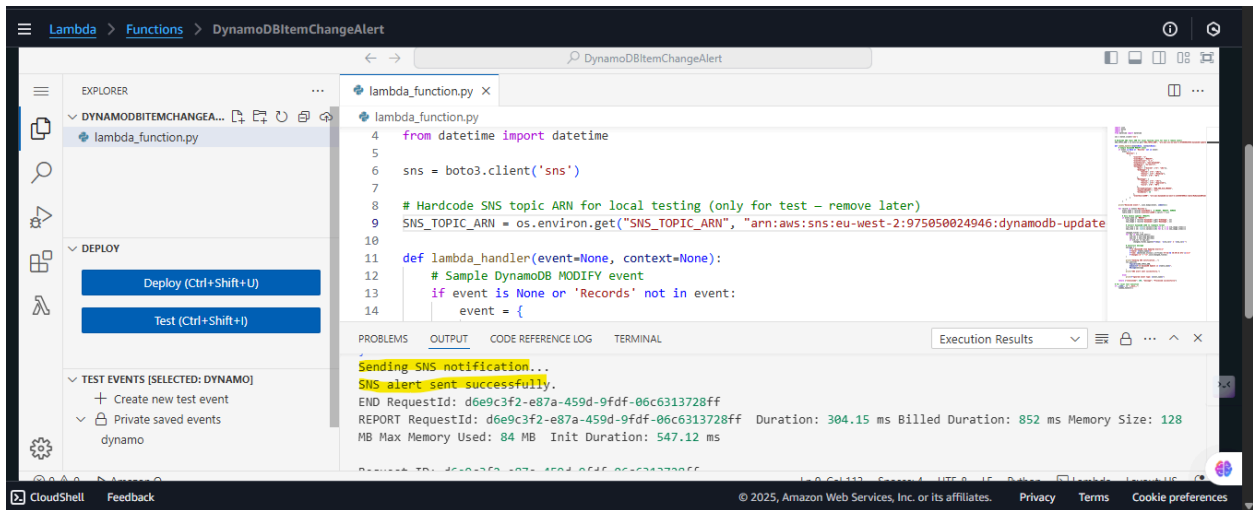
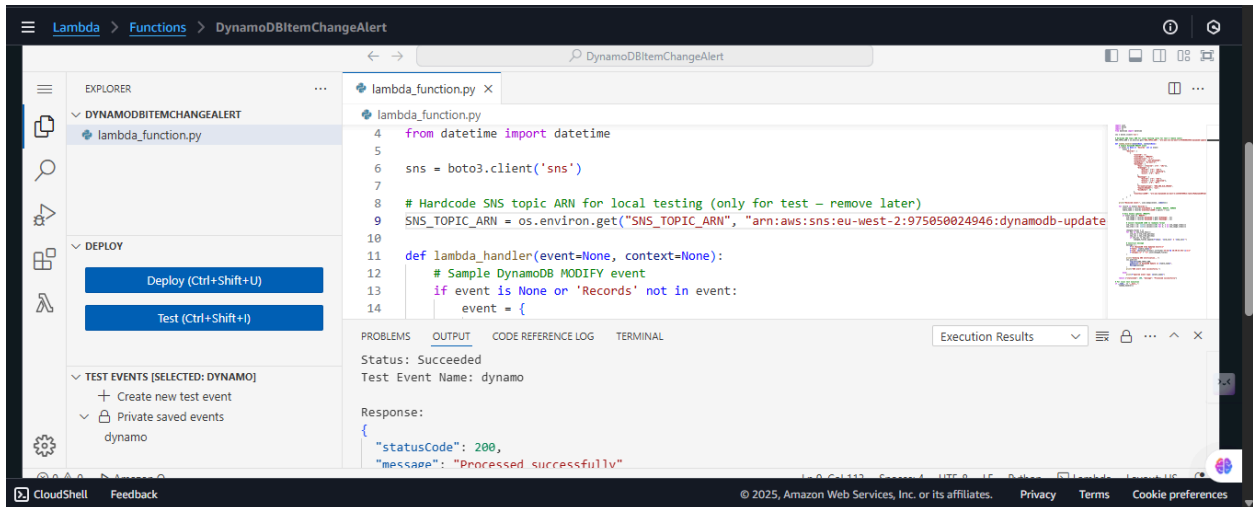
}

```

The screenshot shows the AWS CloudWatch console interface. The breadcrumb navigation at the top indicates the path: CloudWatch > Log groups > /aws/lambda/DynamoDBItemChangeAlert > 2025/10/20/[\$LATEST]48739e0875f840b09ae8c93de02417d3. The left sidebar contains navigation options: CloudWatch, Favorites and recents, Dashboards, Alarms, Logs (selected), Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights, Metrics, and Application Signals (APM). The main panel is titled 'Log events' and includes a search bar, a 'Filter events - press enter to search' prompt, and buttons for 'Display', 'Actions', 'Start tailing', and 'Create metric filter'. Below the search bar, a table displays log events with columns for 'Timestamp' and 'Message'. The events are as follows:

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...

At the bottom of the log events list, a message states: 'No newer events at this moment. Auto retry paused. Resume'. A 'Back to top' button is also visible. The footer of the console shows the copyright notice: © 2025, Amazon Web Services, Inc. or its affiliates. and links for Privacy, Terms, and Cookie preferences.



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()
```