Amazon DynamoDB Streams and TTL

**SPL-TF-200-DBDYL4-10-EN - Version 1.0.5**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab Overview**

AnyCompany is developing a new movie database application. The application will allow users to search for movies by title, year they were released, actors in the movie, rating, and so on. AnyCompany chose Amazon DynamoDB as the database service to store the information about all of the movies that the application will reference.

Amazon DynamoDB is a serverless NoSQL, also known as non-relational, database that supports key-value and document data models. With DynamoDB, you can start small and quickly scale globally as your application and user base grows, with the ability to support petabytes of data and tens of millions of read and write requests per second. AWS manages the DynamoDB service, so there are no servers to update or maintain. For more information, refer to *Amazon DynamoDB Features* and *Core Components of Amazon DynamoDB* in the **Additional resources** section.

As a newly hired junior application developer at AnyCompany, you have been tasked with developing an event driven workflow to notify customers when new movies have been added to, or removed from, your database. Additionally, your company would like a method for recording expired database items to S3 for long term storage. To achieve this you have chosen a solution combining Amazon DynamoDB Streams, Amazon DynamoDB Time to Live (TTL), and AWS Lambda.

TOPICS COVERED

By the end of this lab, you will be able to:

* Use the AWS CLI and shell variables to enable DynamoDB Streams.
* Connect a Lambda function built with the AWS SDK for Python (Boto3) to a DynamoDB Stream.
* Send email notifications to a user whenever an item is added to, modified, or removed from a DynamoDB table.
* Automate DynamoDB item removal by enabling a Time To Live (TTL) attribute for a table item.
* Create a workflow to move expired DynamoDB items to long term storage in Amazon S3.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with using a linux console, NoSQL database concepts, basic bash scripting and be comfortable editing files using a text editor.

ICON KEY

Various icons are used throughout this lab to call attention to certain aspects of the guide. The following list explains the purpose for each one:

* The keyboard icon specifies that you must run a command.
* The clipboard icon indicates that you can verify the output of a command or edited file by comparing it to the provided example.
* The note icon specifies important hints, tips, guidance, or advice.
* The “i” circle icon specifies where to find more information.
* The person with a check mark icon indicates an opportunity to check your knowledge and test what you have learned.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Connect to AWS Cloud9**

In this task, you will connect to the AWS Cloud9 integrated development environment (IDE) and verify it has the tools necessary to interact with DynamoDB.

TASK 1.1: CONNECT TO THE AWS CLOUD9 IDE

1. At the top of the page, in the unified search bar, search for and choose

Cloud9

.

1. In the Environments page, select  **Lab\_Cloud9\_IDE**

* Then choose **Open in Cloud9**

The AWS Cloud9 environment opens in a new browser tab.

1. Select the “+” icon at the top of the upper pane of the Cloud9 IDE and then select “New Terminal”.

The Cloud9 IDE contains an upper pane for file editing, and lower pane for active terminals. Selecting the “+” icon allows you to enter shell commands along side the tabs for file editing. The majority of steps in this course are performed in the terminal command-line shell.

**Note**: The steps of this lab employ the use of shell variables. If you close your terminal window tab, or move to another terminal, the values you store in those variables will be lost and you will need to repeat the steps that store values in those variables for any functions that depend on them.

TASK 1.2: VERIFY THAT THE AWS CLI IS INSTALLED IN THE CLOUD9 ENVIRONMENT

Cloud9 development environments come prepackaged with the AWS CLI and tooling for over 40 programming languages. In this task you will verify that the AWS CLI is installed.

1. To verify the AWS CLI is installed and to display its version, run the following command in a terminal window:

aws --version

 The output should show aws-cli 1.x.x, similar to the following:

aws-cli/2.15.28 Python/3.11.8 Linux/5.10.210-201.852.amzn2.x86\_64 exe/x86\_64.amzn.2 prompt/off

**Task 2: Enable DynamoDB Streams**

In this task you will list the DynamoDB tables in your account and view details of the

movies

 table. You will then enable DynamoDB Streams and verify the Stream details.

TASK 2.1: VIEW TABLE DETAILS

1. Use the AWS CLI to view all tables in your account.

aws dynamodb list-tables

 The output lists all the tables in this account. You will be using the movies table for the remainder of this lab.

{

"TableNames": [

"movies"

]

}

1. Now view the movies table details.

aws dynamodb describe-table --table-name movies

 The output should give the full details of the movies table, similar to the following:

{

"Table": {

"AttributeDefinitions": [

{

"AttributeName": "title",

"AttributeType": "S"

},

{

"AttributeName": "year",

"AttributeType": "N"

}

],

"TableName": "movies",

"KeySchema": [

{

"AttributeName": "year",

"KeyType": "HASH"

},

{

"AttributeName": "title",

"KeyType": "RANGE"

}

],

"TableStatus": "ACTIVE",

"CreationDateTime": "2021-01-01T01:01:01.100000+00:00",

"ProvisionedThroughput": {

"NumberOfDecreasesToday": 0,

"ReadCapacityUnits": 10,

"WriteCapacityUnits": 10

},

"TableSizeBytes": 0,

"ItemCount": 0,

"TableArn": "arn:aws:dynamodb:XX:XX:table/movies",

"TableId": "abababab-9999-9999-9999-abababababab"

}

}

 Notice that there are no stream details in the output.

TASK 2.2: ENABLE AND VERIFY DYNAMODB STREAMS

DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table and stores this information in a log for up to 24 hours. Applications can access this log and view the data items as they appeared before and after they were modified, in near-real time.

A DynamoDB stream is an ordered flow of information about changes to items in a DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table.

Whenever an application creates, updates, or deletes items in the table, DynamoDB Streams writes a stream record with the primary key attributes of the items that were modified. A stream record contains information about a data modification to a single item in a DynamoDB table. You can configure the stream so that the stream records capture additional information, such as the “before” and “after” images of modified items.

Before you enable streams, you first need to decide what data to put into the stream when a change is made to your table. The options are:

* KEYS\_ONLY

 Only the key attributes of the modified item are written to the stream.

* NEW\_IMAGE

 The entire item, as it appears after it was modified, is written to the stream.

* OLD\_IMAGE

 The entire item, as it appeared before it was modified, is written to the stream.

* NEW\_AND\_OLD\_IMAGES

 Both the new and the old item images of the item are written to the stream.

For this lab, you will set StreamViewType to

NEW\_AND\_OLD\_IMAGES

.

1. Enable DynamoDB streams for the movies table. Enter the following in to your terminal.

aws dynamodb update-table \

--table-name movies \

--stream-specification '{"StreamEnabled": true, "StreamViewType": "NEW\_AND\_OLD\_IMAGES"}'

1. Check the command output is similar to that shown below, noting the StreamViewType:

{

"TableDescription": {

"AttributeDefinitions": [

{

"AttributeName": "title",

"AttributeType": "S"

},

{

"AttributeName": "year",

"AttributeType": "N"

}

],

"TableName": "movies",

"KeySchema": [

{

"AttributeName": "year",

"KeyType": "HASH"

},

{

"AttributeName": "title",

"KeyType": "RANGE"

}

],

"TableStatus": "UPDATING",

"CreationDateTime": "2021-01-01T01:01:01.100000+00:00",

"ProvisionedThroughput": {

"NumberOfDecreasesToday": 0,

"ReadCapacityUnits": 10,

"WriteCapacityUnits": 10

},

"TableSizeBytes": 0,

"ItemCount": 0,

"TableArn": "arn:aws:dynamodb:XX:XX:table/movies",

"TableId": "abababab-9999-9999-9999-abababababab",

"StreamSpecification": {

"StreamEnabled": true,

"StreamViewType": "NEW\_AND\_OLD\_IMAGES"

},

"LatestStreamLabel": "2021-01-01T01:10:01.000",

"LatestStreamArn": "arn:aws:dynamodb:XX:XX:table/movies/stream/2021-01-01T01:10:01.000"

}

}

TASK 2.3: VIEW STREAM STATUS

In this task you will use the CLI to verify the stream status and view its configuration.

1. Use the AWS CLI to check a table’s stream status. Enter the following into your terminal:

aws dynamodb describe-table --table-name movies \

--query 'Table.StreamSpecification'

 Notice the use of

--query 'Table.StreamSpecification'

 to display only the required details.

 See the **Additional Resources** section for more details about using the

--query

 command line switch.

1. Check the command output is similar that shown below:

{

"StreamEnabled": true,

"StreamViewType": "NEW\_AND\_OLD\_IMAGES"

}

 You can also view details about a specific stream by using the

aws dynamodbstreams

 command.

aws dynamodbstreams

 returns information about a stream, including the current status of the stream, its Amazon Resource Name (ARN), the composition of its shards, and its corresponding DynamoDB table.

 See the **Additional Resources** section for more details.

1. Populate the

${STREAMARN}

 shell variable. You will use shell variables throughout this lab. To create a shell variable containing your stream’s Amazon Resource Name (ARN), enter the following into your terminal.

STREAMARN=$(aws --output text dynamodb describe-table --table-name movies --query 'Table.LatestStreamArn')

1. You can now use this new shell variable in functions that require your stream’s ARN. Enter the following into your terminal to view information about your stream.

aws dynamodbstreams describe-stream --stream-arn ${STREAMARN}

1. Check the command output. Specifically you are looking for

"StreamStatus": "ENABLED"

{

"StreamDescription": {

"StreamArn": "arn:aws:dynamodb:XX:XX:table/movies/stream/2021-01-01T01:10:01.000",

"StreamLabel": "2021-01-01T01:10:01.000",

"StreamStatus": "ENABLED",

"StreamViewType": "NEW\_AND\_OLD\_IMAGES",

"CreationRequestDateTime": 1612817053.081,

"TableName": "movies",

"KeySchema": [

{

"AttributeName": "year",

"KeyType": "HASH"

},

{

"AttributeName": "title",

"KeyType": "RANGE"

}

],

"Shards": [

{

"ShardId": "shardId-00000001612817053791-d8eff023",

"SequenceNumberRange": {

"StartingSequenceNumber": "673600000000062203542320"

}

}

]

}

}

You have now successfully enabled DynamoDB Streams and configured it to deliver

OLD\_AND\_NEW\_IMAGES

**Task 3: Connect an AWS Lambda Function to DynamoDB Streams**

In this task you will view the details of an existing AWS Lambda function and connect it to the DynamoDB Stream you created in the previous task.

TASK 3.1: LIST THE AWS LAMBDA FUNCTIONS AVAILABLE IN YOUR ACCOUNT

1. Enter the following into your terminal to list the AWS Lambda functions available along with their details.

aws lambda list-functions

1. Take moment to read through the returned output. One of the functions is similar to the sample below. Note the permission “Role” assigned to the Lambda function. This is necessary for our function to interact with the DynamoDB service.

**Note** You can press

q

 to exit from the output stream.

{

"Functions": [

{

"FunctionName": "XX-LambdaStreamConsumer-XX",

"FunctionArn": "arn:aws:lambda:XX:XX:function:XX-LambdaStreamConsumer-XX",

"Runtime": "python3.8",

"Role": "arn:aws:iam::XX:role/XX-LambdaStreamConsumerRole-XX",

"Handler": "index.handler",

"CodeSize": 326,

"Description": "",

"Timeout": 3,

"MemorySize": 128,

"LastModified": "2021-02-08T19:33:44.371+0000",

"CodeSha256": "Xd/qjb9HX6PQvxrm1Rm1fzVv9sbXolH0u50oBKleQXM=",

"Version": "$LATEST",

"Environment": {

"Variables": {

"SNSTOPIC": "arn:aws:sns:XX:XX:XX-SNSTopic-XX"

}

},

"TracingConfig": {

"Mode": "PassThrough"

},

"RevisionId": "42591a7b-d26a-4309-b773-e52702893e8a",

"PackageType": "Zip"

}

]

}

TASK 3.2 VIEW THE AWS LAMBDA FUNCTION CODE

In this task you will retrieve the function from the Lambda service and view its code.

1. Populate the

${FNAME}

 shell variable.

* In the top left section of these instructions page, copy the value of **LambdaFunctionARN**
* Replace

<LambdaFunctionARN>

 below with the value you just copied

FNAME=<LambdaFunctionARN>

1. Use the shell variable to retrieve your function’s location by entering the following into your terminal.

aws lambda get-function --function-name ${FNAME}

 Check the command output is similar to below. Specifically you are looking for the

Location

 element, this is where your function code is stored.

{

"Configuration": {

"FunctionName": "XX-LambdaStreamConsumer-XX",

"FunctionArn": "arn:aws:lambda:XX-XX-X:XX:function:XX-LambdaStreamConsumer-XX",

"Runtime": "python3.8",

"Role": "arn:aws:iam::XX:role/XX-LambdaStreamConsumerRole-XX",

"Handler": "index.handler",

"CodeSize": 685,

"Description": "",

"Timeout": 3,

"MemorySize": 128,

"LastModified": "2021-02-16T22:05:49.574+0000",

"CodeSha256": "DgL0mlw9XwOXN8FHwqOBIUGWWvrYa9X2mrk3vU8DgcQ=",

"Version": "$LATEST",

"Environment": {

"Variables": {

"SNSTOPIC": "arn:aws:sns:XX-XX-X:XX:XX-SNSTopic-XX"

}

},

"TracingConfig": {

"Mode": "PassThrough"

},

"RevisionId": "8a509116-b7b1-402f-8b41-6430735e5a3b",

"State": "Active",

"LastUpdateStatus": "Successful",

"PackageType": "Zip"

},

"Code": {

"RepositoryType": "S3",

"Location": "https://prod-04-2014-tasks.s3.XX-XX-X.amazonaws.com/snapshots/XX/XX-LambdaStreamConsumer-XX?versionId=bmHJ3LoLtsaSaw8jrRHgV4Pb6rjU9bhG&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEEUaCXVzLWVhc3QtMSJHMEUCIAx4hdSFApZ5PZ3w8SdfnUWnvGLzkxYlfoT%2FxFAjHQwnAiEA8cxfu8MrBvMrx%2B4DIcefPaCf7xzCOe7QeckLKljOeFcqtAMITRAAGgw3NDk2Nzg5MDI4MzkiDJUDgVDtg88LZnOgkCqRA89ncXDWIwCJ9bPQQVAnuSsJKSxY1D%2BonRX79P7J2EcnduaqpEak%2BVrxBsKgFk29rxjZdQ0lG3tx4K6C3AJrcH28bIYe7myQZj02tn4nI9UYB9lR9qm8tVXQTxiBc8jq6xK3ntcXkSm05RVU2tIZ5hGVhl68yYb%2BpOrZuQeQ%2Bp%2FJN%2FX%2FUBNd2%2BNhBZ5fjgZa%2BTgSk0byFkeBPFc5i54HaRAyfBkuMrIMoaE5WbcMVYju2B%2FWmKSE7N2GkrybBFiOALG%2BEo0PXwh1TKHuSUN8UWYnx7FY7o0o8CKmI%2FR1JJ750hTrbcH99KtYbh0Mif8L30x37Jo%2Fd2gkazGyCMnHM4eH0%2FeZ6rnw1yagk4Z%2FaRn6nmsXRBOKwPjmhPnzMVmDJVduNnMPnrluhPptlG%2Bid5QTFHn69YPdDBLU40jUkrz5FVADO2OeMKkhcIG9Pym1ueiTf7qEJ7YIHqVAo%2Bn4e0VVyHv%2BzfINx4RbxnHRCIxbYsPkowdSOdnP3aE6zHYAq84JwdBKJGO5kSqxFDBwXw6SMIvbsIEGOusBipXi18w2El6JuHmfllpe3dNGSoHIs7xcM0eMdm434D6r%2By9r5fF12wU4vcdnGlxwIx9ZwLdMU8ZtXq%2BVJ5jiWien%2FT0jZlmGWhGhvkECgi%2F%2BFNL7X7b0UZHKkbG2LMgbAGBUPH709sgPbNW5HLJNMUhzhtr6slZ3WCLr2mdt4Xl3mm0kD%2B%2FbUPK%2BpaftigIcDC5W1jPOQhwuZWN16o9yXLooDXJw4Hgd4eFYATv3G02863CZdHRcxdvSFE04AG0MMtpuBQWD2VokGaY6GuvdDLDIhKoesnUyNrsvdAdmf6Yqu1DQfAtJfyXmoQ%3D%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210216T221858Z&X-Amz-SignedHeaders=host&X-Amz-Expires=599&X-Amz-Credential=ASIA25DCYHY34EBQMJBV%2F20210216%2Fus-east-1%2Fs3%2Faws4\_request&X-Amz-Signature=018e17b607286ee040926b4f6600be167df085c0f67ee79aed3e4c8c3ab285fb"

},

"Tags": {

"aws:cloudformation:stack-name": "XX",

"aws:cloudformation:stack-id": "arn:aws:cloudformation:XX-XX-X:XX:stack/XX/XX-XX-XX-XX-XX",

"aws:cloudformation:logical-id": "LambdaStreamConsumer"

}

}

1. Populate the shell variable FLOC with the value from

Code.Location

.

FLOC=$(aws --output text lambda get-function --function-name ${FNAME} --query 'Code.Location')

1. Download the function code to a local file with the following command.

wget "${FLOC}" -O function.zip

1. Unzip

function.zip

. so you can access your code.

unzip function.zip

1. Look to the top left of your Cloud9 IDE to see its file explorer. There you should now see a file named ‘index.py’. Double-click

index.py

 in the file explorer to open your Python function in a new Cloud9 window. Keep this file open, you will modify it in a later task.

Take a moment to read through the code. This function is a simple event router written in Python and employing the AWS SDK for Python (Boto3). It looks for a field in the stream event to detect what data change action was performed and sets an appropriate email subject.

TASK 3.3 CONNECT THE AWS LAMBDA FUNCTION TO THE STREAM

In this task you will connect the Lambda function to the stream you created.

1. Return to your terminal window and connect the AWS Lambda function to the stream you created previously with the following command.

aws lambda create-event-source-mapping \

--function-name ${FNAME} \

--batch-size 1 \

--starting-position LATEST \

--event-source ${STREAMARN}

1. Check the command output is similar to the following. Note that the command maps an event source, specified in the EventSourceArn, as a trigger to your Lambda function.

{

"UUID": "deca28ac-f828-4164-8d3f-066ecfd4a51f",

"StartingPosition": "LATEST",

"BatchSize": 1,

"MaximumBatchingWindowInSeconds": 0,

"ParallelizationFactor": 1,

"EventSourceArn": "arn:aws:dynamodb:XX:XX:table/movies/stream/2021-01-01T01:10:01.000",

"FunctionArn": "arn:aws:lambda:XX:XX:function:XX-LambdaStreamConsumer-XX",

"LastModified": 1612987562.915,

"LastProcessingResult": "No records processed",

"State": "Creating",

"StateTransitionReason": "User action",

"DestinationConfig": {

"OnFailure": {}

},

"MaximumRecordAgeInSeconds": -1,

"BisectBatchOnFunctionError": false,

"MaximumRetryAttempts": -1,

"TumblingWindowInSeconds": 0,

"FunctionResponseTypes": []

}

You now have an AWS Lambda function connected to the DynamoDB Stream. Any data written to the movies table will now cause the function to execute.

Lambda functions read records from a stream and invoke your function synchronously. A Lambda function can process records individually, as is the case here, or in batches to minimize costs. Here we have configured a batch size of 1 with

--batch-size 1

, a more economical value could be

--batch-size 100

If your function returns an error, Lambda retries the batch until processing succeeds or the data expires. To retain discarded events, you can configure the event source mapping to send details about failed batches to an SQS queue or SNS topic.

 See **Additional Resources** for further details.

**Task 4: Subscribe to an SNS topic**

In this task you will subscribe your email address to an existing Simple Notification Service (SNS)topic. This subscription will act as the destination for DynamoDB Stream events. You will need access to your email inbox to confirm the subscription.

The Lambda function you managed in the previous task is already configured to send stream events to the SNS topic, you are adding a destination to the topic.

TASK 4.1: VIEW SNS TOPICS

1. In this task you will view SNS topics in your account.

aws sns list-topics

1. Check the command output is similar to below:

{

"Topics": [

{

"TopicArn": "arn:aws:sns:XX:XX:XX-SNSTopic-XX"

}

]

}

1. Populate the

${TOPICARN}

 shell variable.

TOPICARN=$(aws --output text sns list-topics --query 'Topics[0].TopicArn')

1. List existing subscriptions.

aws sns list-subscriptions-by-topic --topic-arn ${TOPICARN}

1. Check the command output is similar to below, there should be no existing subscriptions to the topic.

{

"Subscriptions": []

}

TASK 4.2: CREATE SUBSCRIPTION

In this task you will subscribe your email address to the SNS topic.

1. Subscribe your email address to your SNS topic by entering the command below into a terminal.

**Be sure to replace**

**<YOUR\_EMAIL\_ADDRESS>**

**with an email address that you have access to.**

 It is always recommended to edit the command in text editor before running.

aws sns subscribe --topic-arn ${TOPICARN} \

--protocol email \

--notification-endpoint <YOUR\_EMAIL\_ADDRESS>

1. Check the command output is similar to below:

{

"SubscriptionArn": "pending confirmation"

}

Notice

"SubscriptionArn": "pending confirmation"

, it will stay in that status until you have confirmed the subscription.

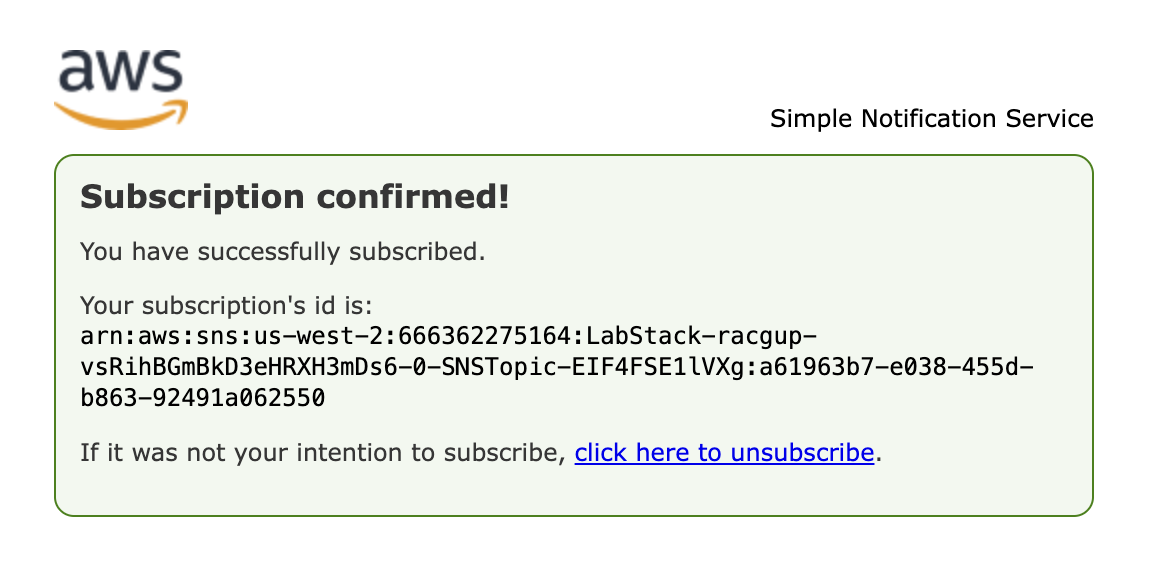
1. Go to the email inbox for the address specified above. You should find a subscription request from

no-reply@sns.amazonaws.com

. Click **Confirm subscription**.

 It may take a minute for the email to show up in inbox.

1. Once you have clicked **Confirm subscription**, you should see **Subscription confirmed** message as below.



1. Return to your terminal window in Cloud9, and verify the subscription by entering the following command:

aws sns list-subscriptions-by-topic --topic-arn ${TOPICARN}

1. Check the command output is similar to the below:

{

"Subscriptions": [

{

"SubscriptionArn": "arn:aws:sns:XX:XX:XX-SNSTopic-XX:66f539ef-d456-4a86-b6a5-4415f6e31dac",

"Owner": "042354412211",

"Protocol": "email",

"Endpoint": "<YOUR\_EMAIL\_ADDRESS>",

"TopicArn": "arn:aws:sns:XX:XX:XX-SNSTopic-XX"

}

]

}

Congratulations! You have created a complete pipeline for capturing and responding to changing data in a DynamoDB table. Data modifications to the movies table are written as individual events into the stream, AWS Lambda pulls events from the stream and executes your Lambda function. The function processes the event and sends it to the SNS Topic, SNS then delivers the data to your email address.

TASK 4.3: ADD AN ITEM TO THE TABLE

We will now demonstrate our data pipeline by adding a new movie record to the table and confirming an email notification.

1. Create a new Cloud9 file selecting “File” at the top left of the Cloud9 IDE and selecting “New File” in the pull-down menu. A new file pane should open in the Cloud9 IDE.
2. Paste the following contents into your new file and then save the file as

item.json

 by clicking

File -> Save As

 in the upper left of the Cloud9 interface menu.

{

"year": {

"N": "2013"

},

"title": {

"S": "Rush"

},

"info": {

"M": {

"directors": {

"L": [

{

"S": "Ron Howard"

}

]

},

"release\_date": {

"S": "2013-09-02T00:00:00Z"

},

"rating": {

"N": "8.3"

},

"genres": {

"L": [

{

"S": "Action"

},

{

"S": "Biography"

},

{

"S": "Drama"

},

{

"S": "Sport"

}

]

},

"image\_url": {

"S": "http://ia.media-imdb.com/images/M/MV5BMTQyMDE0MTY0OV5BMl5BanBnXkFtZTcwMjI2OTI0OQ@@.\_V1\_SX400\_.jpg"

},

"plot": {

"S": "A re-creation of the merciless 1970s rivalry between Formula One rivals James Hunt and Niki Lauda."

},

"rank": {

"N": "2"

},

"running\_time\_secs": {

"N": "7380"

},

"actors": {

"L": [

{

"S": "Daniel Bruhl"

},

{

"S": "Chris Hemsworth"

},

{

"S": "Olivia Wilde"

}

]

}

}

}

}

1. Once you have saved the file, go back to the terminal window and enter the movie data into the table:

aws dynamodb put-item --table-name movies \

--return-consumed-capacity TOTAL \

--item file://item.json \

--condition-expression "attribute\_not\_exists(title)"

* --return-consumed-capacity TOTAL

 lets you know how many WCU/RCU are consumed during the operation.

* --condition-expression "attribute\_not\_exists(title)"

 ensures that you only create a new movies item if it does not exist.

 The output will be as below. This confirms that the data has been written successfully to the movies table.

{

"ConsumedCapacity": {

"TableName": "movies",

"CapacityUnits": 1.0

}

}

1. Open your email client, you will see an email from

no-reply@sns.amazonaws.com

 containing the full event as sent to your lambda function. The subject will be

Record Added

.

**Task 5: Enable Time to Live**

Amazon DynamoDB Time to Live (TTL) allows you to define a per-item timestamp to determine when an item is no longer needed. Shortly after the date and time of the specified timestamp, DynamoDB deletes the item from your table without consuming any write throughput. TTL is provided at no extra cost as a means to reduce stored data volumes by retaining only the items that remain current for your workload’s needs.

TTL is useful if you store items that lose relevance after a specific time. The following are example TTL use cases:

* Remove user or sensor data after one year of inactivity in an application.
* Archive expired items to an Amazon S3 data lake via DynamoDB Streams and AWS Lambda.
* Retain sensitive data for a certain amount of time according to contractual or regulatory obligations.

In this task you will enable TTL on the movies table, update the movie item you inserted in the previous task to include an

expires

. DynamoDB will expire the item and you should receive an email notification.

TASK 5.1: VIEW TTL STATUS

1. View ttl status of your table by entering the following command into a Cloud9 terminal.

aws dynamodb describe-time-to-live --table-name movies

 The output will be as below.

{

"TimeToLiveDescription": {

"TimeToLiveStatus": "DISABLED"

}

}

TASK 5.2: ENABLE TTL

In this task you will enable TTL on the movies table and specify an item attribute for it within DynamoDB

1. Enable ttl by entering the following command into a terminal.

aws dynamodb update-time-to-live \

--table-name movies \

--time-to-live-specification \

Enabled=true,AttributeName=expires

 The output will be as below:

{

"TimeToLiveSpecification": {

"Enabled": true,

"AttributeName": "expires"

}

}

TASK 5.3: MODIFY MOVIE ITEM

Now that you have enabled ttl, you will modify the movie item in the table to add an

expires

 attribute populated with a unix epoch timestamp 30 seconds into the future.

 For more information about using DynamoDB expressions see the **Additional Resources** section.

1. Populate

${FUTURE}

 shell variable.

FUTURE=$(date -d "+30 seconds" +%s)

1. Create a json file with update expression values.

echo "{\":exp\": {\"N\":\"${FUTURE}\" }}" > expression-attribute-values.json

1. Update the item, there will be no output from running this command.

aws dynamodb update-item \

--table-name movies \

--key '{ "year": {"N": "2013"}, "title": {"S":"Rush"} }' \

--update-expression "SET #EXP = :exp" \

--expression-attribute-names '{"#EXP": "expires"}' \

--expression-attribute-values file://expression-attribute-values.json

 For more information about using DynamoDB expressions see the **Additional Resources** section.

1. Now that you have set your TTL attribute and set up your Lambda notifications you can confirm an item modification to your table going to your email inbox. You will see a second email from

no-reply@sns.amazonaws.com

 with the subject “Record modified”

1. View the updated item attributes by entering the following command into a terminal window.

aws dynamodb get-item \

--table-name movies \

--key '{ "year": {"N": "2013"}, "title": {"S":"Rush"} }'

1. The output will be as below. Notice the

expires

 attribute has been added to the original json document you created before.

{

"Item": {

"info": {

"M": {

"actors": {

"L": [

{

"S": "Daniel Bruhl"

},

{

"S": "Chris Hemsworth"

},

{

"S": "Olivia Wilde"

}

]

},

"release\_date": {

"S": "2013-09-02T00:00:00Z"

},

"plot": {

"S": "A re-creation of the merciless 1970s rivalry between Formula One rivals James Hunt and Niki Lauda."

},

"genres": {

"L": [

{

"S": "Action"

},

{

"S": "Biography"

},

{

"S": "Drama"

},

{

"S": "Sport"

}

]

},

"image\_url": {

"S": "http://ia.media-imdb.com/images/M/MV5BMTQyMDE0MTY0OV5BMl5BanBnXkFtZTcwMjI2OTI0OQ@@.\_V1\_SX400\_.jpg"

},

"directors": {

"L": [

{

"S": "Ron Howard"

}

]

},

"rating": {

"N": "8.3"

},

"rank": {

"N": "2"

},

"running\_time\_secs": {

"N": "7380"

}

}

},

"year": {

"N": "2013"

},

"expires": {

"N": "1614281603"

},

"title": {

"S": "Rush"

}

}

}

1. Now that you have set your expiration attribute, DynamoDB will automatically delete the item from the table and forward an email notifying you. Check your inbox for a notification. You will receive a third email from

no-reply@sns.amazonaws.com

 with the subject “Record Expired”

**Note**: TTL is a best-effort feature and the item could remain available and visible in the table for up to 48 hours, depending on database size and activity. This table has has minimal size and activity, so ttl expiration should happen quickly. However, allow **10 minutes** for the item to be processed and an email notification to be sent your account.

While waiting to confirm record deletion you can proceed to the next task.

**Task 6: Archive Expired Items**

In this challenge task you will modify the lambda function code to save expired items to an S3 bucket.

You are encouraged to write your own code, however, fully working code is supplied.

If you are writing your own code you can skip ahead to step 57.

TASK 6.1: MODIFY LAMBDA FUNCTION CODE

1. In the Cloud9 IDE, open the

index.py

 file if you haven’t already done so.

In this task you will first create an S3 client to interact with the S3 service from the lambda function then you will write the event json to the bucket.

1. Locate

# TODO 1 : create an s3 client

 in the code and uncomment the line beginning with

#s3client

 (line 9). This will initialize a boto3 S3 client within your function.

# TODO 1 : create an s3 client

s3client = boto3.client('s3')

1. Locate

# TODO 2 : Enable code that writes to S3

 in the code and uncomment the section shown below (lines 26-29).

#

body = json.dumps(record)

s3client.put\_object(Body=body,

Bucket='<S3BUCKETNAME>',

Key='expired/' + record['eventID'] + '.json')

1. Locate

<S3BUCKETNAME>

 and replace with the

S3BucketName

 value found to the left of these lab instructions.

1. This section of code should now look like:

if 'userIdentity' in record:

# TODO 2 : Enable code that writes to S3

# REPLACE <S3BUCKETNAME> with the S3 bucket name in you lab guide

#

body = json.dumps(record)

s3client.put\_object(Body=body,

Bucket='XXXXXXXXXX-lambdastreamsonsumerbucket',

Key='expired/' + record['eventID'] + '.json')

subject = "Record expired"

1. Save the file by clicking

File -> Save

 in the upper left of the Cloud9 interface.

TASK 6.2 : DEPLOY LAMBDA

Now that you have edited the file, you will deploy the updated code to the AWS Lambda service.

1. Return to a terminal window in the Cloud9 IDE and prepare

index.py

 for deployment by zipping it.

zip consumer.zip index.py

1. Once your Lambda function has been compressed, update and publish the function with the following command:

aws lambda update-function-code --function-name ${FNAME} --zip-file fileb://consumer.zip --publish

1. The returned output should be similar to:

{

"FunctionName": "XX-LambdaStreamConsumer-XX",

"FunctionArn": "arn:aws:lambda:XX-XXXX-X:XXXXXXXXXXXX:function:XX-LambdaStreamConsumer-XX:1",

"Runtime": "python3.8",

"Role": "arn:aws:iam::XXXXXXXXXXXX:role/XX-LambdaStreamConsumerRole-XXXXXXXXXXXX",

"Handler": "index.handler",

"CodeSize": 717,

"Description": "",

"Timeout": 3,

"MemorySize": 128,

"LastModified": "2021-02-25T23:50:56.071+0000",

"CodeSha256": "yIg3KoXhuRz/dSJjmguI92jsr3myt3plX6Axr5Ts4fA=",

"Version": "1",

"Environment": {

"Variables": {

"SNSTOPIC": "arn:aws:sns:XX-XXXX-X:XXXXXXXXXXXX:XX-SNSTopic-XXXXXXXXXXXX"

}

},

"TracingConfig": {

"Mode": "PassThrough"

},

"RevisionId": "4edda443-4566-4150-b8ac-b8826b7b1f40",

"State": "Active",

"LastUpdateStatus": "Successful",

"PackageType": "Zip"

}

TASK 6.3 : ADD AN ITEM TO YOUR TABLE

In this task you will add an item to the movies table that includes an

expires

 attribute and a value set 30 seconds into the future.

1. Populate

${FUTURE}

 shell variable.

FUTURE=$(date -d "+30 seconds" +%s)

1. Create a json file with update expression values.

echo "{\":exp\": {\"N\":\"${FUTURE}\" }}" > expression-attribute-values.json

1. Update the item, there will be no output from running this command.

aws dynamodb update-item \

--table-name movies \

--key '{ "year": {"N": "2013"}, "title": {"S":"Enemy"} }' \

--update-expression "SET #EXP = :exp" \

--expression-attribute-names '{"#EXP": "expires"}' \

--expression-attribute-values file://expression-attribute-values.json

TASK 6.4: VERIFY TTL EXPIRATION AND S3 ARCHIVE

When the item has expired, DynamoDB will delete it from the table. This will write an event to the stream. Your modified lambda function will consume this event and write the record to the S3 bucket.

Remember that TTL is a best-effort, please **allow up 15 minutes** for this operation to complete.

1. Open your email client, you should see two new emails. One will have the subject

Record added

, the other will be

Record expired

.

1. List the contents of your S3 bucket. Be sure to replace

<S3BUCKETNAME>

 with the bucket name from the left of these lab instructions. The bucket should contain a single folder called

expired

 with a single json file inside.

aws s3 ls s3://<S3BUCKETNAME>/expired/

You should now see your expired item in your S3 bucket.

**Note** Depending on your speed , how fast you update-item expression-attribute-names to EXPIRES, you might observe two json files in the output.

Congratulations! You have successfully modified your workflow to archive movie items that have expired.

**Conclusion**

 Congratulations! You should now be able to:

* Use the AWS CLI and shell variables to enable DynamoDB Streams.
* Connect a Lambda function built with the AWS SDK for Python (Boto3) to a DynamoDB Stream.
* Send email notifications to a user whenever an item is added to, modified, or removed from a DynamoDB table.
* Automate DynamoDB item removal by enabling a Time To Live (TTL) attribute for a table item.
* Create a workflow to move expired DynamoDB items to long term storage in Amazon S3.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional Resources**

* **AWS CLI filters and queries**
  + https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-filter.html
* **AWS DynamoDB Expressions**
  + https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.html
* **AWS DynamoDB Streams**
  + https://docs.aws.amazon.com/cli/latest/reference/dynamodbstreams/describe-stream.html
* **Boto3 S3 documentation and examples**
  + https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html
* **Boto3 SNS documentation and examples**
  + https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sns.html
* **AWS Lambda**
  + https://aws.amazon.com/lambda/
  + https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html
* **AWS Step Functions**
  + https://aws.amazon.com/step-functions
* For more information about AWS Training and Certification, see [*http://aws.amazon.com/training/*](http://aws.amazon.com/training/).
* For more AWS Self-Paced Labs, see [*http://amazon.qwiklabs.com*](http://amazon.qwiklabs.com/).

For feedback, suggestions, or corrections, please email us at [*aws-course-feedback@amazon.com*](mailto:aws-course-feedback+spl@amazon.com).