Serverless Architectures with Amazon DynamoDB and Amazon Kinesis Streams with AWS Lambda

**SPL-51 - Version 3.3.2**

© 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**

This is a two-part lab. In part one of the lab, you create an AWS Lambda function from a blueprint, create an Amazon Kinesis Stream, then trigger the function with data from your stream and monitor the process with Amazon CloudWatch.

In part two of the lab, you learn the basics of event-driven programming using Amazon DynamoDB, DynamoDB Streams, and AWS Lambda. You learn the process of building a real-world application using triggers that combine DynamoDB Streams and Lambda.

OBJECTIVES

By the end of this lab, you should be able to do the following:

* Create an AWS Lambda function from a blueprint
* Create an Amazon Kinesis Stream
* Use Amazon CloudWatch to monitor Kinesis event data triggering your Lambda function
* Create an Amazon DynamoDB table and insert items
* Enable the Amazon DynamoDB Streams feature
* Configure and troubleshoot Lambda functions

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with DynamoDB and Kinesis by taking relevant introductory labs. Node.js and Python programming skills are recommended, although full solution code is provided. You should have completed the **Introduction to AWS Lambda** self-paced lab.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Learn more:** Where to find more information.
* **WARNING:** An action that is irreversible and could potentially impact the failure of a command or process (including warnings about configurations that cannot be changed after they are made).
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
* **Task complete:** A conclusion or summary point in the lab.

**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: 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.

LAB ENVIRONMENT

When you start the lab, the only resource is an EC2 server that is configured to send items to an Amazon Kinesis stream named “Lab-Stream”. However, you do not access this system during the lab.

SERVICES USED IN THIS LAB

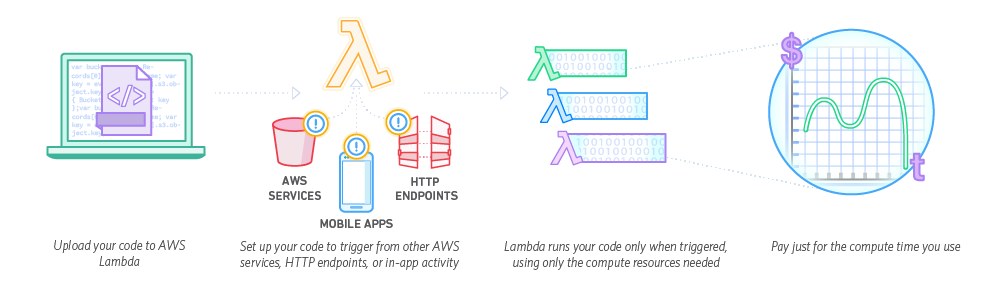
**AWS Lambda**

Lambda is a compute service that provides resizable compute capacity in the cloud to make web-scale computing easier for developers. You can upload your code to AWS Lambda and the service can run the code on your behalf using AWS infrastructure. AWS Lambda supports multiple coding languages, including Node.js, Java, or Python. After you upload your code and create a Lambda function, AWS Lambda takes care of provisioning and managing the servers used to run the code.

In this lab, you use AWS Lambda as an event-driven compute service where AWS Lambda runs your code in response to changes to data in an SNS topic and an Amazon S3 bucket.

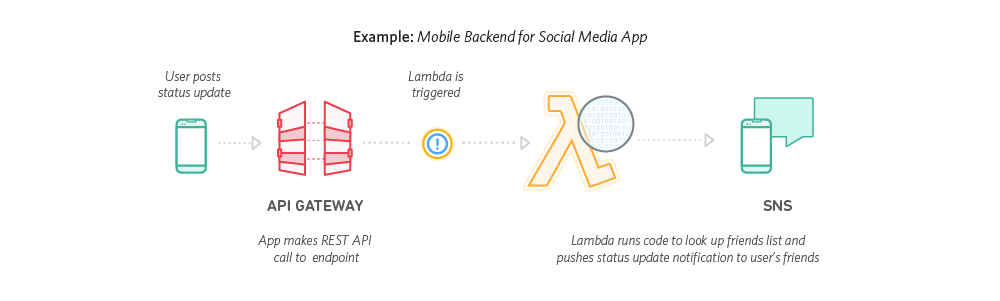
You can use AWS Lambda in two ways:

* As an event-driven compute service where AWS Lambda runs your code in response to events, such as new items in an Amazon Kinesis stream as you see in this lab.



*Image description: In the previous image, a sample workflow shows how Lambda can be configured to trigger when another AWS service activates.*

* As a compute service to run your code in response to HTTP requests using Amazon API Gateway or API calls.



*Image description: In the previous image, the Lambda function is called directly by an API gateway call.*

Lambda passes on to you the financial benefits of Amazon’s scale. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. With these capabilities, you can use Lambda to easily build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB, process streaming data stored in Amazon Kinesis, or create your own back end that operates at AWS scale, performance, and security.

**Learn more:** This lab guide explains basic concepts of AWS in a step by step fashion. However, it can only give a brief overview of Lambda concepts. Refer to *AWS Lambda Documentation* in the **Additional resources** section for more information.

**Lambda Blueprints**

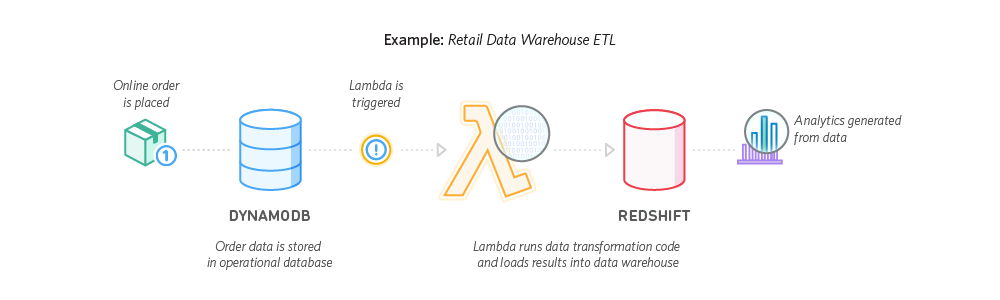
Blueprints are sample configurations of event sources and Lambda functions that do minimal processing for you. Most blueprints process events from specific event sources, such as Amazon S3 or DynamoDB. For example, if you select an s3-get-object blueprint, it provides sample code that processes an object-created event published by Amazon S3 that Lambda receives as parameter.

When you create a new AWS Lambda function, you can use a blueprint that best aligns with your scenario. You can then customize the blueprint as needed. You do not have to use a blueprint (you can author a Lambda function and configure an event source separately).

**Amazon DynamoDB**

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and supports both document and key-value data models. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad-tech, IoT, and many other applications.

**Learn more:** This lab guide explains basic concepts of AWS in a step by step fashion. However, it can only give a brief overview of DynamoDB. Refer to *Amazon DynamoDB Documentation* in the **Additional resources** section for more information.



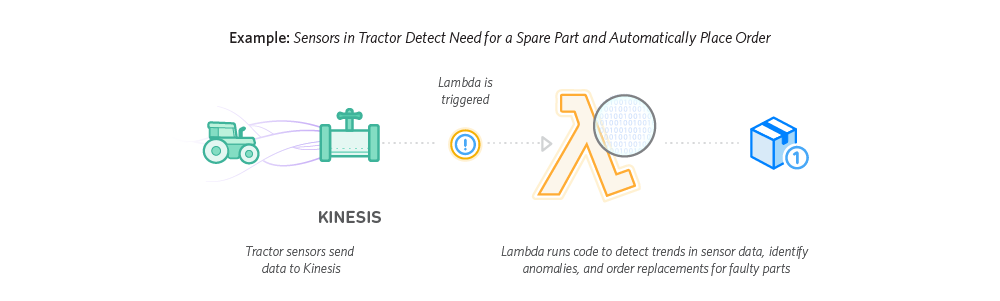
*Image description: In the previous image, a DynamoDB database is shown triggering a Lambda run which could move data into an Amazon Redshift database.*

**Amazon Kinesis**

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale. Amazon Kinesis can collect and process hundreds of terabytes of data per hour from hundreds of thousands of sources, allowing you to easily write applications that process information in real-time, from sources such as web site click-streams, marketing and financial information, manufacturing instrumentation and social media, and operational logs and metering data.

With Amazon Kinesis applications, you can easily send data to a variety of other services such as Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon Lambda, or Amazon Redshift. In a few clicks and a couple of lines of code, you can start building applications which respond to changes in your data stream in seconds, at any scale, while only paying for the resources you use.

**Learn more:** This lab guide explains basic concepts of AWS in a step by step fashion. However, it can only give a brief introduction to Amazon Kinesis. Refer to *Amazon Kinesis Documentation* in the **Additional resources** section for more information.



*Image description: In the previous image, a workflow shows how a Kinesis stream can collect sensor data and trigger a Lambda function to take action on the data.*

AWS SERVICES NOT USED IN THIS LAB

AWS service capabilities used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

**Part 1: Event-Driven Programming with Amazon Kinesis and AWS Lambda**

In the first part of this lab, you learn event-driven programming with Kinesis and Lambda.

**Task 1: Create an Amazon Kinesis Stream**

In this task, you create an Amazon Kinesis stream.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Kinesis

.

1. Choose **Create data stream** .

 The **Create data stream** page is displayed.

1. In **Data stream configuration** section, configure:

* **Data stream name**: Enter

Lab-Stream

**WARNING:** You MUST use exactly **Lab-Stream** as the name of the data stream, or subsequent steps of this lab will not work correctly.

1. In **Data stream capacity** section, configure:

* **Capacity mode**: Choose **Provisioned**
* **Provisioned shards**: Enter

1

Each shard supports a pre-defined capacity, as shown in the **Total data stream capacity** section. This lab only requires one shard, but applications requiring more capacity can simply request more shards.

1. Choose **Create data stream** .

After about 20 seconds, a message is displayed at the top of the page similar to this:

**Data stream Lab-Stream successfully created**

**Task complete:** You have successfully created the Amazon Kinesis data stream.

**Task 2: Create a Lambda Function**

In this task, you define an AWS Lambda function that is triggered by data coming into the stream.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lambda

.

1. Choose **Create function** .

You start by selecting a Lambda blueprint. Blueprints are pre-built for you and can be customized to suit your specific needs

1. Choose **Use a blueprint**.
2. In the **Basic information** section, configure:
3. Select the **Blueprint name** drop-down and search for:

kinesis-process-record-python

.

1. Choose **Process records sent to a Kinesis stream python3.10**.
2. Enter below details:

* **Function name**: Enter

ProcessKinesisRecords

* **Execution role**: Select  Use an existing role.
* **Existing role**: Select **lambda\_basic\_execution** from the drop-down list.

1. In the **Kinesis trigger** section, configure:

* **Kinesis stream**: select

Lab-Stream

This configures the Lambda function so that it is triggered whenever data comes into the Kinesis stream you created earlier. However, for now leave it as disabled until you test the function.

1. Un-select the checkbox for **Activate trigger** if it is already selected.
2. For **Batch size**, enter

10

.

1. For **Starting position**, select **Latest**.
2. Leave all other settings on the page at the default values.
3. Find and examine the Lambda blueprint displayed in the **Lambda function code** section. It does the following:

* Loop through each of the records received
* Decode the data, which is encoded in Base 64
* Print the data to the debug log

1. At the bottom of the screen, choose **Create function** .

Once the trigger is enabled, this function runs whenever data is sent to the stream.

**Task complete:** You have successfully created the AWS Lambda function.

**Task 3: Test your Function**

In this task, you simulate data coming from a stream to trigger your Lambda function using different methods, and compare the difference between using the *Latest* starting point and the *Trim\_horizon* starting point.

**Learn more:** The *Latest* starting point directs the function to start reading from the most recent entry to the data stream when first invoked. The *Trim\_horizon* starting point directs the function to start reading from the beginning of the data stream when first invoked. This difference is illustrated during this task. Refer to *StartingPosition* in the **Additional resources** section for more information.

TASK 3.1: TEST USING THE LAMBDA TEST TAB

1. In the browser tab with the **ProcessKinesisRecords** Lambda function, choose the **Test** tab.

**Note:** An event template for Kinesis is automatically selected. The event contains a simulated message arriving via Kinesis.

The **Test event** page is displayed.

1. For **Event name**: Enter

stream

1. Choose **Save** .
2. Choose **Test** .

The output should look similar to this:

**Executing function: succeeded (logs)**

1. Expand the **Details** section to see the output of the Lambda function.

You can see the following information about the Lambda execution in the Summary section:

* Init duration
* Billed duration
* Max memory used
* Log output

1. Choose the **Monitor** tab.

You can now view the CloudWatch metrics for your Lambda function. Metrics should be available for *Invocations* and *Duration*.

**Note:** If the metrics do not appear, wait a minute or two and then choose  refresh. Repeat until you see a data point on the Invocations graph.

TASK 3.2: TEST USING AN OUTSIDE DATA SOURCE

**Learn more:** For the sake of time, a Kinesis producer using the aws cli has already been setup for you. It is sending a numbered item to the *Lab-Stream* Kinesis stream every 10 seconds. Refer to *Tutorial: Using AWS Lambda with Amazon Kinesis* in the **Additional resources** section for more information.

1. On the browser tab with the Lambda function, choose the **Configuration** tab for the ProcessKinesisRecords function.
2. Choose **Triggers** from the list of items on the left.
3. Select the checkbox for the Kinesis trigger, then choose **Edit** .
4. Select the checkbox for **Activate trigger**, then choose **Save** .
5. Choose the **Monitor** tab.
6. Wait a minute or two, then choose  refresh. You should see more invocations on the graph. If you don’t seen more invocations, wait another minute or two and try again. Sometimes it takes up to five minutes for the metrics to show.
7. Choose **View CloudWatch logs** to open the logs in a new browser tab.
8. Choose **Search log group** .
9. Review the log, and note the various *Decoded payload* messages.

The first log should be for the Lambda test with a payload of “Hello, this is a test 123.” Subsequent payloads should have messages like “message number X”, and the number of *X* should increment in each message.

**Consider:** Which message number was the first to be logged? What happened to messages before that? Why aren’t the messages in batches of 10?

TASK 3.3: USE A DIFFERENT STARTING POINT FOR THE TRIGGER

1. Return to the browser tab with the Lambda function, then choose the **Configuration** tab for the ProcessKinesisRecords function.

**Note:** Once attached, the starting point for a Kinesis trigger cannot be changed. So to change the starting point, you must delete the old trigger and create a new one.

1. Choose **Triggers** from the list of items on the left.
2. Select the checkbox for the Kinesis trigger, then choose **Delete** .
3. Choose **Delete** to confirm the deletion of the trigger.
4. Choose **Close** .
5. Choose **Add trigger** .
6. In **Trigger configuration**, use the drop-down to select **Kinesis**.
7. For **Kinesis stream**, select **Lab-Stream**.
8. Select the checkbox for **Activate trigger** if it is not already selected.
9. For **Batch size**, enter

10

.

1. For **Starting position**, select **Trim horizon**.
2. Leave all other settings on the page at the default values.
3. Choose **Add** .
4. Return to the browser tab with the CloudWatch log.
5. Wait a minute or two, then choose  refresh.

**Note:** You may have to refresh a couple of times before you notice a change in the newer log entries. Give it at least two or three minutes.

1. Review the newer log entries.

**Consider:** How did switching to Trim\_horizon change how the data stream was read by the function? Which method (Latest or Trim\_horizon) would be best if the function needed to evaluate ALL possible data points? Which would be better for efficiently handling the newest data?

**Task complete:** You have successfully tested the AWS Lambda function using different methods and examined the difference between the *Latest* and *Trim\_horizon* starting points.

**Part 2: Event Driven Programming with Amazon DynamoDB and AWS Lambda**

In the second half of this lab, you learn about a different event driven programming method, this time with DynamoDB and Lambda.

**Task 4: Create Tables in DynamoDB**

In this task, you create a DynamoDB table that contains scores for online games.

1. At the top of the AWS Management Console, in the search bar, search for and choose

DynamoDB

.

1. Choose **Create table** and configure:

* **Table name**: Enter

GameScoreRecords

* **Partition key**: Enter

RecordID

 and select **Number** from the drop-down list

1. Choose **Create table** .

You now create another table for linking scores to users.

1. Choose **Create table** and configure:

* **Table name**: Enter

GameScoresByUser

* **Partition key**: Enter

Username

 and select **String** from the drop-down list

1. Choose **Create table** .

You now activate **DynamoDB Streams** on the first table. This generates streaming data whenever there is any change to the table (insert, update, delete).

1. Choose the first table you created, **GameScoreRecords**.
2. On the **Exports and Streams** tab, Scroll to **DynamoDB stream details**, choose **Turn on** .
3. In the **DynamoDB stream details** page, select:

* **View type**:  **New image**
* Choose **Turn on stream** .

Any record sent to this table now sends a message via DynamoDB streams, which triggers the Lambda function.

**Task complete:** You have successfully created the DynamoDB tables.

**Task 5: Create a Lambda Function**

In this task, you create a Lambda function that is triggered by updates to your DynamoDB table.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lambda

.

1. Choose **Create function** .
2. You provide the code to run, so choose **Author from scratch** .
3. Configure the following:

* **Function name**: Enter

AggregateScoresByUser

* **Runtime**: Choose **Node.js 16.x**
* Expand  **Change default execution role**
  + **Execution role:**  **Use an existing role**
  + **Existing role:** Select **lambda\_basic\_execution\_dynamodb** from the drop-down list

1. Choose **Create function** .
2. Select the **Code** tab, then:

* Delete all of the code in the **index.js** editor
* Copy and paste this code into the **index.js** editor:

// Set up AWS client

var AWS = require('aws-sdk');

var dynamodb = new AWS.DynamoDB();

exports.handler = function(event, context) {

// Keep track of how many requests are in flight

var inflightRequests = 0;

event.Records.forEach(function(record) {

console.log('DynamoDB Record: %j', record.dynamodb);

// Get the new image of the DynamoDB Streams record

var newItemImage = record.dynamodb.NewImage;

// Set the appropriate parameters for UpdateItem

// Refer to the ADD operation in the UpdateItem API for UpdateExpression

// http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API\_UpdateItem.html

// Adds the specified value to the item, if attribute does not exist, set the attribute

var updateItemParams = {

TableName: "GameScoresByUser",

Key : { Username : newItemImage.Username },

UpdateExpression : 'ADD Score :attrValue',

ExpressionAttributeValues : {':attrValue' : newItemImage.Score}

}

// Make a callback function to execute once UpdateItem request completes

// It may be helpful to refer to the updateItem method for the Javascript SDK

// http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html\#updateItem-property

var updateItemCallback = function(err, data) {

if (err) {

// log errors

console.log(err, err.stack);

} else {

// check if all requests are finished, if so, end the function

inflightRequests--;

if (inflightRequests === 0) {

context.succeed("Successfully processed " + event.Records.length + " records.");

}

}

};

// Send UpdateItem request to DynamoDB

dynamodb.updateItem(updateItemParams, updateItemCallback);

// Increase count for number of requests in flight

inflightRequests++;

});

// If there are no more requests pending, end the function

if (inflightRequests === 0) {

context.succeed("Successfully processed " + event.Records.length + " records.");

}

};

Examine the code. It does the following:

* Loop through each incoming record
* Create (ADD) an item in the *GameScoresByUser* table with the incoming score
* Wait until all updates have been processed

1. Choose **Deploy** .

You now configure the function to execute when a value is added to the DynamoDB table.

1. Scroll up to the **Function overview** section.
2. Choose **Add trigger** then configure:
3. In the **Trigger configuration** section, configure the following:

* **Select a source:** Select

DynamoDB

 from the drop-down list.

* **DynamoDB table:** Choose

GameScoreRecords

* Choose **Add** .

**Note:** The function is triggered when a new game score is added to the DynamoDB table. You can now test the function with a record that simulates an update of the database.

1. Choose the **Test** tab.
2. For **Event name**: Enter

score

.

1. Delete the existing test code (with key3, etc).
2. Copy and paste this record into the test event window:

{

"Records": [

{

"eventID": "1",

"eventVersion": "1.0",

"dynamodb": {

"Keys": { "RecordID": {"N": "2" }

},

"NewImage": {

"RecordID": {"N": "2" },

"Username": { "S": "Jane Doe" },

"Score": { "N": "100" },

"Nickname": { "S": "JaneD" }

},

"StreamViewType": "NEW\_IMAGE",

"SequenceNumber": "111",

"SizeBytes": 26

},

"awsRegion": "us-west-2",

"eventName": "INSERT",

"eventSourceARN": "arn:aws:dynamodb:us-west-2:account-id:table/GameScoreRecords/stream/2015-10-07T00:48:05.899",

"eventSource": "aws:dynamodb"

}

]

}

Examine the test record. It is simulating an incoming record from the *GameScoreRecords* table.

1. Choose **Save** .
2. Choose **Test** .
3. Confirm that the Lambda function has been invoked. A banner should state:

**Executing function: succeeded (logs)**

1. Expand  **Details** to view the output of the Lambda function.

You should see: **Successfully processed 1 records.**

VERIFY IN DYNAMODB

You now verify that the data was updated in DynamoDB.

1. At the top of the AWS Management Console, in the search bar, search for and choose

DynamoDB

.

1. In the navigation panel on the left side of the screen, choose **Tables**.
2. Select  **GameScoresByUser**.
3. On the top right side of the page, choose **Actions** then **Explore items**.

**Note:** This table was previously empty (you created it yourself), but you should now see an entry for **Jane Doe**.

TRIGGER THE UPDATE

You can perform more tests by inserting values in the Scores table, and confirming that the Lambda updates the User table.

1. Select the  **GameScoreRecords** table on the left.
2. Under **Items returned** section, choose **Create item** .

**Note:** In the following steps, you create a new item and assign it two attributes.

1. Enter

1

 in the text field next to **RecordID**.

1. Open the **Add new attribute** dropdown menu and follow the steps below to add a username attribute:

* Choose **String** from the drop-down list
* **Attribute name**: Enter

Username

* **Value**: Enter a name

1. Open the **Add new attribute** dropdown menu and follow the steps below to add a score attribute:

* Choose **Number** from the drop-down list
* **Attribute name**: Enter

Score

* **Value**: Enter a random integer

1. Choose **Create item** .
2. Confirm that your new item is displayed. It should have also triggered the Lambda function, resulting in a new entry in the other table.
3. Select the  **GameScoresByUser** table.

You should see that the new data you entered has been copied to the User table.

**Note:** Refresh  the page, if you do not see the data populated automatically.

Repeat the test by adding more items in the *GameScoreRecords* table.

**Task complete:** You have successfully triggered the Lambda function.

**Conclusion**

You have successfully done the following:

* Created a Lambda function from a blueprint.
* Created an Amazon Kinesis Stream and used it to trigger your Lambda function.
* Used CloudWatch to monitor your function.
* Created an Amazon DynamoDB table and inserted sample data.
* Enabled Amazon DynamoDB Streams.
* Tested and enabled the Lambda function on an Amazon DynamoDB table.

**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.

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).

**Additional resources**

* [AWS Lambda Documentation](https://docs.aws.amazon.com/lambda/)
* [Amazon DynamoDB Documentation](https://docs.aws.amazon.com/dynamodb/)
* [Amazon Kinesis Documentation](https://docs.aws.amazon.com/kinesis/)
* [StartingPosition](https://docs.aws.amazon.com/kinesis/latest/APIReference/API_StartingPosition.html)
* [Tutorial: Using AWS Lambda with Amazon Kinesis](https://docs.aws.amazon.com/lambda/latest/dg/with-kinesis-example.html)