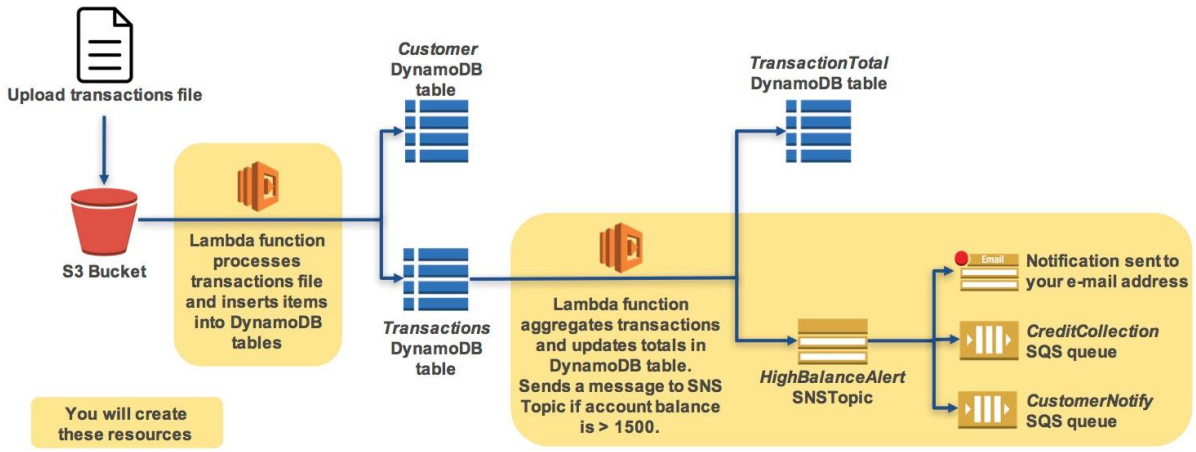
Implementing a Serverless Architecture with AWS Managed Services

In this lab, you will use AWS managed services to implement a serverless architecture.

Your system will receive a transactions file, automatically load its contents into a database and send notifications. This will be done without using any Amazon EC2 servers.

Scenario

The following diagram shows the lab scenario:



The scenario workflow is:

- You will **upload** a *transactions file* to an Amazon S3 bucket
- This will **trigger an AWS Lambda function** that will read the file and insert records into two **Amazon DynamoDB tables**
- This will **trigger another AWS Lambda function** that will calculate customer totals and will **send a message to an Amazon Simple Notification Service (SNS) Topic** if the account balance is over $1500
- Amazon SNS will then **send an email notification to you** and will **store a message in Amazon Simple Queue Service (SQS) queues** to notify the customer and your credit collection department.

Objectives

After completing this lab, you will be able to:

- Use AWS managed services to implement a serverless architecture
- Trigger AWS Lambda functions from Amazon S3 and Amazon DynamoDB

Duration

This lab takes approximately **45 minutes** to complete.

Accessing the AWS Management Console

1. At the top of these instructions, click Start Lab to launch your lab.

   A Start Lab panel opens displaying the lab status.

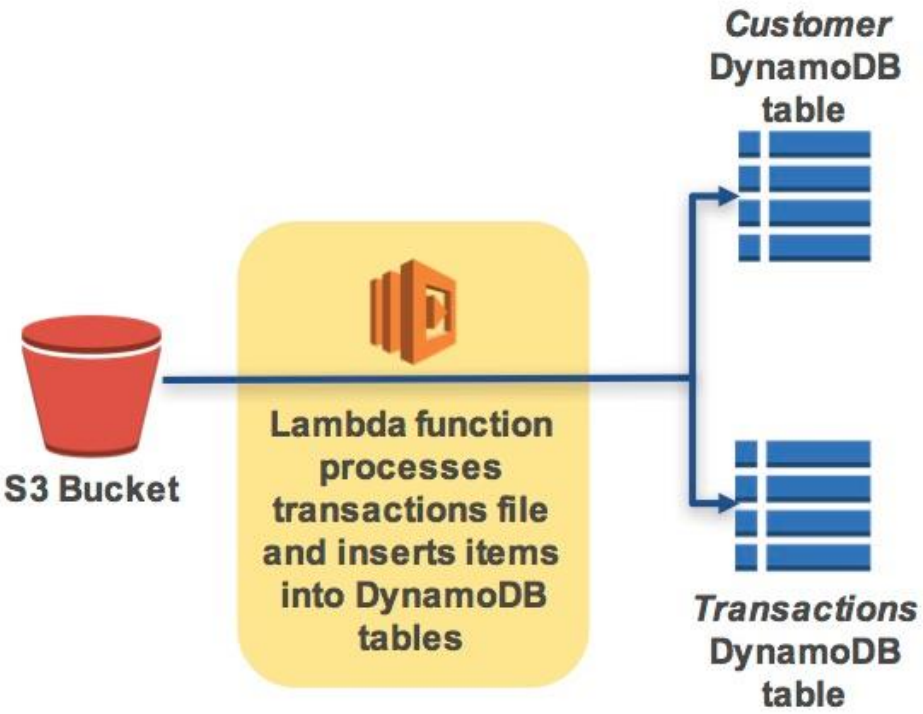
2. Wait until you see the message "**Lab status: ready**", then click the **X** to close the Start Lab panel.
3. At the top of these instructions, click AWS

   This will open the AWS Management Console in a new browser tab. The system will automatically log you in.

   Tip: If a new browser tab does not open, there will typically be a banner or icon at the top of your browser indicating that your browser is preventing the site from opening pop-up windows. Click on the banner or icon and choose "Allow pop ups."

4. Arrange the AWS Management Console tab so that it displays along side these instructions. Ideally, you will be able to see both browser tabs at the same time, to make it easier to follow the lab steps.

Task 1: Create a Lambda Function to Process a Transactions File



In this task, you will create **an AWS Lambda function** to process a transactions file. The Lambda function will read the file and insert information into the *Customer* and *Transactions* DynamoDB tables.

5. In the **AWS Management Console**, on the **Services** menu, click **Lambda**.
6. Click **Create a function**.

   Blueprints are code templates for writing Lambda functions. Blueprints are provided for standard Lambda triggers such as creating Alexa skills and processing Amazon Kinesis Firehose streams. This lab provides you with a pre-written Lambda function, so you will **Author from scratch**.

7. Configure the following:
   - **Name:** `TransactionProcessor`
   - **Runtime:** Python 2.7
   - **Execution Role:** Choose **Use an existing role**
   - **Existing role:** TransactionProcessorRole

   Note: This role gives execution permissions to your Lambda function so it can access Amazon S3 and Amazon DynamoDB.

8. Click **Create function**.
9. In the **Function code** section, delete all of the code that appears in the code editor.
10. Copy the code block below and paste it into the code editor.

```
# TransactionProcessor Lambda function
#
# This function is triggered by an object being created in an Amazon S3
bucket.
# The file is downloaded and each line is inserted into DynamoDB tables.

from __future__ import print_function
import json, urllib, boto3, csv

# Connect to S3 and DynamoDB
s3 = boto3.resource('s3')
dynamodb = boto3.resource('dynamodb')

# Connect to the DynamoDB tables
customerTable     = dynamodb.Table('Customer');
transactionsTable = dynamodb.Table('Transactions');

# This handler is executed every time the Lambda function is triggered
def lambda_handler(event, context):

  # Show the incoming event in the debug log
  print("Event received by Lambda function: " + json.dumps(event,
indent=2))

  # Get the bucket and object key from the Event
  bucket = event['Records'][0]['s3']['bucket']['name']
  key =
urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf
8')
  localFilename = '/tmp/transactions.txt'
```

```
  # Download the file from S3 to the local filesystem
  try:
    s3.meta.client.download_file(bucket, key, localFilename)
  except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. Make sure they exist and
your bucket is in the same region as this function.'.format(key, bucket))
    raise e

  # Read the Transactions CSV file. Delimiter is the '|' character
  with open(localFilename) as csvfile:
    reader = csv.DictReader(csvfile, delimiter='|')

    # Read each row in the file
    rowCount = 0
    for row in reader:
      rowCount += 1

      # Show the row in the debug log
      print(row['customer_id'], row['customer_address'], row['trn_id'],
row['trn_date'], row['trn_amount'])

      try:
        # Insert Customer ID and Address into Customer DynamoDB table
        customerTable.put_item(
          Item={
            'CustomerId': row['customer_id'],
            'Address':  row['customer_address']})

        # Insert transaction details into Transactions DynamoDB table
        transactionsTable.put_item(
          Item={
            'CustomerId':    row['customer_id'],
            'TransactionId':   row['trn_id'],
            'TransactionDate':  row['trn_date'],
            'TransactionAmount': int(row['trn_amount'])})

      except Exception as e:
         print(e)
         print("Unable to insert data into DynamoDB table".format(e))

    # Finished!
    return "%d transactions inserted" % rowCount
```

Examine the code. It performs the following steps:

- Downloads the file from Amazon S3 that triggered the event
- Loops through each line in the file
- Inserts the data into the *Customer* and *Transactions* DynamoDB tables

11. In the **Basic settings** section lower on the page, configure as shown:
    - **Description:** `Process data and send to DynamoDB tables`
    - **Timeout:** 20 sec

    You will now define a *trigger* that will activate the Lambda function.

12. Scroll up to the **Add triggers** section at the top of the page.
13. Under **Add triggers**, click **S3**.

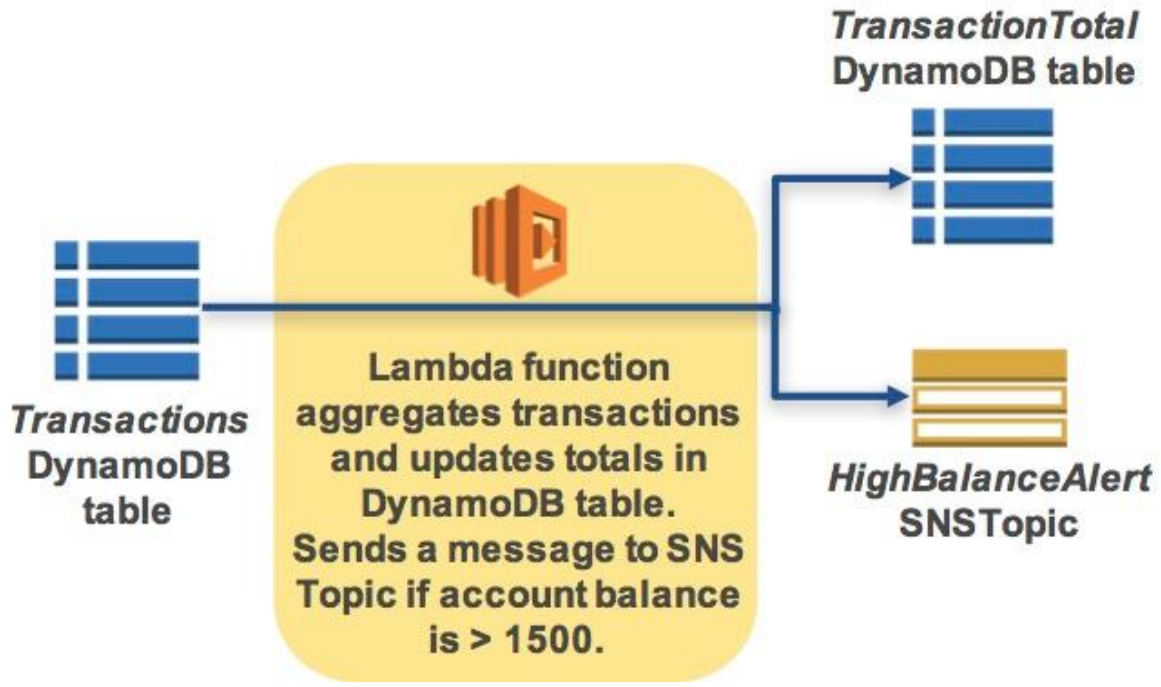
14. Scroll down to the **Configure triggers** panel and configure these settings:
    - **Bucket:** Select the Bucket that includes the words **inputs3bucketfortransact**. It will look similar to ...-*inputs3bucket*-...
    - **Event type:** All object create events
      Do *not* select *Object Removed*.

    The Lambda function will run whenever an object is created in your Amazon S3 bucket.

15. Click **Add** at the bottom of the page.
16. Click **Save** at the top of the page.

    Now whenever a file is uploaded to the selected Amazon S3 bucket, this Lambda function will execute. It will read the data from the uploaded file and will store the data it finds into the *Customer* and *Transactions* tables in **DynamoDB**.

Task 2: Create a Lambda Function to Calculate Transaction Totals and Notify About High Account Balances



In this task, you will create an **AWS Lambda function** to calculate transaction totals and send a Simple Notification Service notification if an account balance exceeds $1500.

17. In the top-left of the page, click **Functions**.
18. Click **Create function**.
19. Configure the following settings:
    - **Function name**: `TotalNotifier`
    - **Runtime:** Python 2.7
    - **Execution role:** Use an existing role

- **Existing role:** TotalNotifierRole

Note: This role gives AWS Lambda the execution permissions required to access Amazon DynamoDB and Amazon Simple Notification Service (SNS).

20. Click **Create function**.
21. In the **Function code** section, delete all of the code that appears in the code editor.
22. Copy the code block below, and paste it into the code editor:

```
# TotalNotifier Lambda function
#
# This function is triggered when values are inserted into the Transactions
DynamoDB table.
# Transaction totals are calculated and notifications are sent to SNS if
limits are exceeded.

from __future__ import print_function
import json, boto3

# Connect to SNS
sns = boto3.client('sns')
alertTopic = 'HighBalanceAlert'
snsTopicArn = [t['TopicArn'] for t in sns.list_topics()['Topics'] if
t['TopicArn'].endswith(':' + alertTopic)][0]

# Connect to DynamoDB
dynamodb = boto3.resource('dynamodb')
transactionTotalTableName = 'TransactionTotal'
transactionsTotalTable = dynamodb.Table(transactionTotalTableName);

# This handler is executed every time the Lambda function is triggered
def lambda_handler(event, context):

  # Show the incoming event in the debug log
  print("Event received by Lambda function: " + json.dumps(event,
indent=2))

  # For each transaction added, calculate the new Transactions Total
  for record in event['Records']:
    customerId = record['dynamodb']['NewImage']['CustomerId']['S']
    transactionAmount =
int(record['dynamodb']['NewImage']['TransactionAmount']['N'])

    # Update the customer's total in the TransactionTotal DynamoDB table
    response = transactionsTotalTable.update_item(
      Key={
        'CustomerId': customerId
      },
      UpdateExpression="add accountBalance :val",
      ExpressionAttributeValues={
        ':val': transactionAmount
      },
      ReturnValues="UPDATED_NEW"
    )

    # Retrieve the latest account balance
    latestAccountBalance = response['Attributes']['accountBalance']
    print("Latest account balance: " + format(latestAccountBalance))
```

```
    # If balance > $1500, send a message to SNS
    if latestAccountBalance >= 1500:

      # Construct message to be sent
      message = '{"customerID": "' + customerId + '", ' +
'"accountBalance": "' + str(latestAccountBalance) + '"}'
      print(message)

      # Send message to SNS
      sns.publish(
        TopicArn=snsTopicArn,
        Message=message,
        Subject='Warning! Account balance is very high',
        MessageStructure='raw'
      )

  # Finished!
  return 'Successfully processed {} records.'.format(len(event['Records']))
```

Examine the code. It performs the following steps:

- Connects to Amazon SNS and Amazon DynamoDB
- Calculates transaction totals and store them in the *TransactionTotal* DynamoDB table
- Sends a notification to Amazon SNS if the transaction total is over $1500

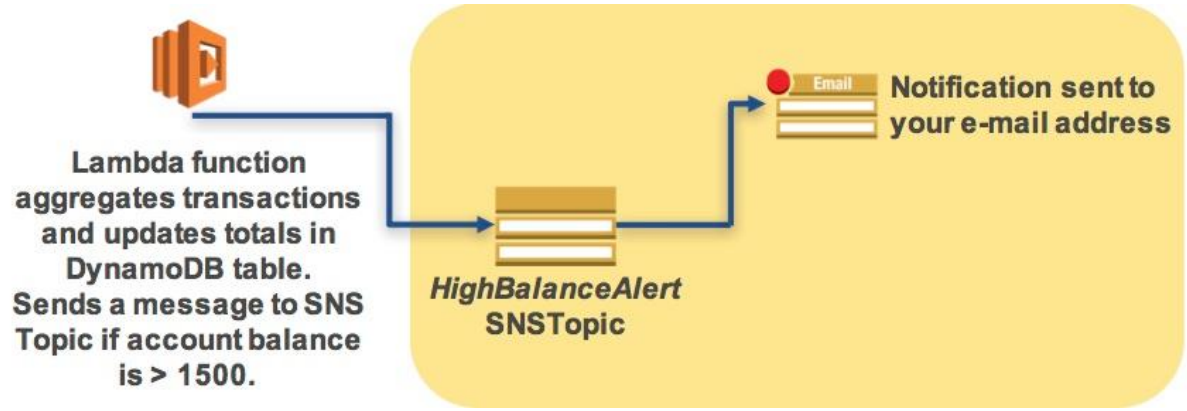
23. In the **Basic settings** section lower on the page:
    - **Description:** `Update total, send notification for balance exceeding $1500`
    - **Timeout:** 20 sec
24. Scroll up to the **Add triggers** section at the top of the page.
25. Under **Add triggers**, click **DynamoDB**.
26. Scroll down to the **Configure triggers** panel and configure these settings:
    - **DynamoDB table:** Transactions
    - **Starting position:** Latest

    The *Starting Position* tells Lambda whether to process the most recent information that was added to DynamoDB or whether to go back to the earliest data available (known as the *Trim Horizon*). This function will use the *Latest* information.

27. Click **Add**.
28. Click **Save** at the top of the page.

    Now whenever the *Transactions* DynamoDB table is updated, this function will calculate each customer's transaction total and store it in the *TransactionTotal* DynamoDB table. It the total exceeds $1500, it will send a message to a Simple Notification Service topic to notify the customer and your credit collection department.

Task 3: Create a Simple Notification Service (SNS) Topic



In this task, you will create a **Simple Notification Service (SNS) topic** that will receive a notification from your Lambda function when an account balance exceeds $1500. You will also subscribe to the topic with an email address and, optionally, via SMS.

29. On the **Services** menu, click **Simple Notification Service**.
30. On the left side of the screen, click on to reveal the Amazon SNS menu, and then click **Topics**.
31. Click **Create topic**.
32. In the **Create topic** dialog box, configure the following settings:
    - **Name:** `HighBalanceAlert`
      Please use this exact name so the Lambda function can trigger the notification.
    - **Display name:** `HighAlert`
33. Click **Create topic**.
34. Click **Create subscription**.
35. In the **Create subscription** dialog box, configure the following settings:
    - **Protocol:** Email
    - **Endpoint:** Enter an email address that you can easily access. This can be either a work or personal email address. This email will receive notifications from the Simple Notification Service topic you have created.
36. Click **Create subscription**.

    A confirmation request will be sent to your email address. You will need to confirm the subscription to receive notifications.

37. Check the email account you just provided for a new email from **HighAlert**. It may take a minute to be delivered.
38. When you receive the email, click the **Confirm subscription** link contained within the email message.

    The Simple Notification Service topic will now send you an email whenever it receives a message.

    Optional: You can also subscribe to receive a message on your phone via SMS. If you wish to do this, then:

- Click **Create subscription**
- For **Protocol**, select **SMS**
- For **Endpoint**, enter your phone number in international format (eg +14155557000 or +917513200000)
- Click **Create subscription**

Task 4: Create Two Simple Queue Service Queues

In this task, you will create two **Simple Queue Service (SQS) queues**. You will subscribe these queues to the Simple Notification Service (SNS) topic you just created. This setup is known as a *fan-out scenario* because each SNS notification is sent to multiple subscribers and those subscribers can independently consume messages from their own queue.

First, create a queue to notify the customer.

39. On the **Services** menu, then click **Simple Queue Service**.
40. Click **Get Started Now**.
41. In the **Create New Queue** dialog box, for **Queue Name**, type: `CustomerNotify`

    The interface will vary depending upon whether your region supports First-In-First-Out (FIFO) queues.

42. If you see "What type of queue do you need?", leave the default **Standard Queue** selected, then scroll down to the bottom.
43. Leave the remaining settings as their default and click either **Create Queue** or **Quick-Create Queue**.

    In a complete application environment, you would use a Lambda function or other application to read the messages in this queue and notify customers of the high balance.

    Next, create a queue to notify the credit collection department.

44. Click **Create New Queue**.
45. In the **Create New Queue** dialog box, for **Queue Name**, type: `CreditCollection`
46. Leave the remaining settings as their default and click **Create Queue** or **Quick-Create Queue**.

    In a complete application environment, you would use a Lambda function or other application to read the messages in this queue and notify your credit collection department to monitor this account.

47. Select the check boxes for *both* queues.
48. Click **Queue Actions**, then **Subscribe Queues to SNS Topic**.
49. In the **Subscribe to a Topic** dialog box, for **Choose a Topic**, click **HighBalanceAlert**, then click **Subscribe**.
50. In the **Topic Subscription Result** dialog box, click **OK**.

    Your two queues are now subscribed to your Simple Notification Service topic. They will automatically receive any messages sent to that topic.

Task 5: Testing the Serverless Architecture by Uploading a Transactions File

In this task, you will retrieve a transactions file and upload it to your S3 bucket. You will then test your serverless architecture.

Task 5.1: Upload the Transactions File

51. Right-click the link below and choose the option (e.g. 'Save Link As...') to download the transactions.txt file to your local machine.

    Download transactions.txt

52. On the **Services** menu, click **S3**.
53. Click on the bucket name with a name similar to *...-inputs3bucketfortransact-...*
54. Click **Upload**.
55. Click **Add files**, and then select the *transactions.txt* file that you downloaded.

    Tip: the file may be inside a 'ql' sub-directory in the direc

56. Click **Upload**.

    Uploading this file to Amazon S3 will immediately trigger the first Lambda function you created, which will immediately read the text file and store data in the **Customer** and **Transactions** DynamoDB tables.

Task 5.2: Check the DynamoDB tables

You can now verify that the transactions file was processed correctly by confirming that data has been loaded into the DynamoDB tables.

58. On the **Services** menu, click **DynamoDB**.
59. In the navigation pane, click **Tables**.
60. Click **Customer**.
61. In the **Items** tab, verify that there are items with the customer id and address for two customers.

    If no data is visible, then your Lambda function either had a failure or was not triggered. Ask your instructor for assistance in diagnosing the configuration.

62. Click **Transactions**.
63. In the **Items** tab, verify that several transactions exist. You should see 24 items total in the list.

    When information was added to the *Transactions* table, your second DynamoDB function would have been automatically triggered. This function calculates transaction totals for each account and stores the total in the *TransactionTotal* table. You can now check whether this process operated correctly.

64. Click **TransactionTotal**.
65. In the **Items** tab, verify that there are items with the customer id and account balance for two customers. Note that the account balance for customer *C2* is above $1500.

    If no data is visible, then your second Lambda function either had a failure or was not triggered. Ask your instructor for assistance in diagnosing the configuration.

Task 5.3: Check your SQS Queues

By now you should have received a new email from *HighAlert* that includes an alert about customer C2's high account balance. That same message was also sent to your two Simple Queue Service queues, ready to be picked up by another process.

If you provided a phone number, you should have also received an SMS notification. These are examples of the many ways that Amazon Simple Notification Service can send notifications to people and systems.

66. On the **Services** menu, click **Simple Queue Service**.

    If the Lambda function worked correctly, both queues should show one message in the **Messages Available** column. If no messages are available, ask your instructor for assistance in diagnosing the configuration.

67. Select **CreditCollection**.
68. Click **Queue Actions**,and then click **View/Delete Messages**.
69. Click **Start Polling for Messages**.
70. Click **More Details** in the message displayed.

71. Verify that the Message Body displays a warning for customer *C2*. Your message should look similar to this:

```
{
"Type" : "Notification",
"MessageId" : "eb0d030d-5f2d-5695-8f22-4c68d0335c0b",
"TopicArn" : "arn:aws:sns:us-east-
1:123456789:HighAccountBalanceAlertSNSTopic",
"Subject" : "Warning! Account balance is very high",
"Message" : "{\\"customerID\\": \\"C2\\",
\\"accountBalance\\":\\"1750\\"}",
...
}
```

72. Click **Close**.
73. Click **Close** again to return to your list of queues.
74. **Optional:** View the message in the **CustomerNotify** queue too. It should contain a message identical to the one in the **CreditCollection** queue.

Task 5.4: Check your Lambda Functions

Your Lambda functions automatically record logs and metrics. You can view this information to confirm that your functions executed correctly.

75. On the **Services** menu, click **Lambda**.
76. Click **TransactionProcessor** (click on the name itself), and then click the **Monitoring** tab to view CloudWatch metrics for the Lambda function.

    The metrics should indicate that the Lambda function was invoked and that no errors occurred.

77. In the top-left of the page, click **Functions**
78. Click **TotalNotifier** (click on the name), and then click the **Monitoring** tab to view CloudWatch metrics for the Lambda function.

    Log files are useful for debugging Lambda functions. They are stored in Amazon CloudWatch Logs.

79. In the **Invocations** chart, click **View logs in CloudWatch** to see the logs for your function. Click the link displayed and look through the logs.

Lab Complete

Congratulations! You have completed the lab.

80. Click End Lab at the top of this page and then click Yes to confirm that you want to end the lab.

A panel will appear, indicating that "DELETE has been initiated... You may close this message box now."

81. Click the **X** in the top right corner to close the panel.

For feedback, suggestions, or corrections, please email us at: *aws-course-feedback@amazon.com*