**Architecting on AWS - Lab 4 - Implementing a Serverless Architecture With Managed Services**

* ***Lab Instructions***

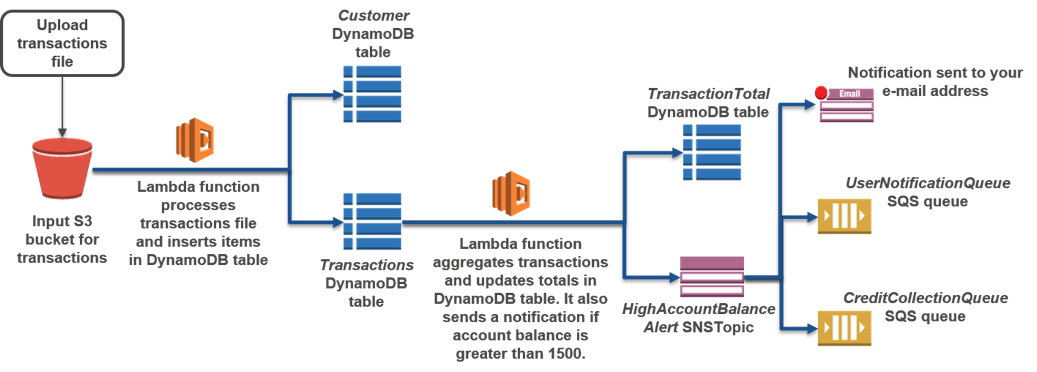
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **lab4-serverless-guide.md** |  |

* lab4-serverless-guide.md
* Lab4-Serverless-CommandRef.txt
* [Lab 4 - Implementing a Serverless Architecture with AWS Managed Services](https://globalknowledge.qwiklab.com/focuses/17010#lab-4-implementing-a-serverless-architecture-with-aws-managed-services)
  + [Accessing the AWS Management Console](https://globalknowledge.qwiklab.com/focuses/17010#accessing-the-aws-management-console)
  + [Task 1 - Reviewing Resources in Lab Environment](https://globalknowledge.qwiklab.com/focuses/17010#task-1-reviewing-resources-in-lab-environment)
    - [Task 1.1: Review Resources](https://globalknowledge.qwiklab.com/focuses/17010#task-1-1-review-resources)
  + [Task 2:Setting Up Simple Notification Service (SNS) Topic and Simple Queue Service (SQS) Queues](https://globalknowledge.qwiklab.com/focuses/17010#task-2-setting-up-simple-notification-service-sns-topic-and-simple-queue-service-sqs-queues)
    - [Task 2.1: Set Up an Simple Notification Service Topic](https://globalknowledge.qwiklab.com/focuses/17010#task-2-1-set-up-an-simple-notification-service-topic)
    - [Task 2.2: Set Up Two Simple Queue Service (SQS) Queues](https://globalknowledge.qwiklab.com/focuses/17010#task-2-2-set-up-two-simple-queue-service-sqs-queues)
  + [Task 3: Configuring Lambda Functions](https://globalknowledge.qwiklab.com/focuses/17010#task-3-configuring-lambda-functions)
  + [Task 3.1: Create Lambda Function to Process Transactions Text File](https://globalknowledge.qwiklab.com/focuses/17010#task-3-1-create-lambda-function-to-process-transactions-text-file)
  + [Task 3.2: Create Lambda Function to Calculate Transaction Totals and Notify About High Account Balances](https://globalknowledge.qwiklab.com/focuses/17010#task-3-2-create-lambda-function-to-calculate-transaction-totals-and-notify-about-high-account-balances)
  + [Task 4 - Testing Serverless Architecture by Uploading Transactions File](https://globalknowledge.qwiklab.com/focuses/17010#task-4-testing-serverless-architecture-by-uploading-transactions-file)
    - [Task 4.1: Upload Transactions File to S3 bucket](https://globalknowledge.qwiklab.com/focuses/17010#task-4-1-upload-transactions-file-to-s3-bucket)
  + [Task 4.2: Verify Implementation](https://globalknowledge.qwiklab.com/focuses/17010#task-4-2-verify-implementation)
  + [Lab Complete](https://globalknowledge.qwiklab.com/focuses/17010#lab-complete)

Lab 4 - Implementing a Serverless Architecture with AWS Managed Services

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

The following diagram shows the lab scenario.



The lab environment is already setup with the following resources:

* Amazon S3 bucket: You will upload a text file which contains a list of credit card transactions to this bucket.
* Amazon DynamoDB tables
  + *Customer* table: This table will store the customer id and customer address.
  + *Transactions* table: This table will store the transaction id, transaction date, and transaction amount. The streams feature is enabled for this table.
  + *TransactionTotal* table: This table will store the current account balance (aggregate transaction amount) for each customer.
* IAM roles: You will use the IAM roles to assign execution permissions to AWS Lambda.

You will configure the following resources:

* *HighAccountBalanceAlertSNSTopic*: This Amazon SNS topic will be used to send notifications if the account balance for a customer is greater than 1. The notifications in this topic are fanned out to SQS queues.
* *UserNotificationQueue*: This Amazon SQS queue subscribes to notifications from *HighAccountBalanceAlertSNSTopic*.
* *CreditCollectionQueue*: This Amazon SQS queue subscribes to notifications from *HighAccountBalanceAlertSNSTopic*.
* Lambda functions:
  + *TransactionProcessor*: This Lambda function will process the transactions text file that is uploaded to the S3 bucket and insert items into the *Customer* and *Transactions* DynamoDB tables.
  + *TransactionAggregatorNotifier*: This Lambda function will update the account balance in the *TransactionTotal* table and send a notification when the account balance exceeds 1500.

**Objectives**

After completing **this** lab, you will be able to:

* Use AWS managed services to implement a serverless architecture.
* Set up Lambda functions to act as triggers in a DynamoDB table.

**Prerequisites**

This lab requires:

* Access to a notebook computer with Wi-Fi running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat)
* For Microsoft Windows users: Administrator access to the computer
* An Internet browser such as Chrome, Firefox, or IE9 (previous versions of Internet Explorer are not supported)

**Duration**

This lab will require around **45 minutes** to complete.

Accessing the AWS Management Console

1. To the right of the lab title, click **Start Lab** to launch your Qwiklabs.

C:\Users\student\Desktop\architect\lab4\Architecting on AWS - Lab 4 - Implementing a Serverless Architecture With Managed Services  Qwiklabs + globalknowledge_files\start.png

1. On the **Connect** tab of the Qwiklabs page, copy the **Password** to the clipboard and then click **Open Console**.

C:\Users\student\Desktop\architect\lab4\Architecting on AWS - Lab 4 - Implementing a Serverless Architecture With Managed Services  Qwiklabs + globalknowledge_files\open.png

1. Sign in to the AWS Management Console using the following steps:
   * For **User Name**, type **awsstudent**
   * For **Password**, paste the password copied from the clipboard.
   * Click **Sign In**.

Task 1 - Reviewing Resources in Lab Environment

In this section, you will explore the resources that have been set up in the lab environment.

**Task 1.1: Review Resources**

1. In the **AWS Management Console** , on the **Services** menu, click **CloudFormation**.
2. Select the check box next to the stack name that begins with *qls*.
3. On the **Outputs** tab, review the resources that have been created in the lab environment. You should see they include four entries: **InputS3BucketForTransactionsFilesName** , **CustomerDynamoDBTable** , **TransactionsDynamoDBTable** , and **TransactionTotalDynamoDBTable**.
4. (Optional) On the **Services** menu, click **S3** to view the S3 bucket created for this lab (it includes **inputs3bucketfortransact** in the name).
5. (Optional) On the **Services** menu, click **DynamoDB**. In the navigation pane, click **Tables** to view the tables created for this lab. Note that these tables currently contain no items. They will be populated automatically as you complete this lab.

Task 2:Setting Up Simple Notification Service (SNS) Topic and Simple Queue Service (SQS) Queues

In this task, you will set up a Simple Notification Service topic and two Simple Queue Service queues. You will subscribe the Simple Queue Service queues to the Simple Notification Service topic. This setup is known as a fan-out scenario because each Simple Notification Service notification is distributed to multiple Simple Queue Service queues

**Task 2.1: Set Up an Simple Notification Service Topic**

In this section, you will set up the SNS topic that will receive notifications about a customer's high account balance. You will also subscribe to the topic with an email address.

1. In the **AWS Management Console** , on the **Services** menu, click **SNS** , and then click **Get started** on the main page.
2. In the navigation pane, click **Topics** , and then click **Create new topic**.
3. In the **Create new topic** dialog box, enter the following properties:
   * For **Topic name**, type 
   * For **Display name**, type 
4. Click **Create topic**.
5. Click the link with the topic ARN for **HighAccountBalanceAlertSNSTopic**.
6. Copy and paste the **Topic ARN** into a text editor. You will use this value later in the lab.
7. Click **Create subscription**.
8. In the **Create subscription** window that appears, click **Protocol** > **Email**.
9. For **Endpoint** , type in 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 SNS Topic you've created.
10. Click **Create subscription**.
11. Check the email account you just provided for a new email from **HABTopic**. It may take a minute to be delivered.
12. When you receive the email, open it and click the **Confirm subscription** link contained within.

This SNS topic will now notify your email if it receives a message, however it will also have two more subscribers: the two SQS queues you will create next.

**Task 2.2: Set Up Two Simple Queue Service (SQS) Queues**

In this subtask, you will set up two SQS queues that will subscribe to notifications from the SNS topic you just created.

1. In the **AWS Management Console** , on the **Services** menu, click **SQS** , and then click **Get Started Now** on the main page.
2. In the **Create New Queue** dialog box, for **Queue Name** , type 
3. Leave the remaining settings as their default and click **Create Queue**.  
   **Note** In a complete application environment, you could use a Lambda function or other application to read the messages in this queue and notify users of a high balance.
4. Click **Create New Queue**.
5. In the **Create New Queue** dialog box, for **Queue Name** , type 
6. Leave the remaining settings as their default and click **Create Queue**.  
   **Note** In a complete application environment, you could use a Lambda function or other application to read the messages in this queue and notify your credit collection department to monitor this account.
7. Select the check boxes for both queues.
8. Click **Queue Actions** > **Subscribe Queues to SNS Topic**.
9. In the **Subscribe to a Topic** dialog box, for **Choose a Topic** , click **HighAccountBalanceAlertSNSTopic** , and then click **Subscribe**.
10. In the **Topic Subscription Result** dialog box, click **OK**.

Your two queues are now subscribed to your SNS topic. They will automatically receive any messages pushed to that topic.

Task 3: Configuring Lambda Functions

In this task, you will set up Lambda functions to:

* Process transactions text file
* Calculate transaction totals and notify about high account balances

Task 3.1: Create Lambda Function to Process Transactions Text File

In this section, you will set up a Lambda function to process the transactions text file that you will upload to your S3 bucket, insert customer id and address into the Customer table, and insert transaction information into the Transactions DynamoDB table.

1. On the **Services** menu, click **Lambda**.
2. Click **Get Started Now**.
3. On the **Select blueprint** page, click **Blank Function**.
4. On the **Configure triggers** page, click the empty square to the left of the Lambda icon.
5. In the drop-down list that appears, click **S3**.
6. For **Bucket** , click the bucket that begins with *qls* and contains the text *inputs3bucket*.
7. For **Event type** , click **Object Created (All)**.
8. Click the checkbox for **Enable trigger** so that it is checked.
9. Click **Next**.
10. On the Configure function page, for **Name** , type 
11. For **Description** , type 
12. For **Runtime** , click **Python 2.7**.
13. For **Code entry type** , make sure that **Edit code inline** is selected.
14. Delete all of the code that appears below the **Code entry type** section so that the field is empty.
15. Copy and paste the first Lambda function's code:

Copy Code Block

from \_\_future\_\_ import print\_function

import json

import urllib

import boto3

import csv

print('Loading function')

customerTableName = 'Customer'

transactionsTableName = 'Transactions'

s3 = boto3.resource('s3')

dynamodb = boto3.resource('dynamodb')

customerTable = dynamodb.Table(customerTableName);

transactionsTable = dynamodb.Table(transactionsTableName);

def lambda\_handler(event, context):

print("Received event: " + 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('utf8')

trnFileName = '/tmp/transactions.txt'

try:

s3.meta.client.download\_file(bucket, key, trnFileName)

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(trnFileName) as csvfile:

reader = csv.DictReader(csvfile, delimiter='|')

for row in reader:

print(row['customer\_id'], row['customer\_address'], row['trn\_id'], row['trn\_date'], row['trn\_amount'])

# Insert customer id and address in customer DynamoDB table

try:

resp = customerTable.put\_item(

Item={

'CustomerId': row['customer\_id'],

'Address': row['customer\_address']})

resp = 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))

return "done"

1. For **Role**, make sure that **Choose an existing role** is already selected.
2. For **Existing role** , click the role that contains the text ***S3LambdaDynamoDBRole***. This gives AWS Lambda the execution permissions required to access Amazon S3 and Amazon DynamoDB.
3. **Note** Do not select the role that contains the text *SNS*.
4. Change the **Timeout** value to **0 min 20 sec**.
5. Leave other values with default settings, and then click **Next**.
6. **Note** Since you will not be using any resources that exist inside of a VPC for this architecture, you do not need to specify a VPC for this function.
7. Review your function's settings and click **Create function**.

Now whenever a file is uploaded to your input bucket in Amazon S3, this Lambda function will run and sort the data it finds into the **Customer** and **Transactions** tables in **DynamoDB**.

Task 3.2: Create Lambda Function to Calculate Transaction Totals and Notify About High Account Balances

In this section, you will set up a Lambda function to calculate transaction totals and send an SNS notification if an account balances exceeds 1500.

1. Click **Functions** near the top left corner of the page.
2. Click **Create a Lambda function**.
3. On the **Select blueprint** page, click **Blank Function**.
4. On the **Configure triggers** page, click the empty square to the left of the Lambda icon.
5. In the drop-down list that appears, click **DynamoDB**.
6. In the drop-down list for **DynamoDB table**, select **Transactions**.
7. In the drop-down list for **Starting position**, select **Trim horizon**.
8. Click the checkbox for **Enable trigger** so that it is checked.
9. Click **Next**.
10. On the Configure function page, for **Name** , type 
11. For **Description**, type 
12. For **Runtime**, click **Python 2.7**.
13. For **Code entry type** make sure that **Edit code inline** is selected.
14. Delete all of the code that appears below the **Code entry type** section so that the field is empty.
15. Copy and paste the second Lambda function code into the empty field.

Copy Code Block

from \_\_future\_\_ import print\_function

import json

import boto3

print('Loading function')

### STUDENT TODO: Update the value of snsTopicArn ###

snsTopicArn = '<ARN for HighAccountBalanceAlertSNSTopic>'

dynamodb = boto3.resource('dynamodb')

transactionTotalTableName = 'TransactionTotal'

transactionsTotalTable = dynamodb.Table(transactionTotalTableName);

sns = boto3.client('sns')

def lambda\_handler(event, context):

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

for record in event['Records']:

customerId = record['dynamodb']['NewImage']['CustomerId']['S']

transactionAmount = int(record['dynamodb']['NewImage']['TransactionAmount']['N'])

response = transactionsTotalTable.update\_item(

Key={

'CustomerId': customerId

},

UpdateExpression="add accountBalance :val",

ExpressionAttributeValues={

':val': transactionAmount

},

ReturnValues="UPDATED\_NEW"

)

print("Added transaction to account balance in TransactionTotal table")

latestAccountBalance = response['Attributes']['accountBalance']

print("Latest account balance: ".format(latestAccountBalance))

if latestAccountBalance >= 1500:

message = '{"customerID": "' + customerId + '", ' + '"accountBalance": "' + str(latestAccountBalance) + '"}'

print(message)

print("Account balance is very high: ".format(latestAccountBalance))

sns.publish(

TopicArn=snsTopicArn,

Message=message,

Subject='Warning! Account balance is very high',

MessageStructure='raw'

)

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

1. On **line 9** of the function code you've just pasted, you will see this:

Copy Code Block

snsTopicArn = '\_<ARN for HighAccountBalanceAlertSNSTopic>\_'

1. Replace with the ARN that you noted earlier. Replace the text within the single quotes. Do not delete the single quotes.
2. For **Role**, make sure that **Choose an existing role** is already selected.
3. For **Existing role**, click the role that contains the text *SNSLambdaDynamoDBRole*. This gives AWS Lambda the execution permissions required to access Amazon DynamoDB and Amazon SNS.

**Note** Do not select the role that contains the text *S3*.

1. Change the **Timeout** value to **0 min 20 sec**.
2. Leave other values with default settings, and then click **Next**.
3. Review your function's settings and click **Create function**.

You have now configured the second Lambda function, which starts when the **Transactions** table is updated, and then takes the data it finds there, adds up each customer's transaction amounts, stores that value in the **TransactionTotal** table, and sends a message to your SNS topic if a customer's balance exceeds 1500.

Task 4 - Testing Serverless Architecture by Uploading Transactions File

In this task, you will upload a transactions file to an S3 bucket. You will verify that:

* The *TransactionProcessor* Lambda function has processed the input transaction file and inserted items into the *Customer* and *Transactions* DynamoDB tables.
* The *TransactionAggregatorNotifier* Lambda function has updated transaction totals in the *TransactionTotal* DynamoDB table and has sent SNS notifications about high account balances.
* The *HighAccountBalanceAlertSNSTopic* SNS topic has fanned the notification out to the SQS queues that have subscribed to it.

**Task 4.1: Upload Transactions File to S3 bucket**

In this section, you will retrieve the transactions file and upload it to the S3 bucket that has been created for this lab.

1. Click on the following link to download the transactions.txt file to your local machine. If your web browser displays the file instead of downloading it, right click on the web page, and then save the page as a text file.

<https://s3-us-west-2.amazonaws.com/us-west-2-aws-staging/awsu-ilt/academy-cca/v3.0/labs/lab12-serverless/scripts/transactions.txt>

1. On the **Services** menu, click **S3**.
2. Click the bucket name that begins with the text *qls* and contains the text *inputs3bucket*.
3. Click **Upload**.
4. Click **Add Files** , and then select the transactions.txt file that you downloaded earlier and click **Open**.
5. Click **Start Upload**.

Uploading this file to Amazon S3 will immediately trigger the first Lambda function you created, which will immediately start sorting the data it finds in the uploaded text file and storing customer data from it into your **Customers** DynamoDB table.

Task 4.2: Verify Implementation

In this section, you will verify that the transactions file was processed correctly.

1. On the **Services** menu, click **DynamoDB**.
2. In the navigation pane, click **Tables**.
3. Click **Customer**.
4. In the **Items** tab, verify that there are items with the customer id and address for two customers.
5. Click **Transactions**.
6. In the **Items** tab, verify that several transactions exist. You should see 24 items total in the list.
7. Click **TransactionTotal**.
8. In the **Items** tab, verify that there are items with the customer id and account balance for two customers. Note the account balance for customer *C2*.

You should have by now received another email from **HABTopic** that includes an alert about customer C2's high account balance. That same message was also sent to your two SQS queues, ready to be picked up by another process (not included in this lab).

To see the messages sent to your queues in the console, you need to start polling for them, which is what you will do next.

1. On the **Services** menu, click **SQS**.
2. Click **CreditCollection**.
3. Click **Queue Actions** > **View/Delete Messages**.
4. Click **Start Polling for Messages**. Wait for the progress bar at the bottom to complete before proceeding.
5. Click **More Details** in the message displayed.
6. Verify that the *Message* attribute displays a warning for customer *C2*, and then click **Close**.

Copy Code Block

{

"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\"}",

...

}

1. Click **Close** again to return to your list of queues.
2. Clear the check box for **Credit Collection** and select the check box for **UserNotification**.
3. Click **Queue Actions** > **View/Delete Messages**.
4. Click **Start Polling for Messages**. Wait for the progress bar at the bottom to complete before proceeding.
5. Click **More Details** in the message displayed.
6. Verify that the *Message* attribute displays a warning for customer *C2*, and then click **Close**.
7. Click **Close** again to return to your list of queues.
8. On the **Services** menu, click **Lambda**.
9. Click **TransactionProcessor** , and then click the **Monitoring** tab to view CloudWatch metrics for the Lambda function. You can also click **View logs in CloudWatch** to see the logs for your function.
10. Return to the **Lambda Console** and click **Functions** near the top of the page.
11. Click **TransactionAggregatorNotifier** , and then click the **Monitoring** tab to view CloudWatch metrics for the Lambda function and view logs in Amazon CloudWatch.

Lab Complete

Congratulations! You have successfully completed the lab. To clean up your lab environment, do the following:

1. To sign out of the **AWS Management Console** click **awsstudent** in the navigation bar, and then click **Sign Out**.
2. Return to the **qwikLABS** page where you launched your lab and click **End**.

© for the use of: shweta\_misra@fanniemae.com

* Connect
* Lab Details
* Addl. Info
* **Lab Connection**   
  Please follow the lab instructions to connect to your lab

**Warning:** Do not transmit data into the AWS Console that is not related to Qwiklabs or the lab you are taking.

* + **AWS Console Details**
  + **Access Key Details**
  + [Show Access Keys](https://globalknowledge.qwiklab.com/focuses/17010#accessKeyIdModal)
  + **Key Pair Details**

|  |  |
| --- | --- |
| **Setup Time (min.)** | 3 |
| **Duration (min.)** | 120 |
| **Access (min.)** | 120 |

|  |
| --- |
| **Tags:** [**AWS-100-ARC-51-lab-4-serverless**](https://globalknowledge.qwiklab.com/tags/AWS-100-ARC-51-lab-4-serverless)  **Lab Description:** In this lab, you will learn how to use AWS managed services to implement a serverless architecture.  **Lab Creator:** qwikLABS publisher  **Date Created:** May 18, 2017 08:51  **AWS Region::** [us-west-2] **US West (Oregon)** |

Additional Lab Information:

* **InputS3BucketForTransactionsFilesName:**
* qls-134979-15e5dfe6d3a8b-inputs3bucketfortransact-1sxwwrnjzzn10
* **CustomerDynamoDBTable:**
* Customer
* **TransactionsDynamoDBTable:**
* Transactions
* **TransactionTotalDynamoDBTable:**
* TransactionTotal

×

**Access Key Details**

Use access keys to make secure REST or Query protocol requests to any AWS service API.