**uided Lab 2: Implementing a Serverless Architecture on AWS**

**Scenario**

You are creating an inventory tracking system. Stores from around the world will upload an inventory file to Amazon S3. Your team wants to be able to view the inventory levels and send a notification when inventory levels are low.

In this lab, you will:

* You will *upload* an *inventory file* to an Amazon S3 bucket.
* This upload will *trigger a Lambda function* that will read the file and insert items into an *Amazon DynamoDB table*.
* A serverless, web-based dashboard application will use Amazon Cognito to authenticate to AWS. The application will then gain access to the DynamoDB table to display inventory levels.
* Another Lambda function will receive updates from the DynamoDB table. This function will send a message to an *SNS topic* when an inventory item is out of stock.
* Amazon SNS will then *send you a notification through short message service (SMS) or email* that requests additional inventory.

**Lab overview**

Traditionally, applications run on servers. These servers can be physical (or bare metal). They can also be virtual environments that run on top of physical servers. However, you must purchase and provision all these types of servers, and you must also manage their capacity. In contrast, you can run your code on AWS Lambda without needing to pre-allocate servers. With Lambda, you only need to provide the code and define a trigger. The Lambda function can run when it is needed, whether it is once per week or hundreds of times per second. You only pay for what you use.

This lab demonstrates how to trigger a Lambda function when a file is uploaded to Amazon Simple Storage Service (Amazon S3). The file will be loaded into an Amazon DynamoDB table. The data will be available for you to view on a dashboard page that retrieves the data directly from DynamoDB. This solution *does not use Amazon Elastic Compute Cloud (Amazon EC2)*. It is a *serverless solution* that *automatically scales when it is used*. It also incurs *little cost* when it is in use. When it is idle, there is *practically no cost* because will you only be billed for data storage.

After completing this lab, you should be able to:

* Implement a serverless architecture on AWS
* Trigger Lambda functions from Amazon S3 and Amazon DynamoDB
* Configure Amazon Simple Notification Service (Amazon SNS) to send notifications

At the **end** of this lab, your architecture will look like the following example:

**Duration**

This lab takes approximately **40 minutes** to complete.

**AWS service restrictions**

In this lab environment, access to AWS services and service actions might be restricted to the ones that are needed to complete the lab instructions. You might encounter errors if you attempt to access other services or perform actions beyond the ones that are described in this lab.

**Accessing the AWS Management Console**

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

A **Start Lab** panel opens, and it displays the lab status.

**Tip**: If you need more time to complete the lab, restart the timer for the environment by choosing the Start Lab button again.

1. Wait until the **Start Lab** panel displays the message *Lab status: ready*, then close the panel by choosing the **X**.
2. At the top of these instructions, choose AWS.

This action opens the AWS Management Console in a new browser tab. The system automatically logs you in.

**Tip**: If a new browser tab does not open, a banner or icon is usually at the top of your browser with the message that your browser is preventing the site from opening pop-up windows. Choose the banner or icon, and then choose **Allow pop-ups**.

1. Arrange the **AWS Management Console** tab so that it displays alongside these instructions. Ideally, you will have both browser tabs open at the same time so that you can follow the lab steps more easily.

**Do not change the Region unless specifically instructed to do so**.

**Task 1: Creating a Lambda function to load data**

In this task, you will create *a Lambda function* that will process an inventory file. The Lambda function will read the file and insert information into a DynamoDB table.

1. In the **AWS Management Console**, on the **Services** menu, choose **Lambda**.
2. Choose **Create function**

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

1. Configure the following settings:
   * **Function name:** Load-Inventory
   * **Runtime:** *Python 3.7*
   * Expand  **Choose or create an execution role**.
   * **Execution role:** *Use an existing role*
   * **Existing role:** *Lambda-Load-Inventory-Role*

This role gives the Lambda function permissions so that it can access Amazon S3 and DynamoDB.

1. Choose **Create function**
2. Scroll down to the **Code source** section, and in the **Environment** pane, choose lambda\_function.py.
3. In the code editor, delete all the code.
4. In the **Code source** editor, copy and paste the following code:



# Load-Inventory 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 a DynamoDB table.

import json, urllib, boto3, csv

# Connect to S3 and DynamoDB

s3 = boto3.resource('s3')

dynamodb = boto3.resource('dynamodb')

# Connect to the DynamoDB tables

inventoryTable = dynamodb.Table('Inventory');

# This handler is run 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.parse.unquote\_plus(event['Records'][0]['s3']['object']['key'])

localFilename = '/tmp/inventory.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 Inventory CSV file

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['store'], row['item'], row['count'])

    try:

       # Insert Store, Item and Count into the Inventory table

      inventoryTable.put\_item(

         Item={

           'Store': row['store'],

           'Item':   row['item'],

           'Count': int(row['count'])})

    except Exception as e:

        print(e)

        print("Unable to insert data into DynamoDB table".format(e))

   # Finished!

  return "%d counts inserted" % rowCount

Examine the code. It performs the following steps:

* + Download the file from Amazon S3 that triggered the event
  + Loop through each line in the file
  + Insert the data into the DynamoDB *Inventory* table

1. Choose **Deploy** to save your changes.

Next, you will configure Amazon S3 to trigger the Lambda function when a file is uploaded.

**Task 2: Configuring an Amazon S3 event**

Stores from around the world provide inventory files to load into the inventory tracking system. Instead of uploading their files via FTP, the stores can upload them directly to Amazon S3. They can upload the files through a webpage, a script, or as part of a program. When a file is received, it triggers the Lambda function. This Lambda function will then load the inventory into a DynamoDB table.

In this task, you will create an S3 bucket and configure it to trigger the Lambda function.

1. On the **Services** menu, choose **S3**.
2. Choose **Create bucket**

Each bucket must have a unique name, so you will add a random number to the bucket name. For example: *inventory-123*

1. For **Bucket name** enter: inventory-<number> (Replace with a random number)
2. Choose **Create bucket**

 You might receive an error that states: *The requested bucket name is not available*. If you get this error, choose the first **Edit** link, change the bucket name, and try again until the bucket name is accepted.

You will now configure the bucket to automatically trigger the Lambda function when a file is uploaded.

1. Choose the name of your *inventory-* bucket.
2. Choose the **Properties** tab.
3. Scroll down to **Event notifications**.

You will configure an event to trigger when an object is created in the S3 bucket.

1. Click **Create event notification** then configure these settings:
   * **Name:** Load-Inventory
   * **Event types:**  *All object create events*
   * **Destination:** *Lambda Function*
   * **Lambda function:** *Load-Inventory*
   * Choose **Save changes**

When an object is created in the bucket, this configuration tells Amazon S3 to trigger the *Load-Inventory* Lambda function that you created earlier.

Your bucket is now ready to receive inventory files!

**Task 3: Testing the loading process**

You are now ready to test the loading process. You will upload an inventory file, then check that it loaded successfully.

1. Download the inventory files by opening the context (right-click) menu for these links:

[inventory-berlin.csv](https://labs.vocareum.com/web/2153179/1120197.0/ASNLIB/public/scripts/inventory-berlin.csv)

[inventory-calcutta.csv](https://labs.vocareum.com/web/2153179/1120197.0/ASNLIB/public/scripts/inventory-calcutta.csv)

[inventory-karachi.csv](https://labs.vocareum.com/web/2153179/1120197.0/ASNLIB/public/scripts/inventory-karachi.csv)

[inventory-pusan.csv](https://labs.vocareum.com/web/2153179/1120197.0/ASNLIB/public/scripts/inventory-pusan.csv)

[inventory-shanghai.csv](https://labs.vocareum.com/web/2153179/1120197.0/ASNLIB/public/scripts/inventory-shanghai.csv)

[inventory-springfield.csv](https://labs.vocareum.com/web/2153179/1120197.0/ASNLIB/public/scripts/inventory-springfield.csv)

These files are the inventory files that you can use to test the system. They are comma-separated values (CSV) files. The following example shows the contents of the Berlin file:



store,item,count

Berlin,Echo Dot,12

Berlin,Echo (2nd Gen),19

Berlin,Echo Show,18

Berlin,Echo Plus,0

Berlin,Echo Look,10

Berlin,Amazon Tap,15

1. In the console, return to your S3 bucket by choosing the **Objects** tab.
2. Choose **Upload**
3. Choose **Add files**, and select one of the inventory CSV files. (You can choose any inventory file.)
4. Choose **Upload**

Amazon S3 will automatically trigger the Lambda function, which will load the data into a DynamoDB table.

A serverless Dashboard application has been provided for you to view the results.

1. At the top of these instructions, choose the **Details** button, and to the right of **AWS**, choose the **Show** button.
2. From the **Credentials** window, copy the **Dashboard** URL.
3. Open a new web browser tab, paste the URL, and press ENTER.

The dashboard application will open and display the inventory data that you loaded into the bucket. The data is retrieved from DynamoDB, which proves that the upload successfully triggered the Lambda function.

 If the dashboard application does not display any information, ask your instructor to help you diagnose the problem.

The dashboard application is served as a static webpage from Amazon S3. The dashboard authenticates via Amazon Cognito as an *anonymous user*, which provides sufficient permissions for the dashboard to retrieve data from DynamoDB.

You can also view the data directly in the DynamoDB table.

1. On the **Services** menu, choose **DynamoDB**.
2. In the left navigation pane, choose **Tables**.
3. Choose the **Inventory** table.
4. Choose the **Items** tab.

The data from the inventory file will be displayed. It shows the store, item and inventory count.

**Task 4: Configuring notifications**

You want to notify inventory management staff when a store runs out of stock for an item. For this serverless notification functionality, you will use **Amazon SNS**.

Amazon SNS is a flexible, fully managed publish/subscribe messaging and mobile notifications service. It delivers messages to subscribing endpoints and clients. With Amazon SNS, you can fan out messages to a large number of subscribers, including distributed systems and services, and mobile devices.

1. On the **Services** menu, choose **Simple Notification Service**.
2. In the **Create topic** box, for **Topic name**, enter: NoStock
3. Choose **Next step**
4. Scroll to the bottom of the page.
5. Choose **Create topic**

To receive notifications, you must *subscribe* to the topic. You can choose to receive notifications via several methods, such as SMS and email.

1. In the lower half of the page, choose **Create subscription** and configure these settings:
   * **Protocol:** *Email*
   * **Endpoint:** Enter your email address
   * Choose **Create subscription**

 After you create an email subscription, you will receive a confirmation email message. Open the message and choose the *Confirm subscription* link.

Any message that is sent to the SNS topic will be forwarded to your email.

**Task 5: Creating a Lambda function to send notifications**

You could modify the existing *Load-Inventory* Lambda function to check inventory levels while the file is being loaded. However, this configuration is not a good architectural practice. Instead of overloading the *Load-Inventory* function with business logic, you will create another Lambda function that is triggered when data is loaded into the DynamoDB table. This function will be triggered by a *DynamoDB stream*.

This architectural approach offers several benefits:

* Each Lambda function performs a single, specific function. This practice makes the code simpler and more maintainable.
* Additional business logic can be added by creating additional Lambda functions. Each function operates independently, so existing functionality is not impacted.

In this task, you will create another Lambda function that looks at inventory while it is loaded into the DynamoDB table. If the Lambda function notices that an item is out of stock, it will send a notification through the SNS topic you created earlier.

1. On the **Services** menu, choose **Lambda**.
2. Choose **Create function** and configure these settings:
   * **Function name:** Check-Stock
   * **Runtime:** *Python 3.7*
   * Expand  **Choose or create an execution role**.
   * **Execution role:** *Use an existing role*
   * **Existing role:** *Lambda-Check-Stock-Role*
   * Choose **Create function**

This role was configured with permissions to send a notification to Amazon SNS.

1. Scroll down to the **Code source** section, and in the **Environment** pane, choose lambda\_function.py.
2. In the code editor, delete all the code.
3. Copy the following code, and in the **Code Source** editor, paste the copied code:



# Stock Check Lambda function

#

# This function is triggered when values are inserted into the Inventory DynamoDB table.

# Inventory counts are checked and if an item is out of stock, a notification is sent to an SNS Topic.

import json, boto3

# This handler is run 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 inventory item added, check if the count is zero

 for record in event['Records']:

  newImage = record['dynamodb'].get('NewImage', None)

   if newImage:

    count = int(record['dynamodb']['NewImage']['Count']['N'])

     if count == 0:

      store = record['dynamodb']['NewImage']['Store']['S']

      item  = record['dynamodb']['NewImage']['Item']['S']

       # Construct message to be sent

      message = store + ' is out of stock of ' + item

      print(message)

       # Connect to SNS

      sns = boto3.client('sns')

      alertTopic = 'NoStock'

      snsTopicArn = [t['TopicArn'] for t in sns.list\_topics()['Topics']

                       if t['TopicArn'].lower().endswith(':' + alertTopic.lower())][0]

       # Send message to SNS

      sns.publish(

         TopicArn=snsTopicArn,

         Message=message,

         Subject='Inventory Alert!',

         MessageStructure='raw'

      )

 # Finished!

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

Examine the code. It performs the following steps:

* + Loop through the incoming records
  + If the inventory count is zero, send a message to the *NoStock* SNS topic

You will now configure the function so it triggers when data is added to the *Inventory* table in DynamoDB.

1. Choose **Deploy** to save your code changes
2. Scroll to the **Designer** section (which is at the top of the page).
3. Choose **Add trigger** and then configure these settings:
   * **Select a trigger:** *DynamoDB*
   * **DynamoDB Table:** *Inventory*
   * Choose **Add**

You are now ready to test the system!

**Task 6: Testing the System**

You will now upload an inventory file to Amazon S3, which will trigger the original *Load-Inventory* function. This function will load data into DynamoDB, which will then trigger the new *Check-Stock* Lambda function. If the Lambda function detects an item with zero inventory, it will send a message to Amazon SNS. Then, Amazon SNS will notify you through SMS or email.

1. On the **Services** menu, choose **S3**.
2. Choose the name of your *inventory-* bucket.
3. Choose **Upload** and upload a different inventory file.
4. Return to the **Inventory System Dashboard** and refresh  the page.

You should now be able to use the **Store** menu to view the inventory from both stores.

Also, you should receive a *notification through SMS or email* that the store has an out-of-stock item (each inventory file has one item that is out of stock).

 If you did not receive a notification, wait a few minutes and upload a different inventory file. The DynamoDB trigger can sometimes take a few minutes to enable.

1. Try to upload multiple inventory files at the same time. What do you think will happen?

**Submitting your work**

1. At the top of these instructions, choose **Submit** to record your progress and when prompted, choose **Yes**.
2. If the results don't display after a couple of minutes, return to the top of these instructions and choose Grades

**Tip**: You can submit your work multiple times. After you change your work, choose **Submit** again. Your last submission is what will be recorded for this lab.

1. To find detailed feedback on your work, choose Details followed by  **View Submission Report**.

**Lab complete**

 Congratulations! You have completed the lab.

1. Choose End Lab at the top of this page, and then select **Yes** to confirm that you want to end the lab.

A panel indicates that *DELETE has been initiated... You may close this message box now.*

1. Select the **X** in the top right corner to close the panel.