



COS20019

Cloud Computing Architecture

Week 9 – ACA Module 14

Guided Lab: Implementing a Serverless Architecture on AWS

Truong Ngoc Gia Hieu
105565520

Guided Lab: Implementing a Serverless Architecture on AWS

A. Lab overview and objectives

Traditionally, applications run on servers. These servers can be physical or 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 need to provide only the code and define a trigger. The Lambda function can run when it is needed whether it is once a week or hundreds of times each second. You pay for only what you use.

This lab demonstrates how to invoke 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 you will be billed for data storage only.

After completing this lab, you should be able to do the following:

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

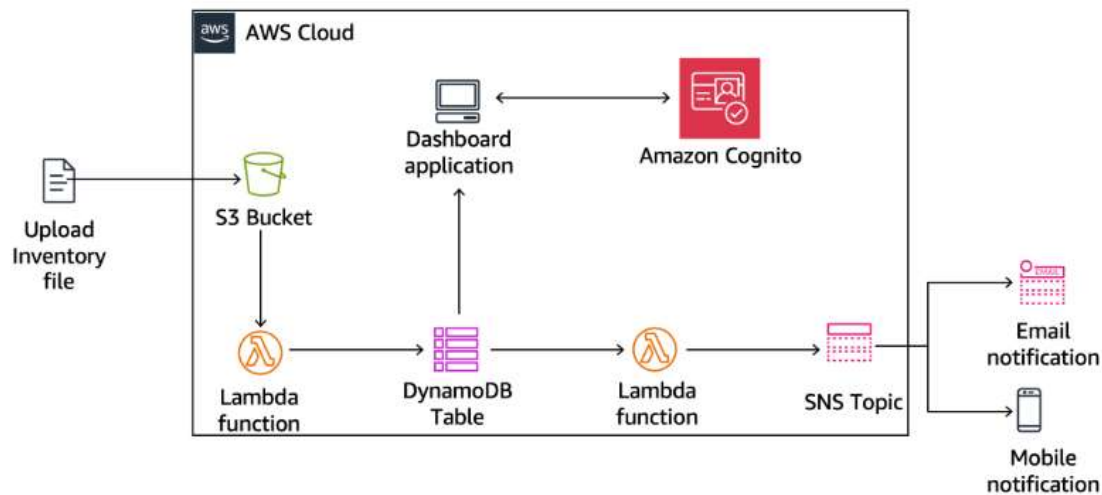
Scenario

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 upload an inventory file to an S3 bucket, which initiates the following sequence of events:

- Invoke a Lambda function that reads the file and inserts items into a DynamoDB table.
- Another Lambda function receives updates from the DynamoDB table, which then sends a message to an SNS topic when an inventory item is out of stock.
- Amazon SNS then sends a notification through short message service (SMS) or email that requests additional inventory.
- A serverless, web-based dashboard application that uses Amazon Cognito to authenticate to Amazon Web Services (AWS) gains access to the DynamoDB table to display inventory levels.

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



Duration

This lab requires 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.

B. Access AWS Management Console



Figure 1: AWS Activated

C. Task 1: Creating a Lambda function to load data

Step 1.1: Search Lambda service and choose Create function button. Configure following settings:

- For **Function name**, enter Load-Inventory.
- For **Runtime**, choose **Python 3.8**.
- Expand **Change default execution role**, and configure the following options:
 - For **Execution role**, choose **Use an existing role**.
 - For **Existing role**, choose **Lambda-Load-Inventory-Role**.

This role gives the Lambda function permission to access Amazon S3 and DynamoDB.

The screenshot shows the 'Create function' page in the AWS Lambda console. At the top, there are three tabs: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below the tabs, the 'Basic information' section is expanded, showing the following settings:

- Function name:** 'Load-Inventory' (with a note that the name must be 1 to 64 characters, must be unique in the Region, and can't include spaces, hyphens, or certain characters).
- Runtime:** 'Python 3.8' (with a note that the runtime must be one of the supported runtimes).
- Architecture:** 'x86_64' (with a note that the architecture must be one of the supported architectures).
- Permissions:** 'Change default execution role' is expanded, showing three options: 'Create a new role with basic Lambda permissions', 'Use an existing role' (selected), and 'Create a new role from AWS policy templates'. Under 'Existing role', 'Lambda-Load-Inventory-Role' is selected.

At the bottom, there is an 'Additional configurations' section with a note about setting up networking, security, and governance. The 'Create function' button is visible at the bottom right.

Figure 2: Function configuration

Step 1.2: Then, in the Source code editor, copy the code, paste to the file `lambda_function.py` file, Save and Deploy

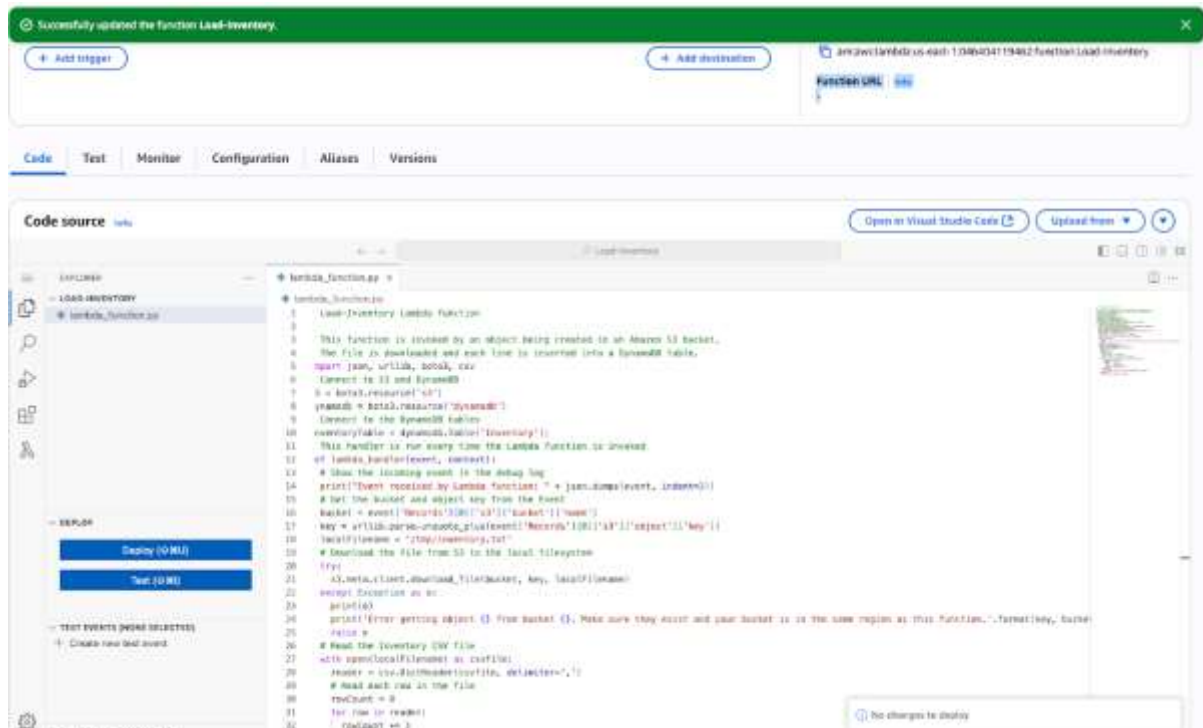


Figure 3: Deploy successfully

D. Task 2: Configuring an Amazon S3 event

Step 2.1: On the AWS Management Console, in the search box, enter and choose S3. Next, create bucket with name *inventory-105565520*



Figure 4: Bucket created successfully

Step 2.2: Choose the bucket *inventory-105565520* and select **Properties** tab. In the **Event notifications** section, choose **Create event notification**, and then configure these settings:

- **Event name:** Enter *Load-Inventory*.
- **Event types:** Choose *All object create events*.
- **Destination:** Choose *Lambda function*.
- **Lambda function:** Choose *Load-Inventory*.



Figure 5: Event notification configured successfully

E. Task 3: Testing the loading process

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

Step 3.1: Download the inventory files by opening (right-clicking) the context menu for these links:

- [inventory-berlin.csv](#)
- [inventory-calcutta.csv](#)
- [inventory-karachi.csv](#)
- [inventory-pusan.csv](#)
- [inventory-shanghai.csv](#)
- [inventory-springfield.csv](#)

Step 3.2: Upload one of the inventory files above to the S3 bucket



Figure 6: Upload files successfully

Step 3.3: Choose AWS Details and copy the Dashboard URL into a new web browser tab.

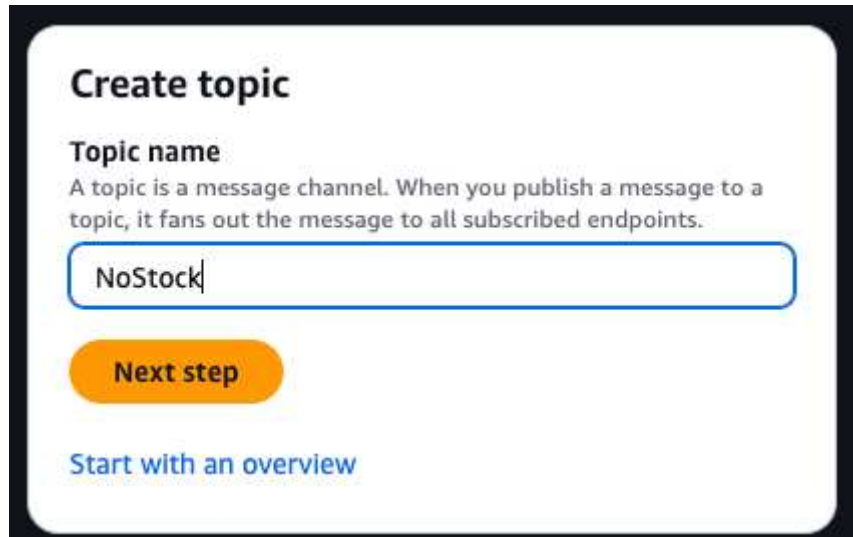


Figure 7: URL Display successfully

F. 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 use Amazon SNS.

Step 4.1: On the AWS Management Console, in the search box, enter and choose SNS. In the **Create topic** section, for **Topic name**, enter NoStock. Choose **Next step**.



Create topic

Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

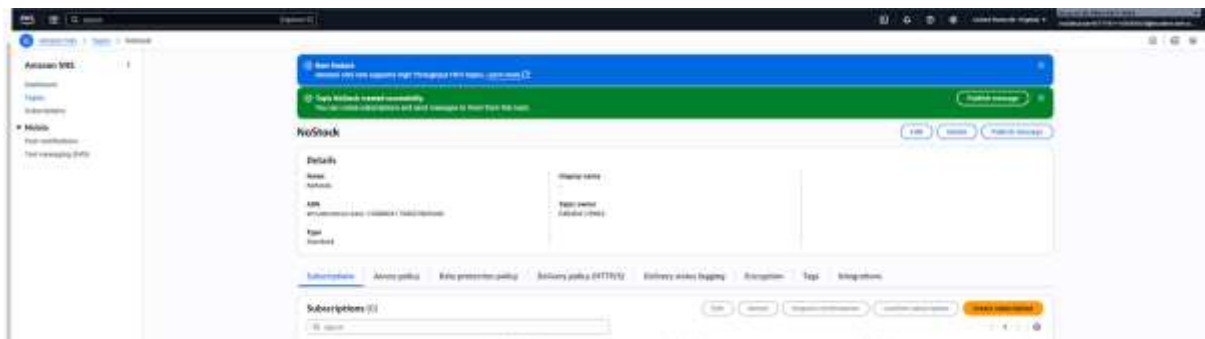
NoStock

Next step

[Start with an overview](#)

Figure 8: Topic name configuration

Step 4.2: On the Create topic page, keep **Standard** selected. Choose **Create topic**.



NoStock

Details

Name: NoStock

ARN: arn:aws:sns:us-east-1:123456789012:NoStock

Topic owner: 123456789012

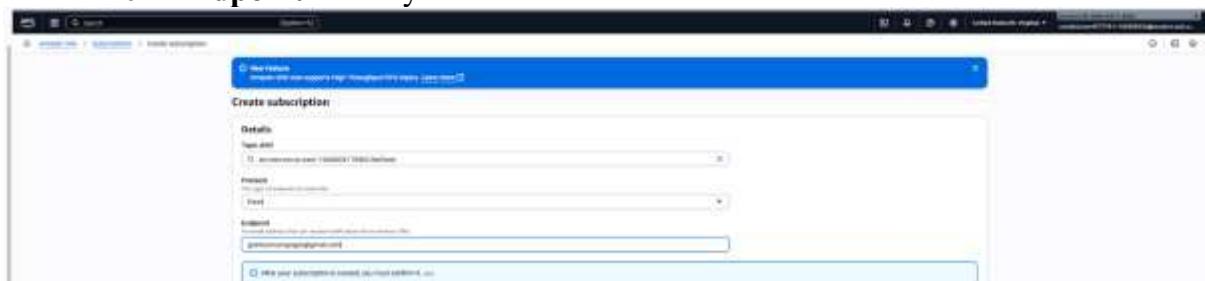
Topic created: 1/1/2020

Subscriptions (0)

Figure 9: Topic created successfully

Step 4.3: On the **NoStock** topic page, in the Subscriptions section, choose **Create subscription**. On the **Create subscription** page, configure these settings:

- **Protocol:** Choose **Email**.
- **Endpoint:** Enter your email address.



Create subscription

Details

Name: NoStock

Protocol: Email

Endpoint: john.doe@example.com

Figure 10: Subscription configuration

Step 4.4: To confirm your subscription, open the email message, and choose the **Confirm subscription** link.

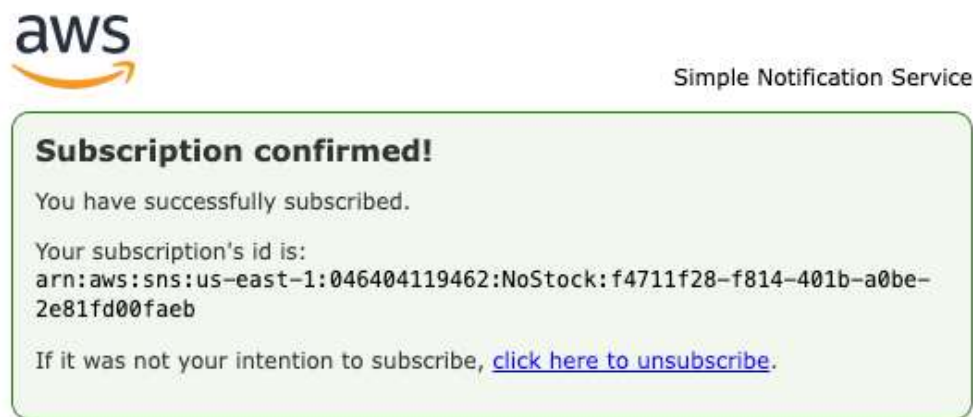


Figure 11: Subscription confirmation

G. 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 create another Lambda function that is invoked when data is loaded into the DynamoDB table. A DynamoDB stream invokes this function.

This architectural approach offers several benefits:

- Each Lambda function performs a single, specific function. This practice makes the code clearer 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 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 sends a notification through the SNS topic that you created earlier.

Step 5.1: On the AWS Management Console, in the search box, enter and choose **Lambda**. Choose **Create function** and configure these settings:

- For **Function name**, enter **Check-Stock**.
- For **Runtime**, choose **Python 3.8**.
- Expand **Change default execution role**, and configure the following options:
 - For **Execution role**, choose **Use an existing role**.
 - For **Existing role**, choose **Lambda-Check-Stock-Role**.

Create function [info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**
Start with a simple Python 3.x or Node.js example.

☐ **Use a blueprint**
Build a Lambda application from sample code and configurations provided for common use cases.

☐ **Container image**
Select a container image for deploying Docker-based functions.

Basic information

Function name
Enter a name that describes the purpose of your function.

Function names must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, 0-9, hyphen (-), and underscore (_).

Runtime [info](#)
Choose the language to use to write your function. Note that the runtime code editor supports only Node.js, Python, and Ruby.

Architecture [info](#)
Choose the instruction set architecture you want for your function code.
☒ **x86_64**

Permissions [info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions

☒ **Use an existing role**

☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

View the Lambda-Check-Stock-Role role on the IAM console

► **Additional configurations**
Use additional configurations to set up reentrancy, security, and governance for your function. These settings help secure and customize your Lambda function deployment.

Figure 12: Function confirmation

Step 5.2: In the Code source section, in the Environment pane, choose **lambda_function.py** and configure code:

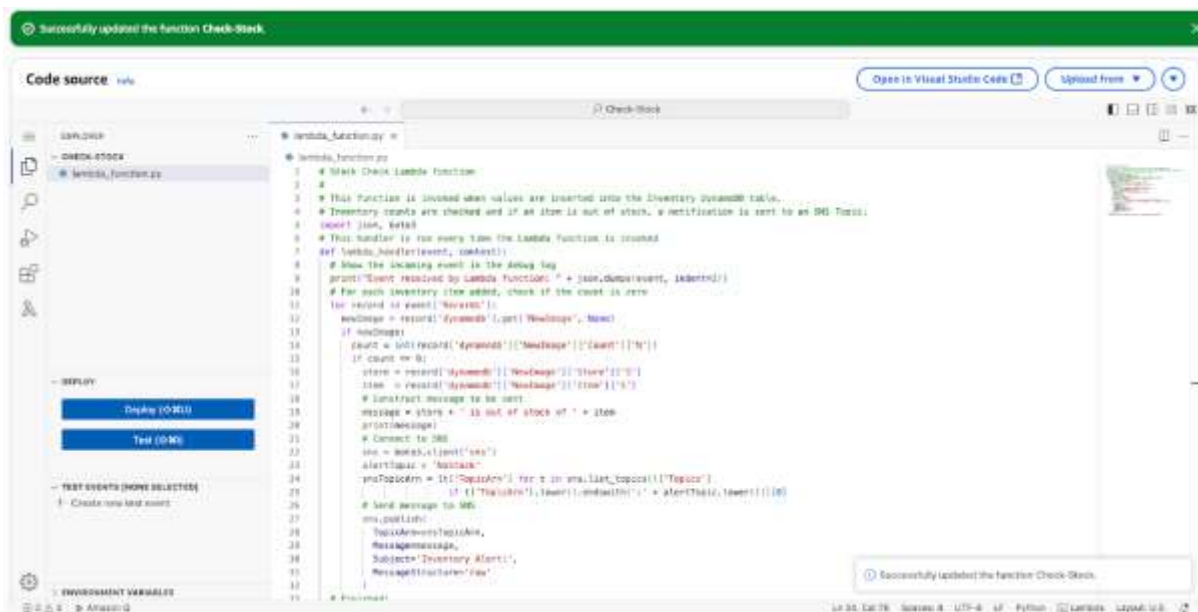


Figure 13: Code confirmation

Step 5.3: In the Function overview section, choose **Add trigger**, and configure these settings:

- **Select a source:** Choose **DynamoDB**.
- **DynamoDB table:** Choose **Inventory**.

Add trigger

Trigger configuration [info](#)

DynamoDB
aws::dynamodb::event-source-mapping::trigger::polling

DynamoDB table
Choose or enter the ARN of a DynamoDB table.

Event poller configuration

☒ **Activate trigger**
Select to activate the trigger now. Being prechecked for create the trigger in a distributed state for testing development.

☐ **Enable metrics**
Monitor your event source with metrics. You can view these metrics on CloudWatch console. Enabling this feature incurs additional costs. [Learn more](#)

Batch size [info](#)
The maximum number of records to fetch in each batch to send to the function.

Starting position [info](#)
The position in the stream to start reading from.

Batch window - optional
The maximum amount of time to gather records before invoking the function, in seconds.

Additional settings

In order to read from the DynamoDB trigger, your execution role must have proper permissions.

Figure 14: Trigger configuration

H. Task 6: Testing the system

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

Step 6.1: Choose bucket *inventory-105565520* and upload different inventory file.

Upload successful
For more information, see the Files and folders table.

Upload: status [close](#)

After you navigate away from this page, the following information is no longer available.

Summary

Destination: [s3://inventory-105565520](#)

Succeeded: [4 Files, 598.0 B \(100.00%\)](#)

Failed: [0 Files, 0 B \(0%\)](#)

Files and folders [Configuration](#)

Files and folders (4 total, 598.0 B)

Name	Folder	Type	Size	Status	Error
inventory-kacatu.csv	-	text/csv	145.0 B	Succeeded	-
inventory-pouan.csv	-	text/csv	153.0 B	Succeeded	-
inventory-changthai.csv	-	text/csv	152.0 B	Succeeded	-
inventory-springfield.csv	-	text/csv	148.0 B	Succeeded	-

Figure 15: Upload successful

Step 6.2: Test to upload multiple inventory files at the same time.

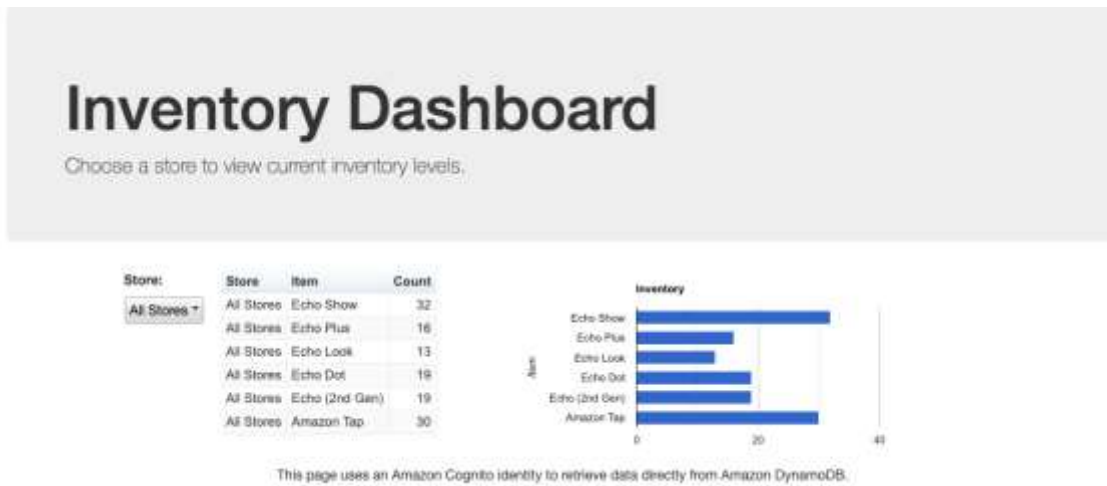


Figure 16: Test to upload

I. Task Completed

Total score	40/40
[Task 1A] Load-Inventory Function Created	5/5
[Task 1B] Load-Inventory Execution Role	5/5
[Task 2A] Bucket Created	5/5
[Task 2B] Bucket Event	5/5
[Task 3] Database Table Data	5/5
[Task 4] SNS Topic Created	5/5
[Task 5A] Check-Stock Function Created	5/5
[Task 5B] Check-Stock has Trigger	5/5

Figure 17: Task completed

J. Conclusions

I successfully implemented a fully serverless inventory tracking and alerting system on AWS, achieving a robust, scalable, and cost-effective solution without relying on traditional server infrastructure like Amazon EC2. The core architecture utilized an Amazon S3 event trigger to invoke the Load-Inventory Lambda function, which processed data into a DynamoDB table. By using DynamoDB Streams to activate a separate Check-Stock Lambda function, I maintained clear architectural separation for business logic, ensuring items with zero inventory triggered notifications via Amazon SNS. This complete,

event-driven workflow was successfully validated through the dashboard and the receipt of email alerts, confirming my practical understanding of orchestrating decoupled, highly available cloud services.