

# Project 2: Implementing a Serverless Architecture on AWS with AWS Lambda

## Scenario

I am creating an inventory tracking system. It stores from around the world will upload an inventory file to Amazon S3. I want to be able to view the inventory levels and send a notification when inventory levels are low.

In this project:

- I 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.

## Overview

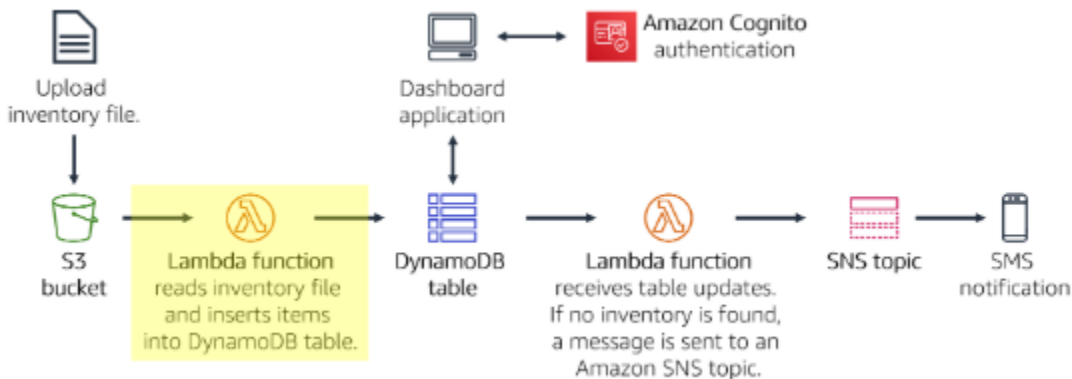
Traditionally, applications run on servers. These servers can be physical (or bare metal). They can also be virtual environments that run on top of physical servers. However, you must purchase and provision all these types of servers, and you must also manage their capacity. In contrast, you can run 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.

Note: Created Required Roles to perform this project, we can see those screenshots at the end of the document.

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

In this task, I 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.

3. Configure the following settings:

- \* Function name: Load-Inventory
- \* Runtime: Python 3.12
- \* Expand Change default 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.

aws [Alt+S] United States (N. Virginia) voclabs/user3432748=Srinadh\_Reddy\_Kamireddy @ 4511-2394-2857

Lambda > Functions > Create function

### Create function [Info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

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

☐ **Container image**  
Select a container image to deploy for your function.

---

#### Basic information

**Function name**  
Enter a name that describes the purpose of your function.  
Load-Inventory  
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
Python 3.12

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**Permissions** [Info](#)

▼ **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.  
 Lambda-Load-Inventory-Role  
[View the Lambda-Load-Inventory-Role role](#) on the IAM console.

► **Additional Configurations**  
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#) [Create function](#)

4. Choose Create function

5. Scroll down to the Code source section, and in the Environment pane, choose lambda\_function.py.

Lambda > Functions > Load-Inventory

☑ Successfully created the function Load-Inventory. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

### Load-Inventory

[Throttle](#) [Copy ARN](#) [Actions](#)

[Export to Infrastructure Composer](#) [Download](#)

#### ▼ Function overview [Info](#)

[Diagram](#) [Template](#)

**Load-Inventory**

Layers (0)

[+ Add trigger](#) [+ Add destination](#)

**Description**  
-

**Last modified**  
22 seconds ago

**Function ARN**  
[arn:aws:lambda:us-east-1:451123942857:function:Load-Inventory](#)

**Function URL** [Info](#)  
-

**Code** | **Test** | **Monitor** | **Configuration** | **Aliases** | **Versions**

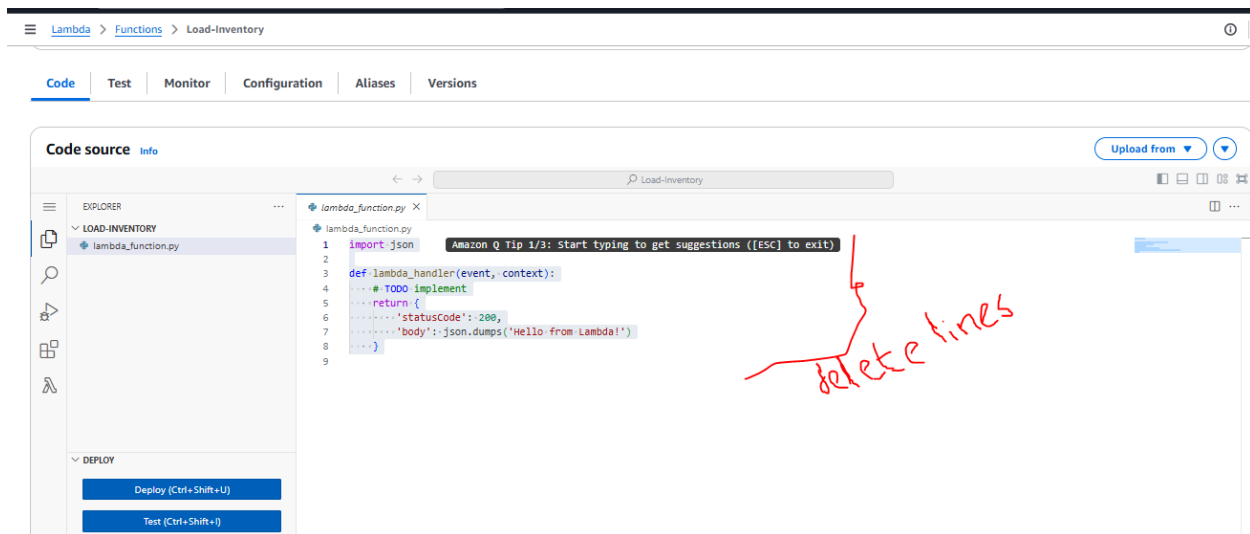
#### Code source [Info](#)

[Upload from](#)

EXPLORER

- ▼ LOAD-INVENTORY
  - lambda\_function.py

6. In the code editor, delete all the code.



7. 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
```

```

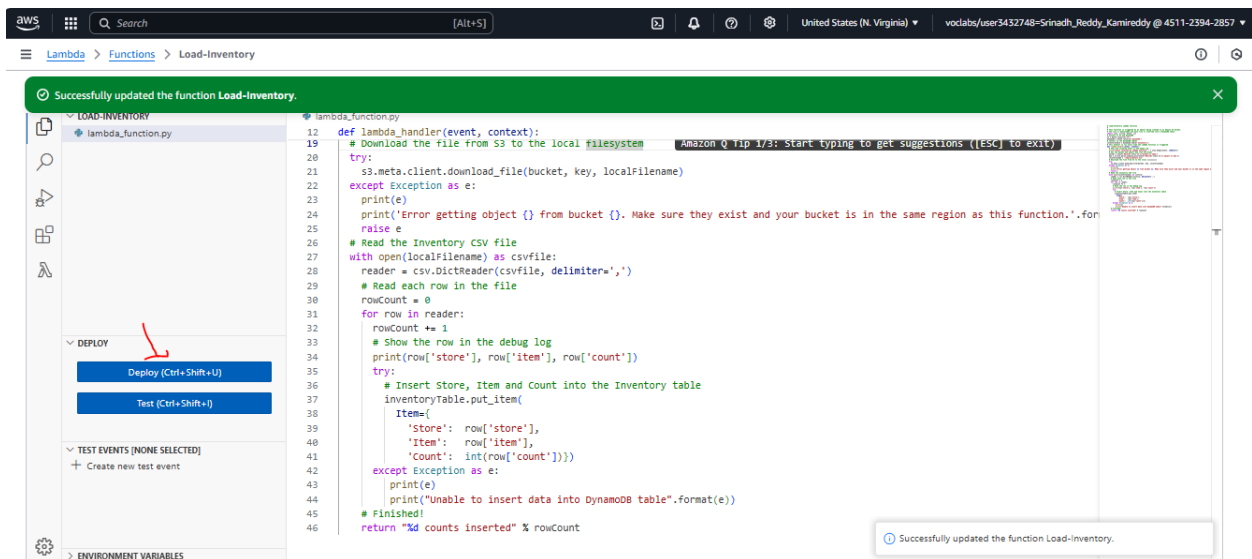
# Read the Inventory CSV file
with open(localFilename) as csvfile:
    reader = csv.DictReader(csvfile, delimiter=',')
    # Read each row in the file
    rowCount = 0
    for row in reader:
        rowCount += 1
    # Show the row in the debug log
    print(row['store'], row['item'], row['count'])
try:
    # 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 above 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

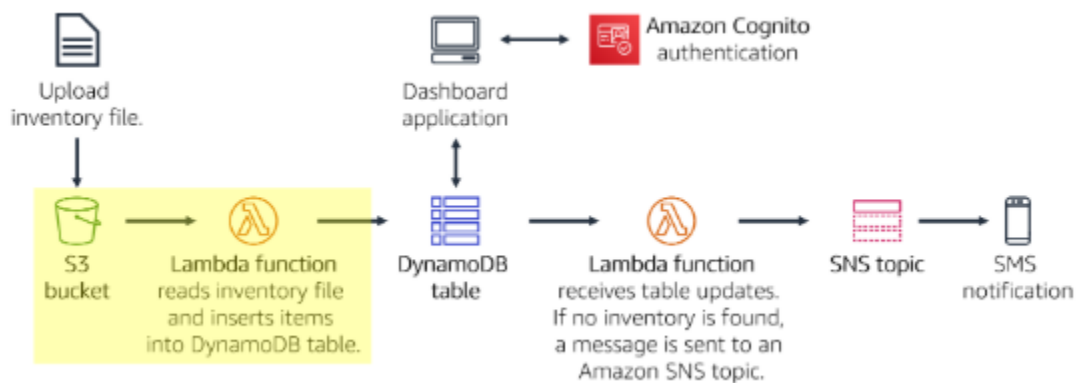
8. Choose Deploy to save your changes.



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

## Task 2: Configuring an Amazon S3 event

It 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, I will create an S3 bucket and configure it to trigger the Lambda function.

9. On the Services menu, choose S3.

10. Choose Create bucket

Each bucket must have a unique name, so I will add a random number to the bucket name. For example: inventory-6569

11. For Bucket name enter: inventory-<number> (Replace with a random number)

12. Choose Create bucket at the bottom

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.

## Create bucket [Info](#)

Buckets are containers for data stored in S3.

### General configuration

#### AWS Region

US East (N. Virginia) us-east-1

#### Bucket type [Info](#)

##### ☒ General purpose

Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

##### ☐ Directory

Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

#### Bucket name [Info](#)

inventory-6569

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#)

#### Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

##### [Choose bucket](#)

Format: s3://bucket/prefix

### Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

##### ☒ ACLs disabled (recommended)

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

##### ☐ ACLs enabled

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

### Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

#### ☒ Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

##### ☒ Block public access to buckets and objects granted through new access control lists (ACLs)

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

##### ☒ Block public access to buckets and objects granted through any access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

##### ☒ Block public access to buckets and objects granted through new public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

##### ☒ Block public and cross-account access to buckets and objects through any public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

### Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

#### Bucket Versioning

##### ☒ Disable

##### ☐ Enable

### Tags - optional (0)

You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

No tags associated with this bucket.

No tags associated with this bucket.

##### [Add tag](#)

### Default encryption [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

#### Encryption type [Info](#)

##### ☒ Server-side encryption with Amazon S3 managed keys (SSE-S3)

##### ☐ Server-side encryption with AWS Key Management Service keys (SSE-KMS)

##### ☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

Secure your objects with two separate layers of encryption. For details on pricing, see [DSSE-KMS pricing](#) on the [Storage](#) tab of the [Amazon S3 pricing page](#).

#### Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

##### ☐ Disable

##### ☒ Enable

### Advanced settings

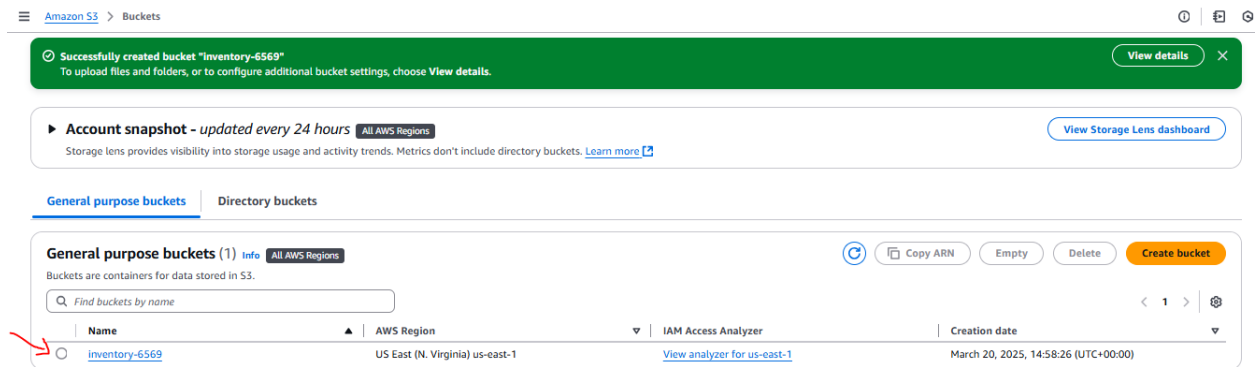
After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

[Cancel](#)

[Create bucket](#)

Now I configure the bucket to automatically trigger the Lambda function when a file is uploaded.

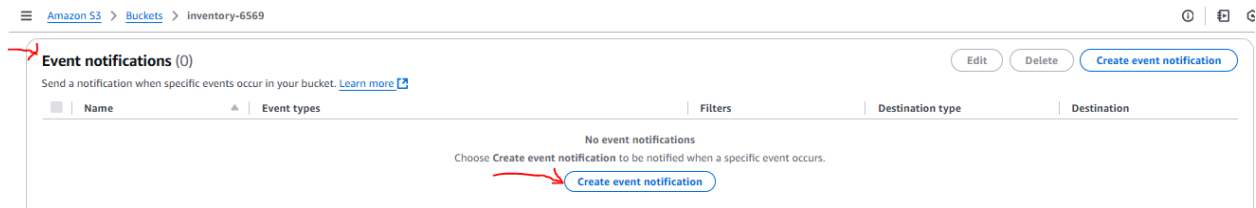
13. Choose the name of your inventory- bucket and click on bucket name inventory-6569



14. Choose the Properties tab.

15. Scroll down to Event notifications.

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



16. 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



aws Search [Alt+S] United States (N. Virginia) vocabs/user3432748-Srinadh\_Reddy\_Kamireddy @ 4511-2394-2857

Amazon S3 > Buckets > Inventory-6569 > Create event notification

### Create event notification info

To enable notifications, you must first add a notification configuration that identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications.

#### General configuration

**Event name**  
Load-Inventory  
Event name can contain up to 255 characters.

**Prefix - optional**  
Limit the notifications to objects with key starting with specified characters.  
Images/

**Suffix - optional**  
Limit the notifications to objects with key ending with specified characters.  
.jpg

#### Event types

Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.

**Object creation**

☒ All object create events  
s3:ObjectCreated:\*

☐ Put  
s3:ObjectCreated:Put

☐ Post  
s3:ObjectCreated:Post

#### Destination

Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)

**Destination**  
Choose a destination to publish the event. [Learn more](#)

☒ **Lambda function**  
Run a Lambda function script based on S3 events.

☐ **SNS topic**  
Fanout messages to systems for parallel processing or directly to people.

☐ **SQS queue**  
Send notifications to an SQS queue to be read by a server.

**Specify Lambda function**

☒ Choose from your Lambda functions

☐ Enter Lambda function ARN

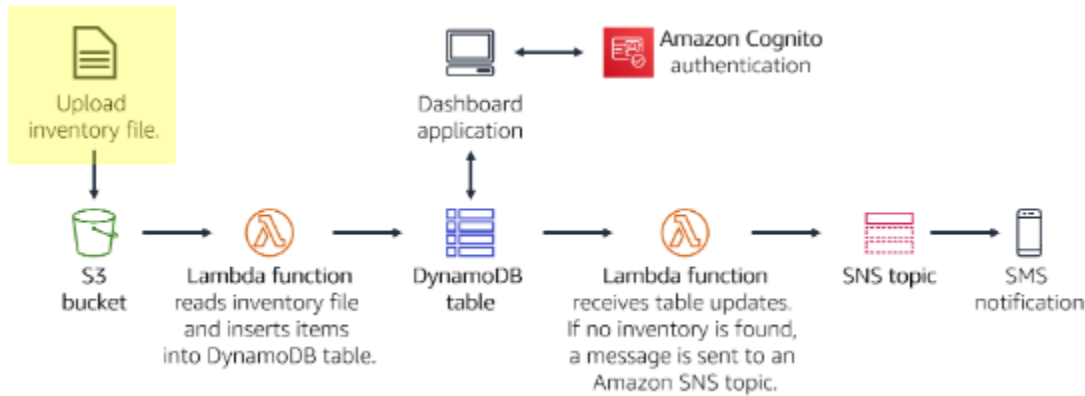
**Lambda function**  
Load-Inventory

[Cancel](#) [Save changes](#)

When an object is created in the bucket, this configuration tells Amazon S3 to trigger the Load-Inventory Lambda function that I created earlier.  
My bucket is now ready to receive inventory files!

### Task 3: Testing the loading process

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



21. Download the inventory files from the attachments.

inventory-berlin.csv

inventory-calcutta.csv

inventory-karachi.csv

inventory-pusan.csv

inventory-shanghai.csv

inventory-springfield.csv

These files are the inventory files that I can use to test the system. They are comma-separated values (CSV) files.

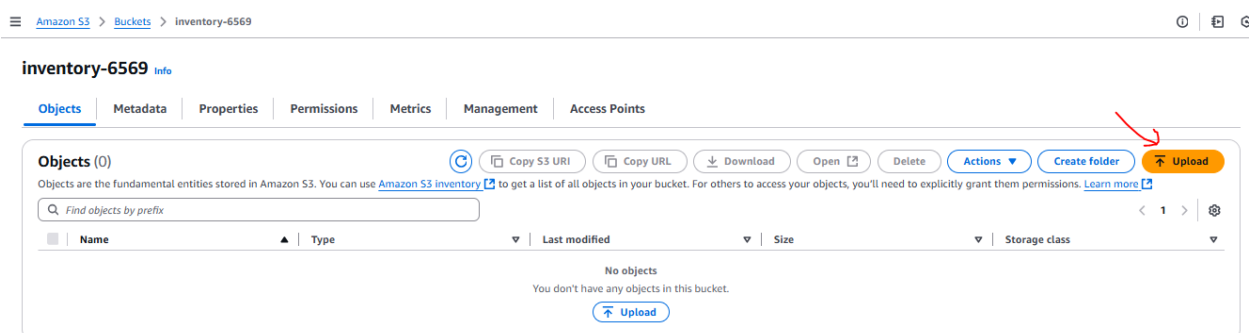
The following example shows the contents of the Berlin file:

```

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
  
```

18. In the console, return to your S3 bucket by choosing the Objects tab.

19. Choose Upload

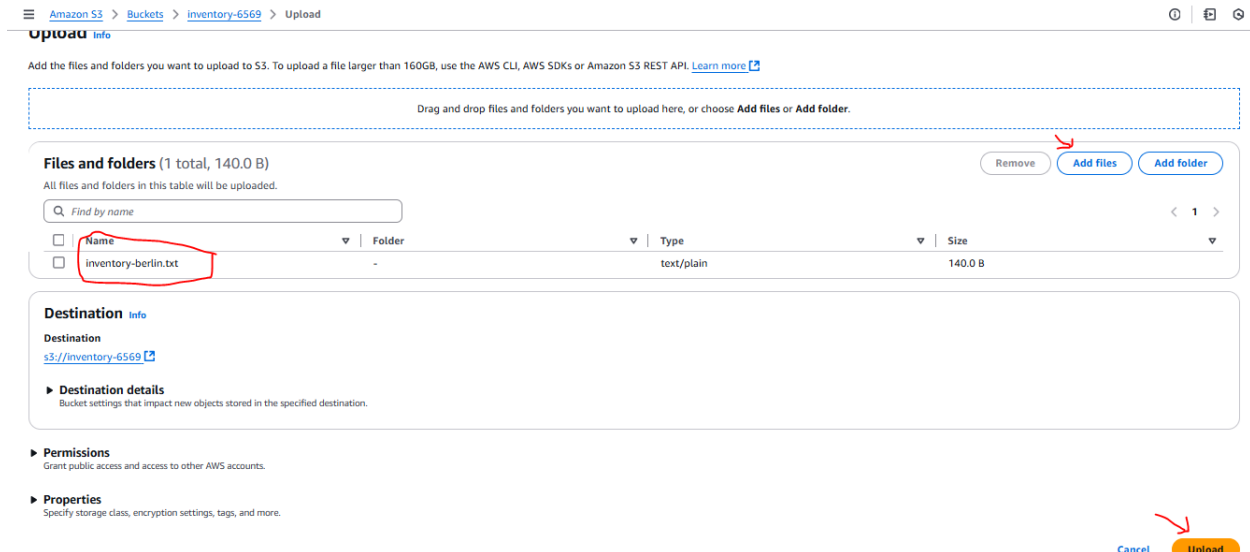


20. Choose Add files, and select one of the inventory CSV files. (we can choose any inventory file.)

21. 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.



23. Copy the below Dashboard URL.

<https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/ILT-TF-200-ACACAD-20-EN/mod13-guided/web/inventory.htm?region=us-east-1&poolId=us-east-1:1e90c542-93f9-422f-95f0-8772ff8fd948>

24. 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.

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.

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

25. On the Services menu, choose DynamoDB.

26. In the left navigation pane, choose Tables.

27. Choose the Inventory table.

28. Choose the Items tab.

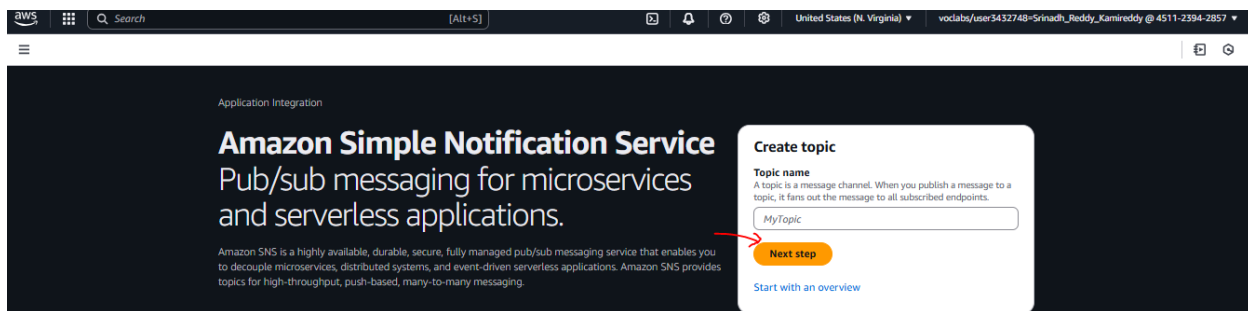
The data from the inventory file will be displayed. It shows the store, item and inventory count.

## Task 4: Configuring notifications

I want to notify inventory management staff when a store runs out of stock for an item. For this serverless notification functionality, I 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, I can fan out messages to a large number of subscribers, including distributed systems and services, and mobile devices.

29. On the Services menu, choose Simple Notification Service. Click on Next step.



30. In the Create topic box.

\* Details: Standard

\* Name: NoStock

\* Display Name: NoStock

Amazon SNS > Topics > Create topic

### Create topic

**Details**  
**Type** | [Info](#)  
Topic type cannot be modified after topic is created

☐ FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

☒ Standard

- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

**Name**  
NoStock  
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

**Display name - optional** | [Info](#)  
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.  
NoStock  
Maximum 100 characters.

## 31. Choose Create topic

Amazon SNS > Topics > Create topic

► **Access policy - optional** [Info](#)  
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

► **Data protection policy - optional** [Info](#)  
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

► **Delivery policy (HTTP/S) - optional** [Info](#)  
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

► **Delivery status logging - optional** [Info](#)  
These settings configure the logging of message delivery status to CloudWatch Logs.

► **Tags - optional**  
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

► **Active tracing - optional** [Info](#)  
Use AWS X-Ray active tracing for this topic to view its traces and service map in Amazon CloudWatch. Additional costs apply.

[Cancel](#) [Create topic](#)

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

Amazon SNS > Topics > NoStock

Topic NoStock created successfully.  
You can create subscriptions and send messages to them from this topic. [Publish message](#)

**NoStock** [Edit](#) [Delete](#) [Publish message](#)

**Details**

<b>Name</b> NoStock	<b>Display name</b> NoStock
<b>ARN</b> arn:aws:sns:us-east-1:451123942857:NoStock	<b>Topic owner</b> 451123942857
<b>Type</b> Standard	

**Subscriptions** | Access policy | Data protection policy | Delivery policy (HTTP/S) | Delivery status logging | Encryption | Tags | Integrations

**Subscriptions (0)** [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

Search

ID	Endpoint	Status	Protocol
No subscriptions found You don't have any subscriptions to this topic.			

[Create subscription](#)

32. In the lower of the page, choose Create subscription and configure these settings:

Protocol: Email

Endpoint: Enter your email address

Choose Create subscription

## Create subscription

## Details

## Topic ARN

## Protocol

The type of endpoint to subscribe

## Endpoint

An email address that can receive notifications from Amazon SNS.

ⓘ After your subscription is created, you must confirm it. [Info](#)

▶ Subscription filter policy - optional [Info](#)

This policy filters the messages that a subscriber receives.

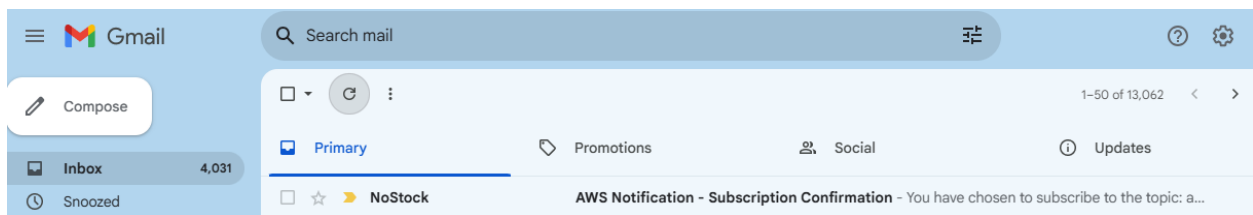
▶ Redrive policy (dead-letter queue) - optional [Info](#)

Send undeliverable messages to a dead-letter queue.

[Cancel](#)[Create subscription](#)

After I create an email subscription, I 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 my email.

AWS Notification - Subscription Confirmation Inbox x

**NoStock** <no-reply@sns.amazonaws.com>  
to me ▾

3:48 PM (0 minutes ago) ☆ 😊

You have chosen to subscribe to the topic:  
**arn:aws:sns:us-east-1:451123942857:NoStock**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

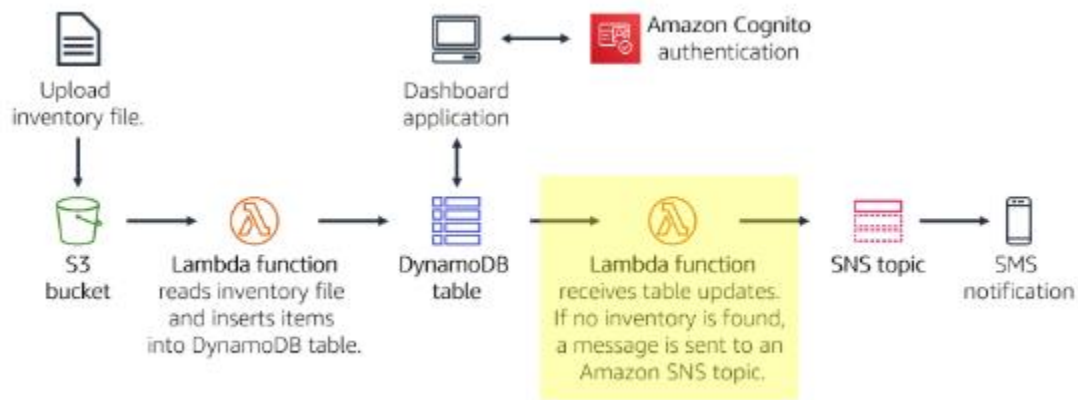
## Task 5: Creating a Lambda function to send notifications

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.



33. On the Services menu, choose Lambda.



34. Choose Create function and configure these settings:

- \* Function name: Check-Stock
- \* Runtime: Python 3.12
- \* Expand Change default 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.

**Create function** [Info](#)

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

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

☐ **Container image**  
Select a container image to deploy for your function.

---

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
  
 Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

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

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**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**

▼ **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 code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#) [Create function](#)

35. Scroll down to the Code source section, and in the Environment pane, choose `lambda_function.py`.

36. In the code editor, delete all the code.

37. 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']
```



```

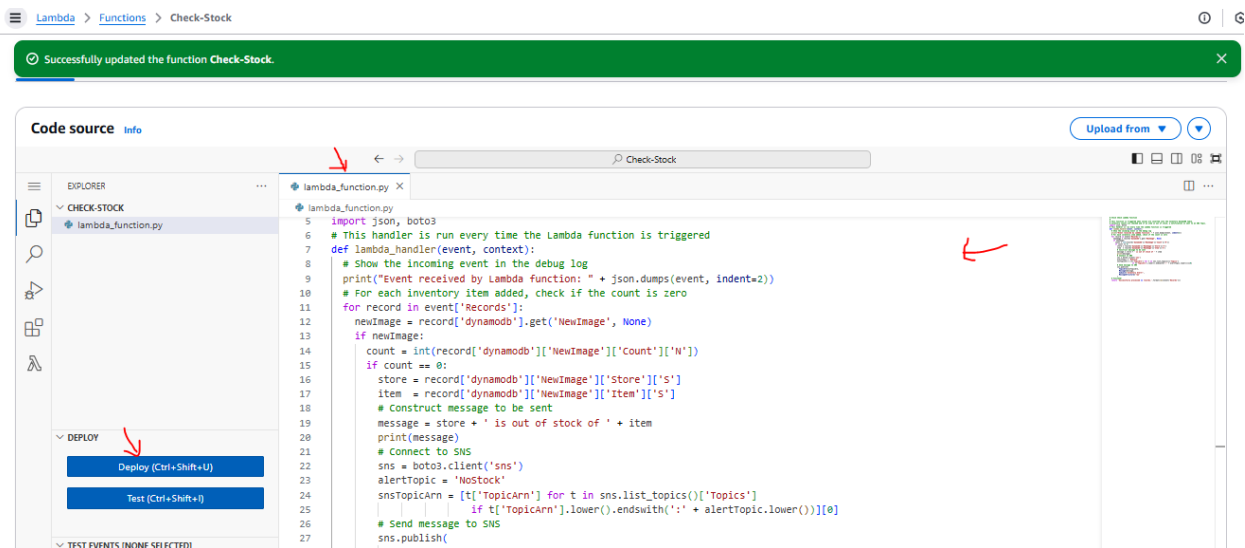
item = record['dynamodb']['NewImage']['Item']['S']
# Construct message to be sent
message = store + ' is out of stock of ' + item
print(message)
# Connect to SNS
sns = boto3.client('sns')
alertTopic = 'NoStock'
snsTopicArn = [t['TopicArn'] for t in sns.list_topics()['Topics']
                if t['TopicArn'].lower().endswith(':' + 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

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



38. Choose Deploy to save your code changes

39. Scroll to the Designer section (which is at the top of the page).

aws Search [Alt+S] United States (N. Virginia) vocabs/user3432748-Srinadh\_Reddy\_Kamireddy @ 4511-2394-2857

Lambda > Functions > Check-Stock

### Check-Stock

Throttle Copy ARN Actions

Export to Infrastructure Composer Download

**Function overview** Info

Diagram Template

+ Add trigger

Check-Stock

Layers (0)

+ Add destination

Description

Last modified 2 minutes ago

Function ARN [arn:aws:lambda:us-east-1:451123942857:function:Check-Stock](#)

Function URL [Info](#)

Code Test Monitor Configuration Aliases Versions

40. Choose Add trigger and then configure these settings:

- \* Select a trigger: DynamoDB
- \* DynamoDB Table: Inventory
- \* Choose Add

Lambda > Add triggers

### Add trigger

Trigger configuration Info

DynamoDB

DynamoDB table

Choose or enter the ARN of a DynamoDB table.

arn:aws:dynamodb:us-east-1:451123942857:table/Inventory

Event source mapping configuration

☒ Activate trigger

Select to activate the trigger now. Keep unchecked to create the trigger in a deactivated state for testing (recommended).

☐ Enable metrics

Monitor your event source with metrics. You can view those metrics in CloudWatch console. Enabling this feature incurs additional costs. [Learn more](#)

Batch size

The number of records in each batch to send to the function.

100

Starting position

The position in the stream to start reading from. For more information, see [ShardIteratorType](#) in the Amazon Kinesis API Reference.

Latest

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.

Cancel Add

Lambda > Functions > Check-Stock

### Check-Stock

Throttle Copy ARN Actions

Export to Infrastructure Composer Download

**Function overview** Info

Diagram Template

DynamoDB

+ Add trigger

Check-Stock

Layers (0)

+ Add destination

Description

Last modified 4 minutes ago

Function ARN [arn:aws:lambda:us-east-1:451123942857:function:Check-Stock](#)

Function URL [Info](#)

Code Test Monitor Configuration Aliases Versions

The trigger Inventory was successfully added to function Check-Stock. The trigger is in a disabled state.

We are now ready to test the system!

## Task 6: Testing the System

I 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.

41. On the Services menu, choose S3.

42. Choose the name of your inventory- bucket.

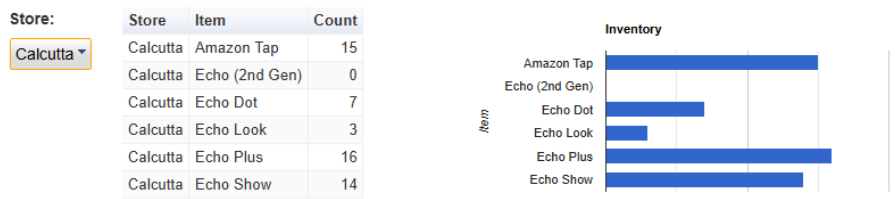
43. Choose Upload and upload a different (Calcutta) inventory file.

44. Return to the Inventory System Dashboard and refresh the page.

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

# Inventory Dashboard

Choose a store to view current inventory levels.

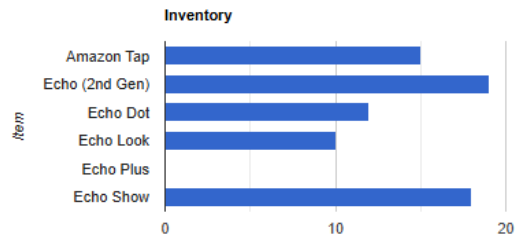


This page uses an Amazon Cognito identity to retrieve data directly from Amazon DynamoDB.

Store:

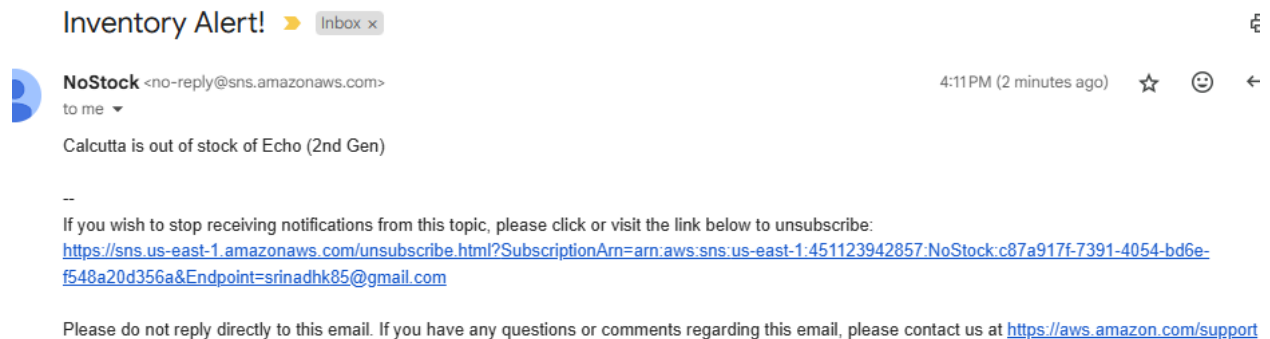
Berlin ▾

Store	Item	Count
Berlin	Amazon Tap	15
Berlin	Echo (2nd Gen)	19
Berlin	Echo Dot	12
Berlin	Echo Look	10
Berlin	Echo Plus	0
Berlin	Echo Show	18



This page uses an Amazon Cognito identity to retrieve data directly from Amazon DynamoDB.

Also, I 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).

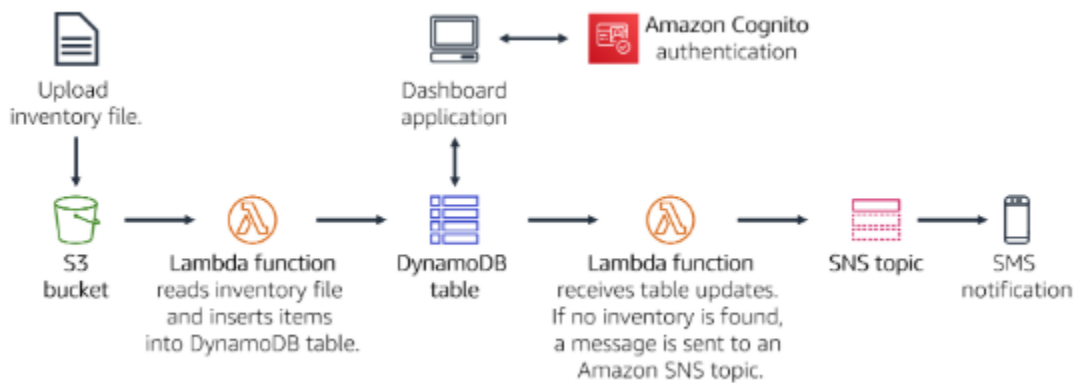


If I did not receive a notification, wait a few minutes and upload a different inventory file. The DynamoDB trigger can sometimes take a few minutes to enable.

45. Upload multiple inventory files at the same time and check it.

After completing this project, our architecture will look like the following diagram:

- 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



+++++

Created Required Roles to perform this project

## 1. Lambda-Load-Inventory-Role

The screenshot shows the AWS IAM console for the role 'Lambda-Load-Inventory-Role'. The left sidebar shows the 'Roles' link under 'Access management'. The main content area shows the role's summary and permissions policies.

**Summary**

- Creation date: March 20, 2025, 14:07 (UTC)
- Last activity: 53 minutes ago
- ARN: `arn:aws:iam::451123942857:role/Lambda-Load-Inventory-Role`
- Maximum session duration: 1 hour

**Permissions policies (3)**

Policy name	Type	Attached entities
<a href="#">AmazonDynamoDBFullAccess</a>	AWS managed	1
<a href="#">AmazonS3ReadOnlyAccess</a>	AWS managed	1
<a href="#">CWLogsPolicy</a>	Customer inline	0

## CWLogsPolicy

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Action": [  
6         "logs:CreateLogGroup",  
7         "logs:CreateLogStream",  
8         "logs:PutLogEvents"  
9       ],  
10      "Resource": "arn:aws:logs:*:*:*",  
11      "Effect": "Allow"  
12    }  
13  ]  
14 }
```

## 2. Lambda-Check-Stock-Role

The screenshot displays the AWS IAM console interface for the role 'Lambda-Check-Stock-Role'. The left sidebar shows the navigation menu with 'Roles' selected. The main content area is divided into two sections: 'Summary' and 'Permissions policies (2)'. The 'Summary' section shows the role's creation date (March 20, 2025, 14:07 UTC), last activity (8 minutes ago), ARN (arn:aws:iam::451123942857:role/Lambda-Check-Stock-Role), and maximum session duration (1 hour). The 'Permissions policies (2)' section shows two attached policies: 'AmazonSNSFullAccess' and 'AWSLambdaDynamoDBExecutionRole', both of which are AWS managed. The 'Permissions' tab is selected, and the 'Permissions policies (2)' section is expanded. A red arrow points to the 'Permissions' tab, and another red arrow points to the 'Permissions policies (2)' section. A red bracket highlights the two attached policies.

**Summary**

Creation date  
March 20, 2025, 14:07 (UTC)

Last activity  
8 minutes ago

ARN  
arn:aws:iam::451123942857:role/Lambda-Check-Stock-Role

Maximum session duration  
1 hour

**Permissions policies (2)**

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
<input type="checkbox"/> AmazonSNSFullAccess	AWS managed	1
<input type="checkbox"/> AWSLambdaDynamoDBExecutionRole	AWS managed	1

+++++