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

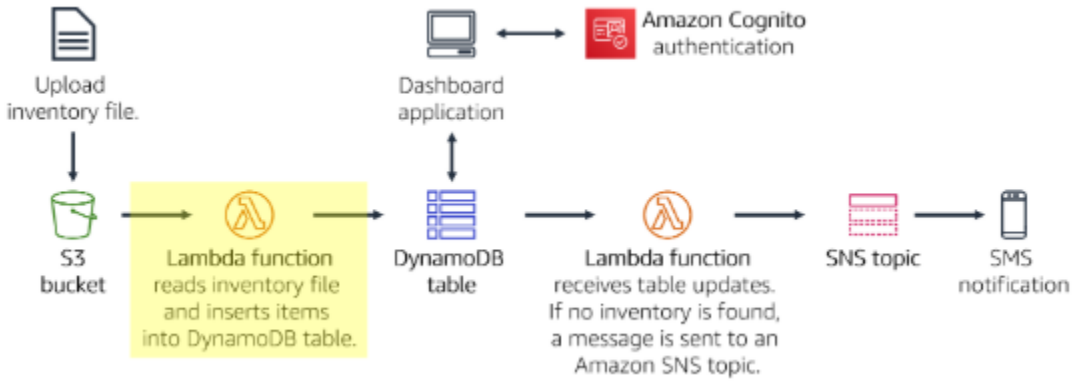
Traditionally, applications run on servers. These servers can be physical (or bare metal). They can also be virtual environments that run on top of physical servers. However, you must purchase and provision all these types of servers, and you must also manage their capacity. In contrast, you can run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

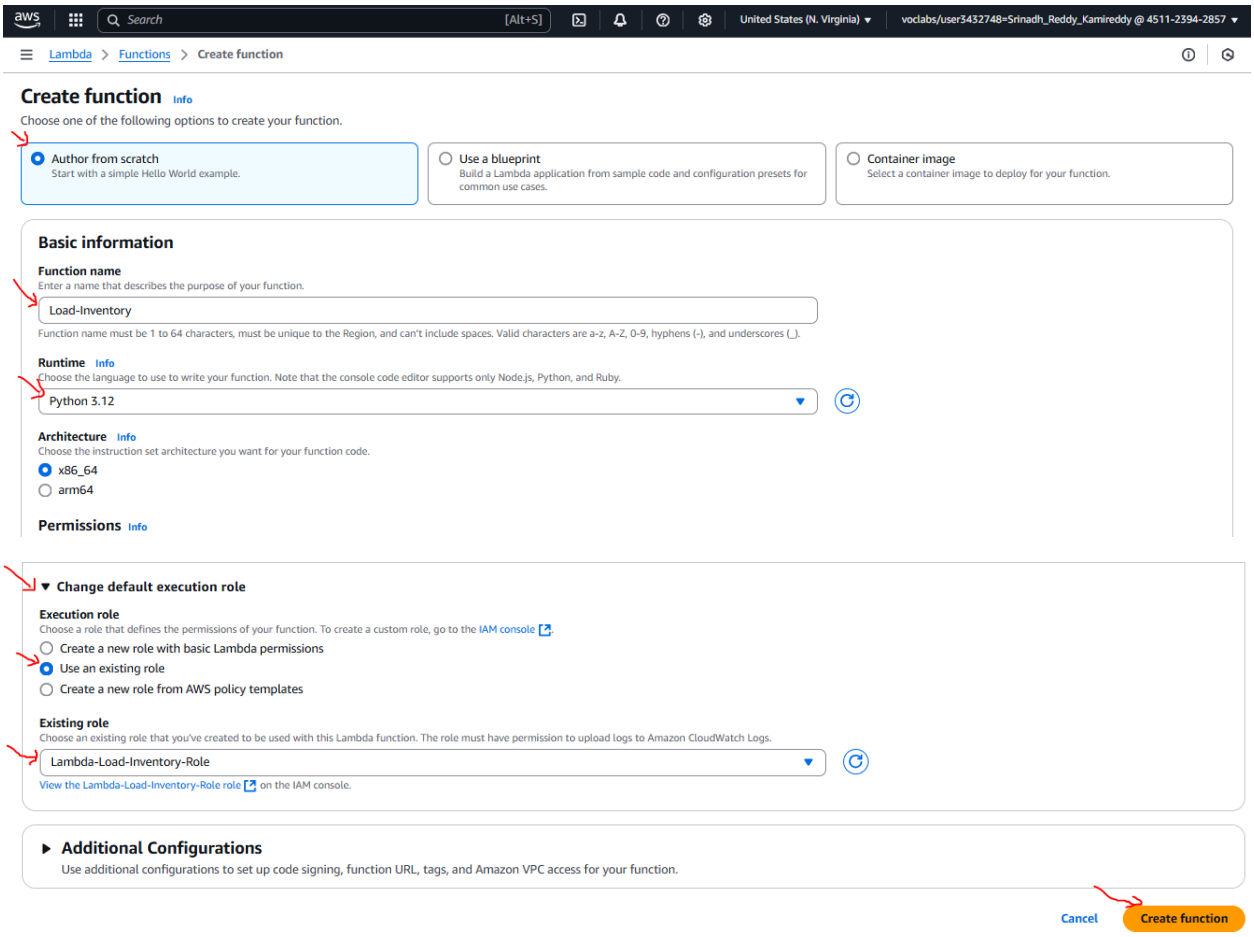
* Runtime: Python 3.12

* Expand Change default execution role.

* Execution role: Use an existing role

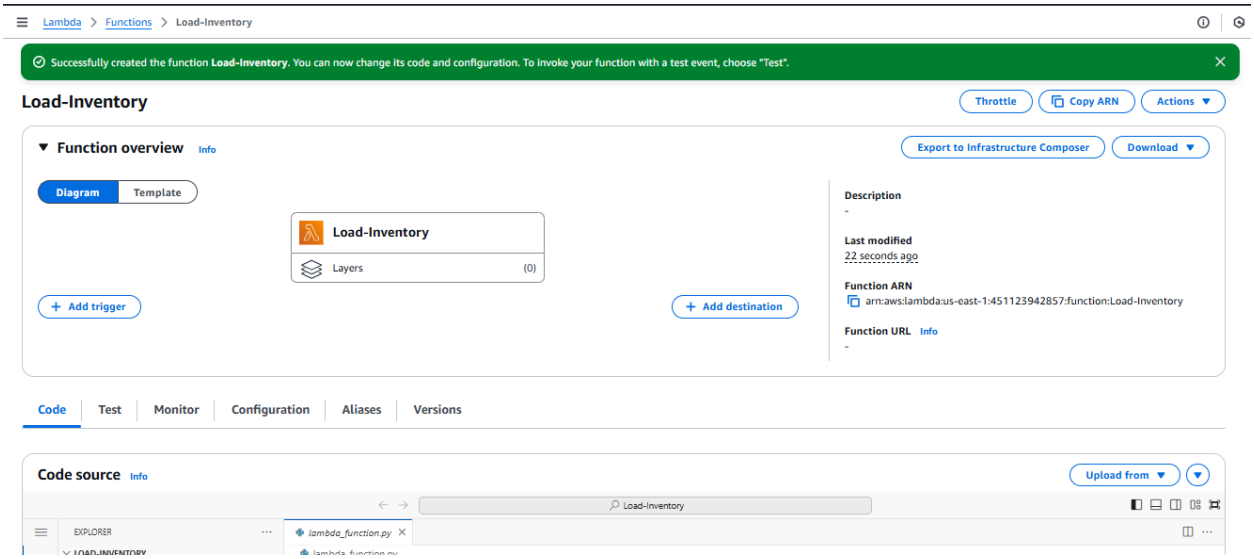
* Existing role: Lambda-Load-Inventory-Role

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

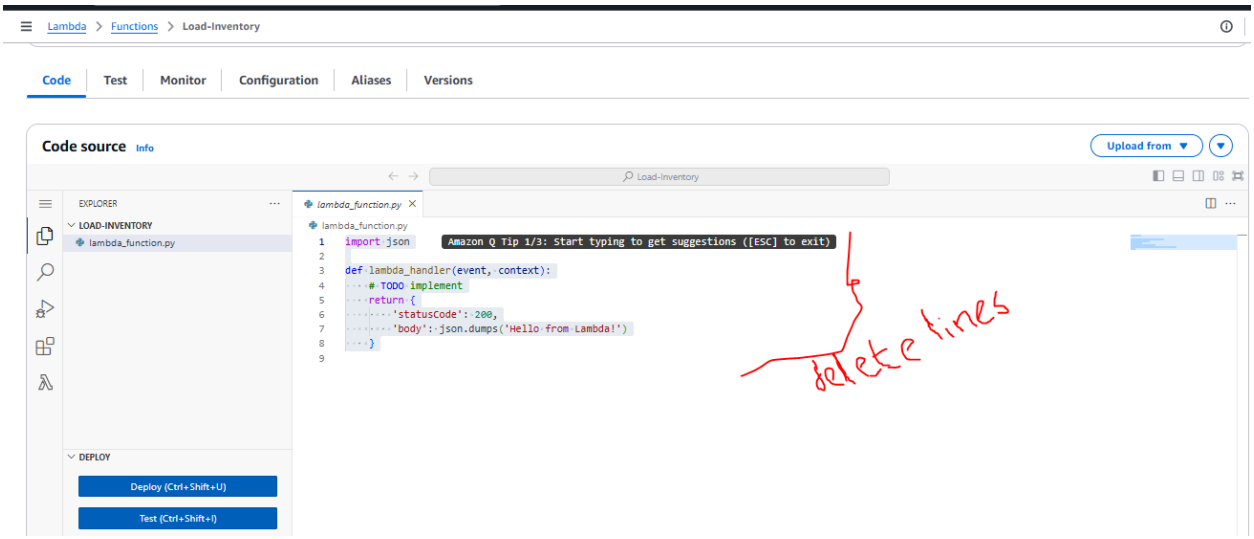


4. Choose Create function

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



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



7. In the Code source editor, copy and paste the following code:

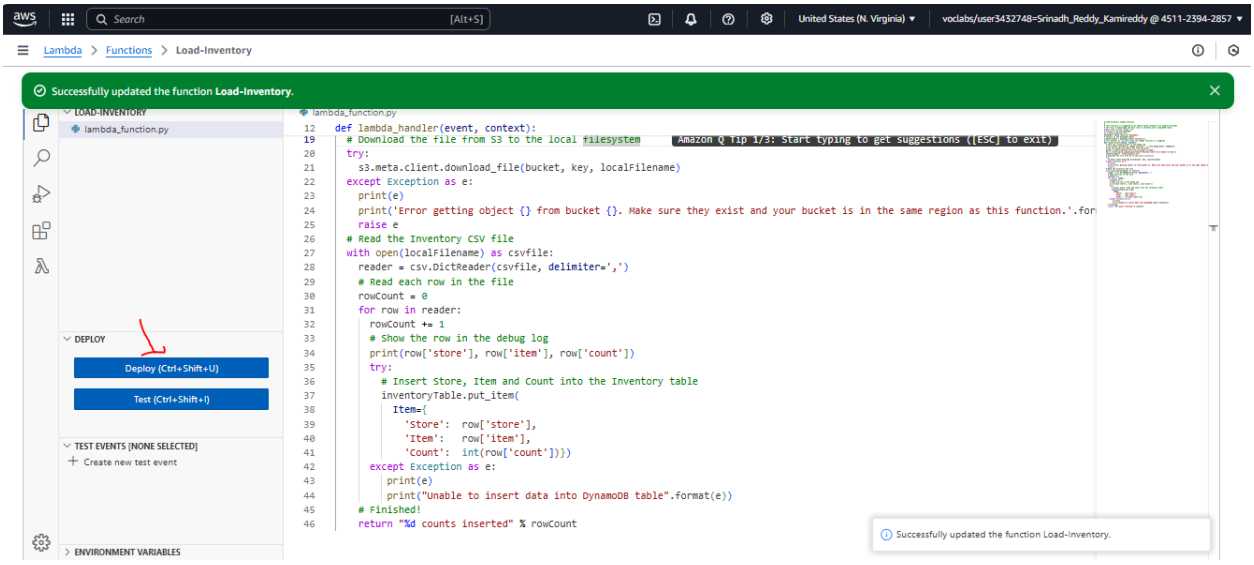
```
# Load-Inventory Lambda function

#
# 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

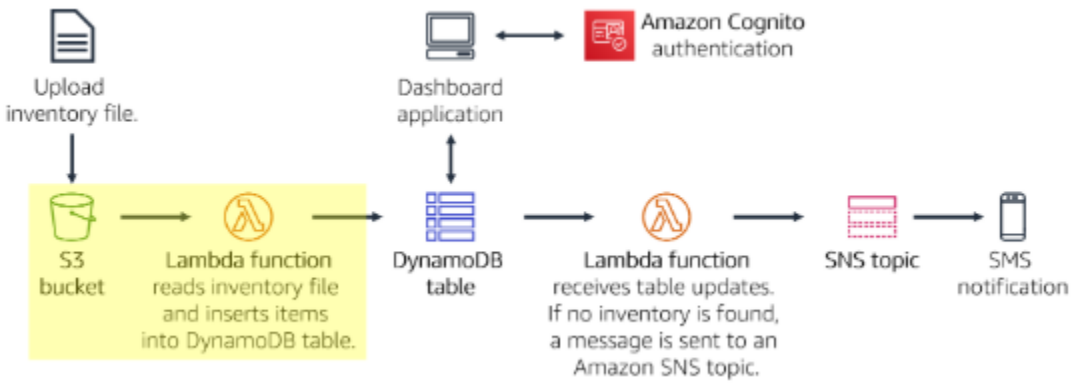
8. Choose Deploy to save your changes.



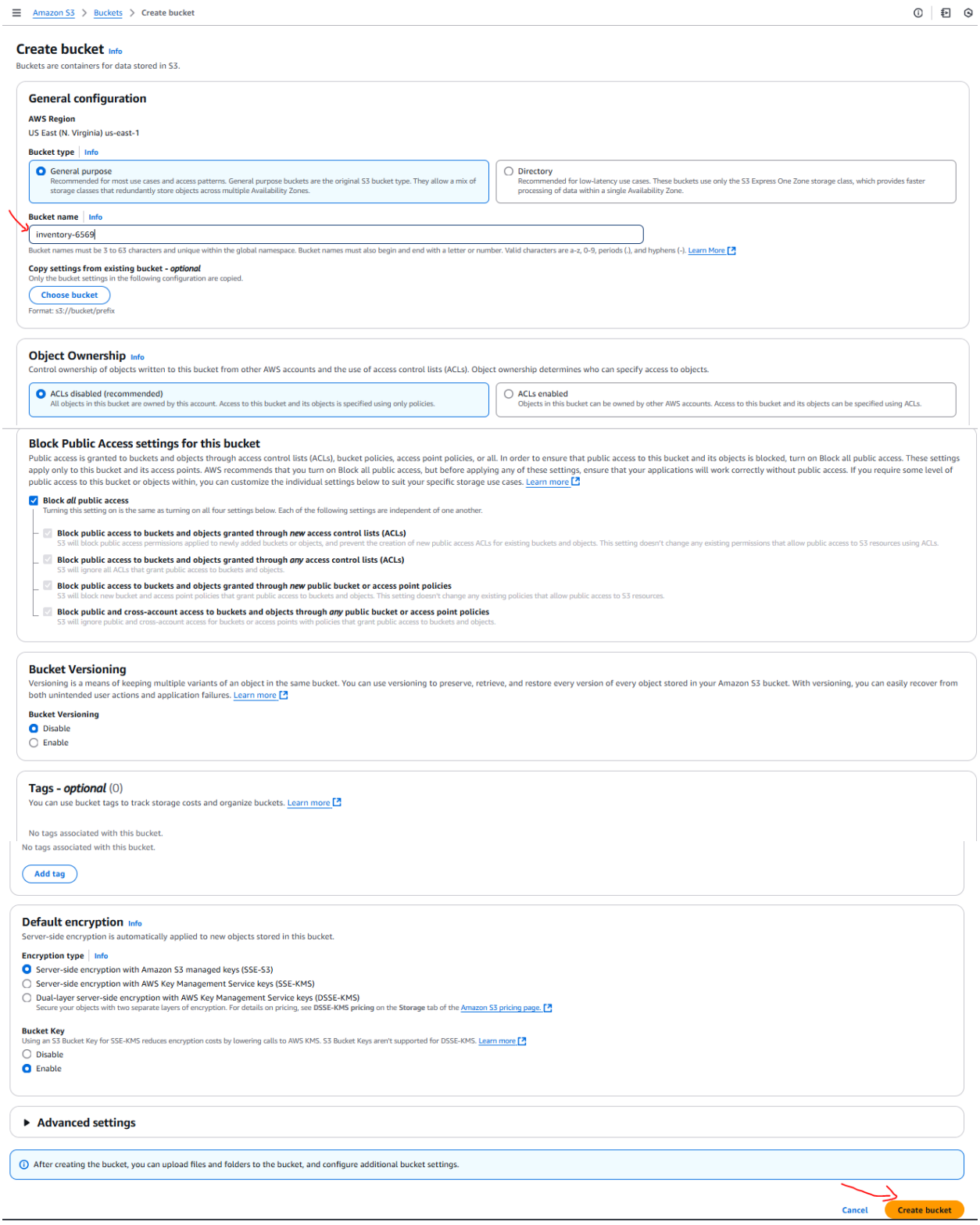
Next, I will configure Amazon S3 to trigger the Lambda function when a file is uploaded.

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.

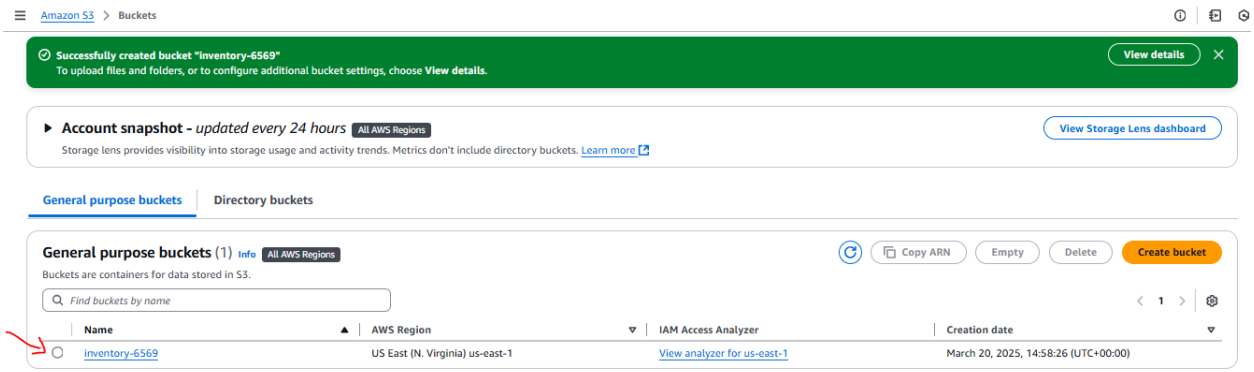


In this task, I will create an S3 bucket and configure it to trigger the Lambda function.
9. On the Services menu, choose S3.
10. Choose Create bucket
Each bucket must have a unique name, so I will add a random number to the bucket name. For example: inventory-6569
11. For Bucket name enter: inventory-<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.



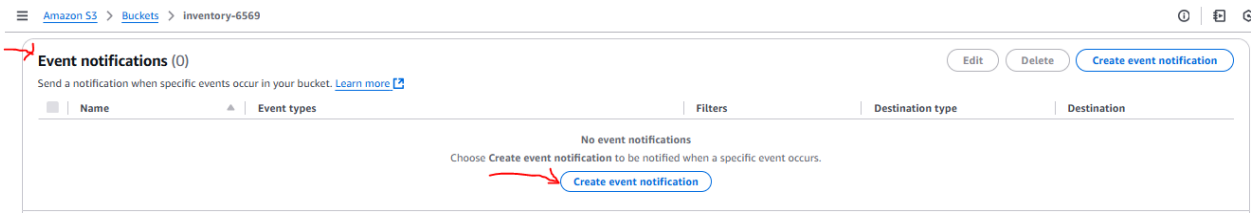
Now I configure the bucket to automatically trigger the Lambda function when a file is uploaded.

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



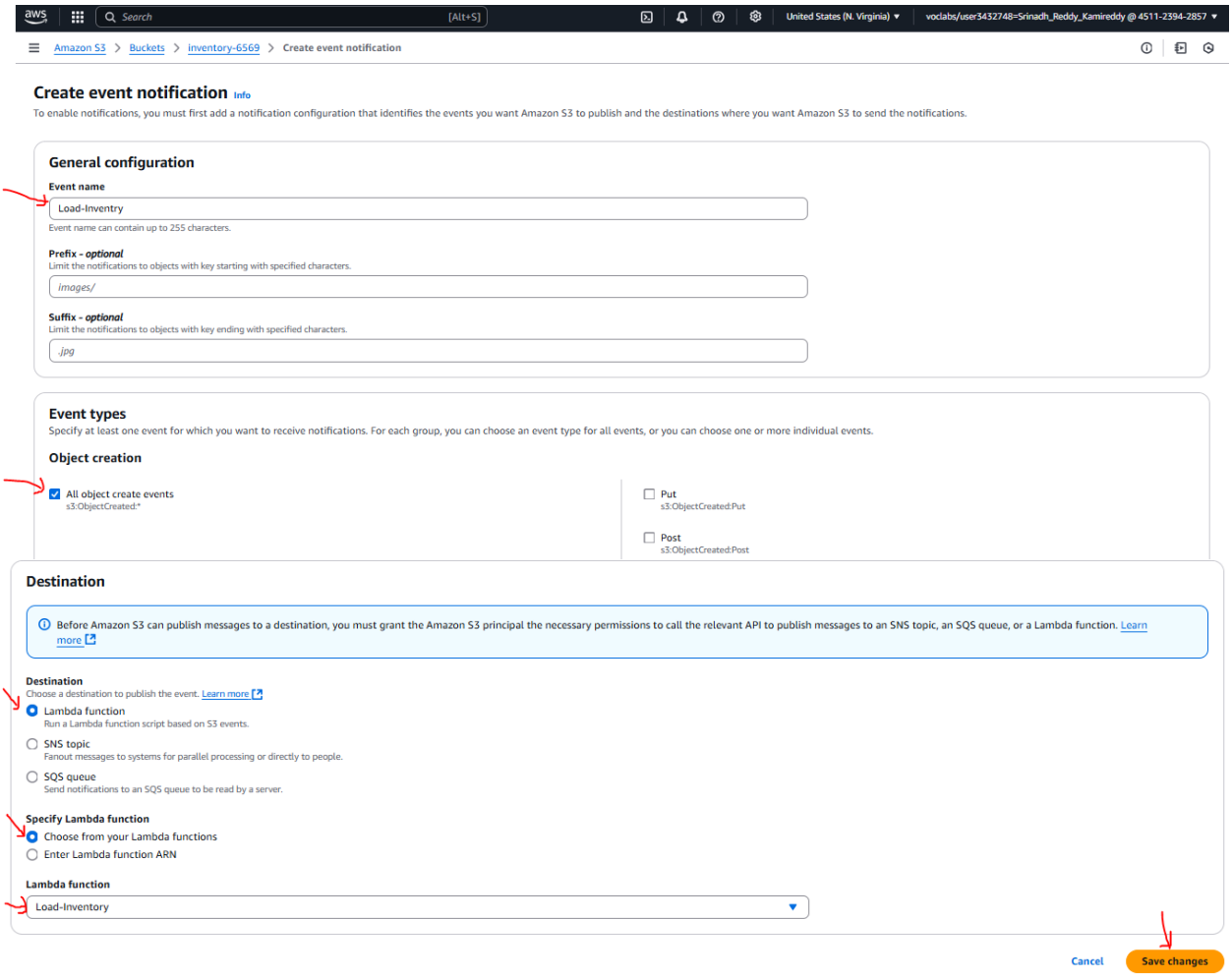
14. Choose the Properties tab.

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:

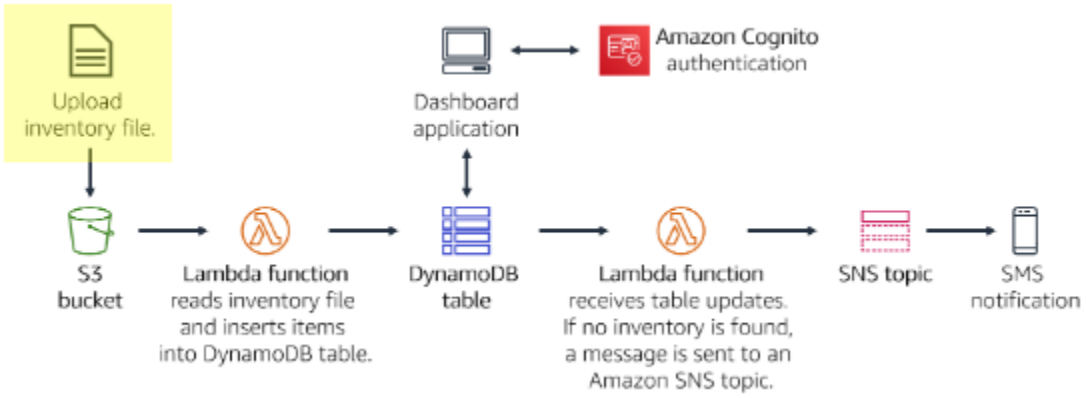
* Name: Load-Inventory
* Event types: All object create events
* Destination: Lambda Function
* Lambda function: Load-Inventory
* Choose Save changes



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.

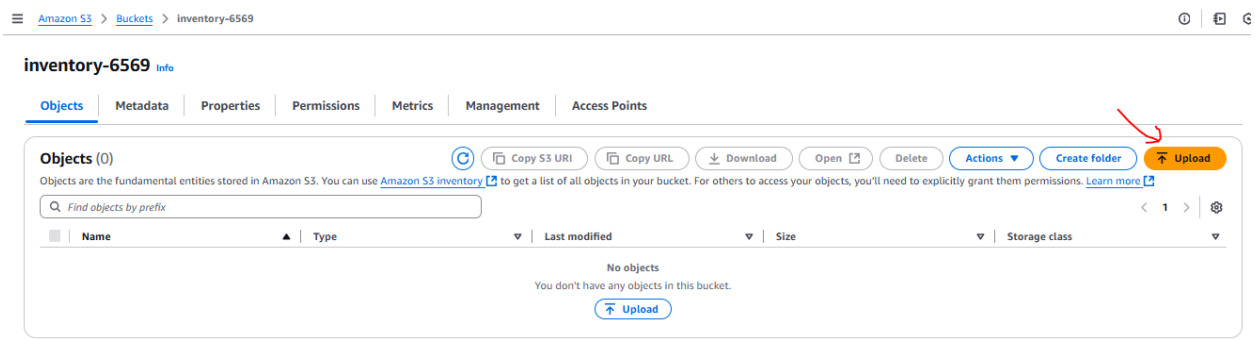


21. Download the inventory files from the attachments.
inventory-berlin.csv
inventory-calcutta.csv
inventory-karachi.csv
inventory-pusan.csv
inventory-shanghai.csv
inventory-springfield.csv

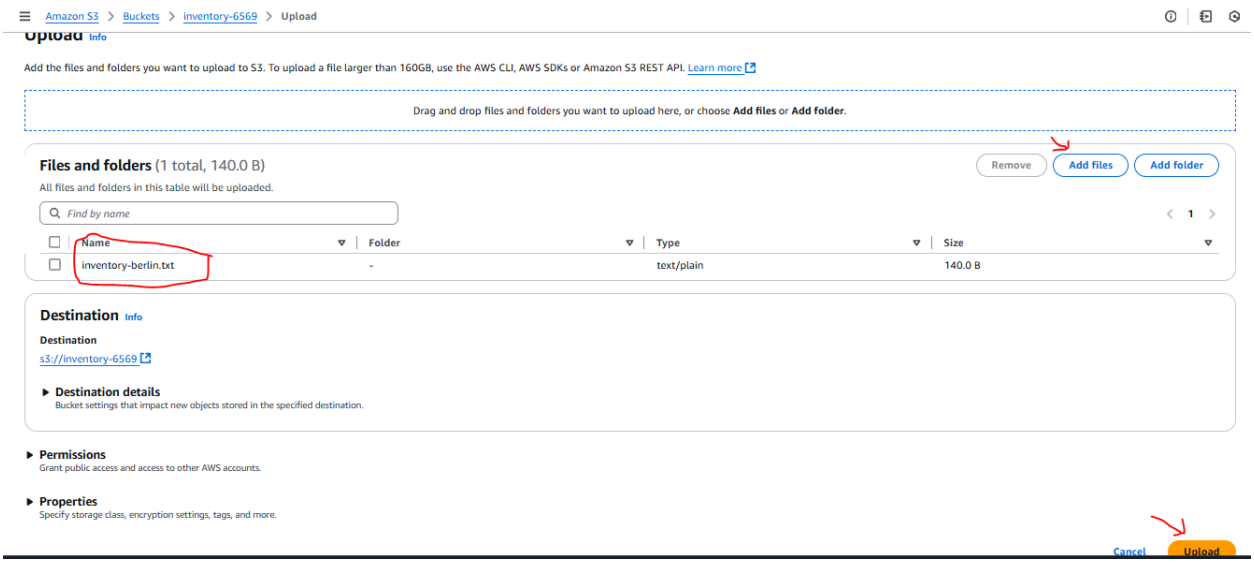
These files are the inventory files that I can use to test the system. They are comma-separated values (CSV) files.
The following example shows the contents of the Berlin file:

```
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.

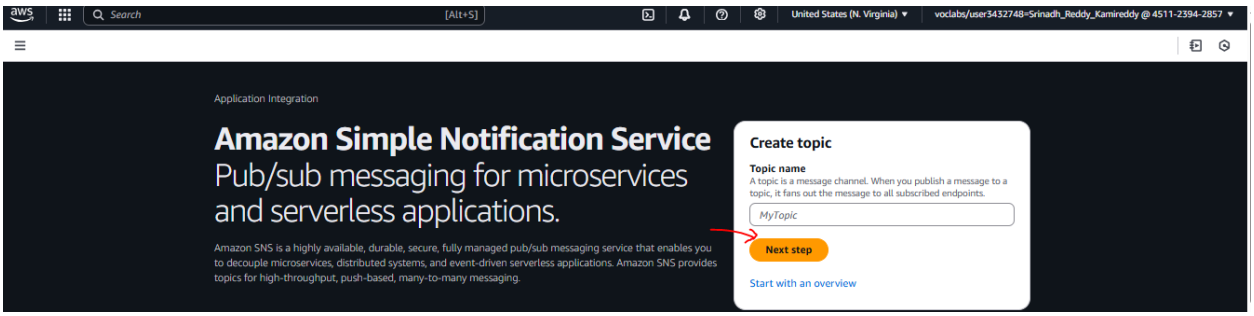
The dashboard application is served as a static webpage from Amazon S3. The dashboard authenticates via Amazon Cognito as an anonymous user, which provides sufficient permissions for the dashboard to retrieve data from DynamoDB.
We can also view the data directly in the DynamoDB table.
25. On the Services 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

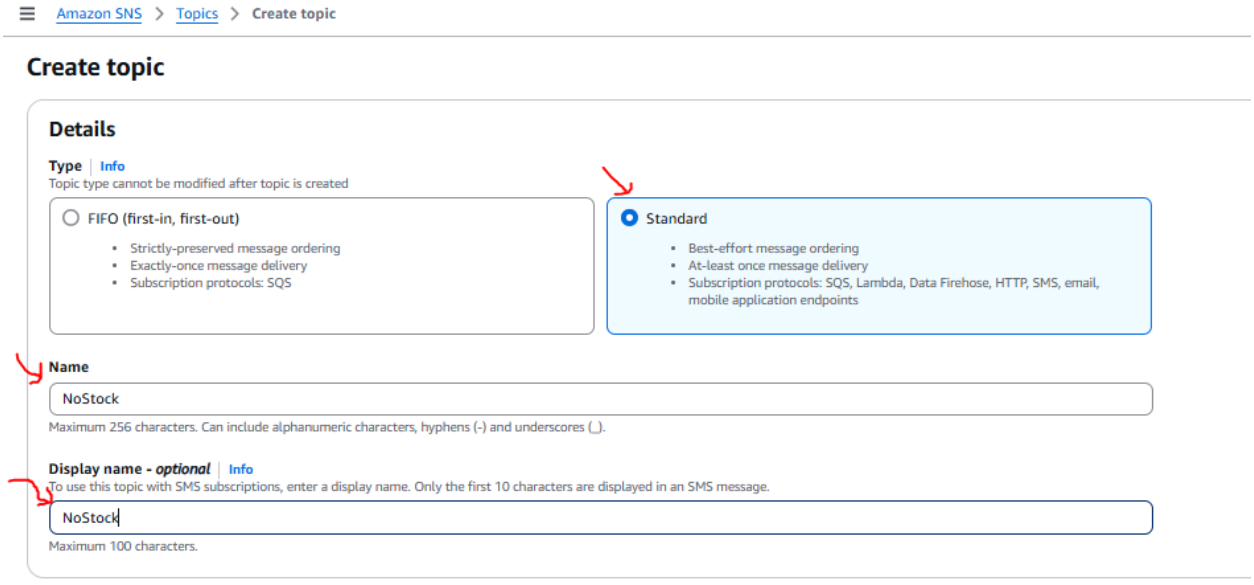
I want to notify inventory management staff when a store runs out of stock for an item. For this serverless notification functionality, I will use Amazon SNS.

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.

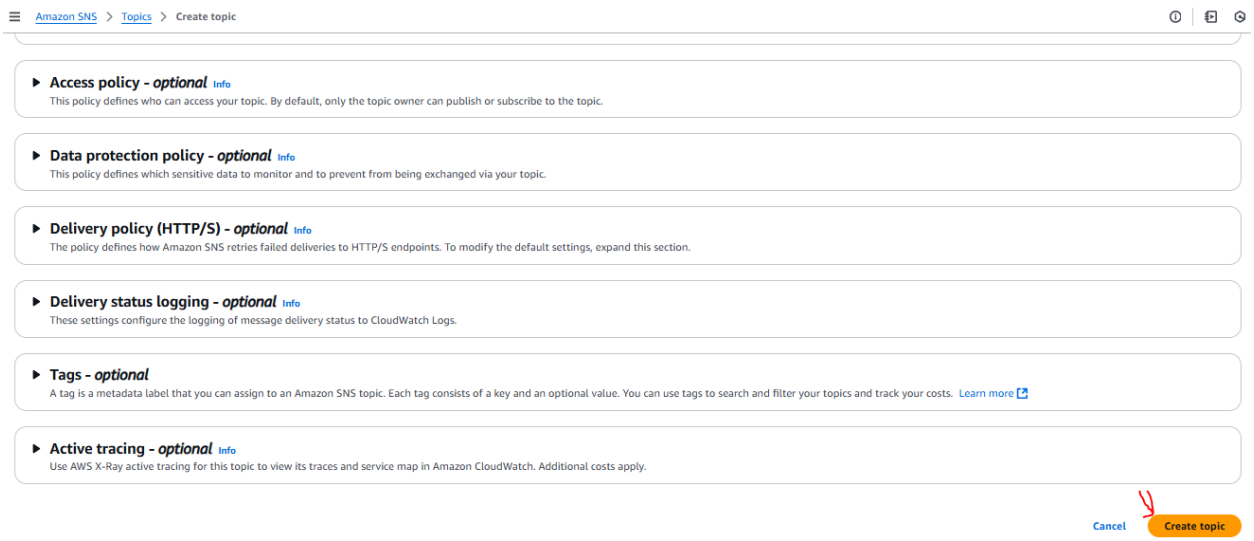
29. On the Services menu, choose Simple Notification Service. Click on Next step.



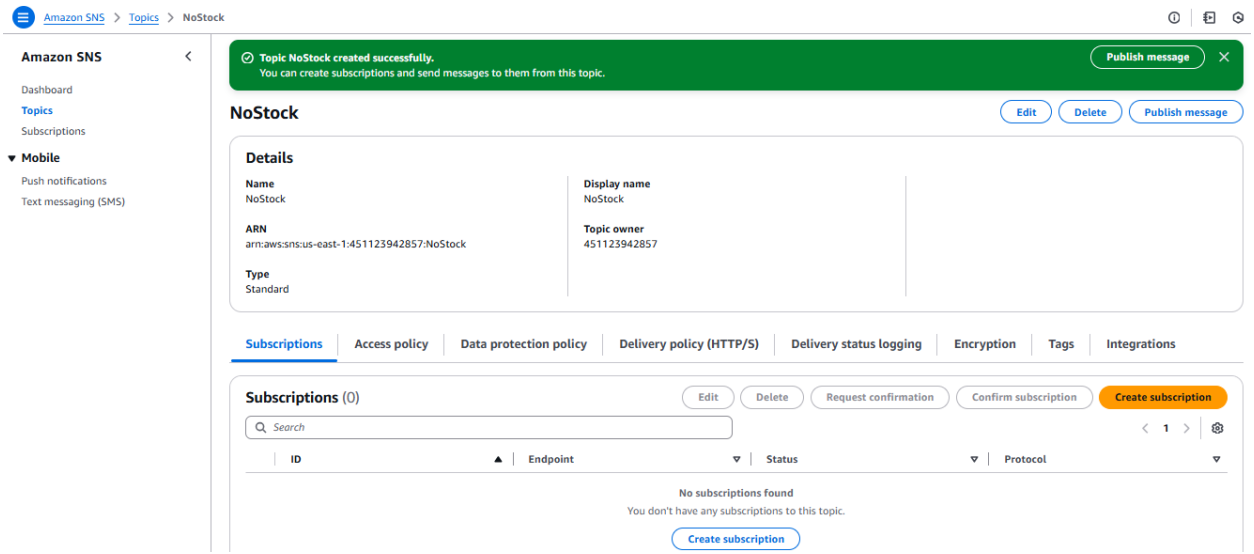
30. In the Create topic box.
* Details: Standard
* Name: NoStock
* Display Name: NoStock



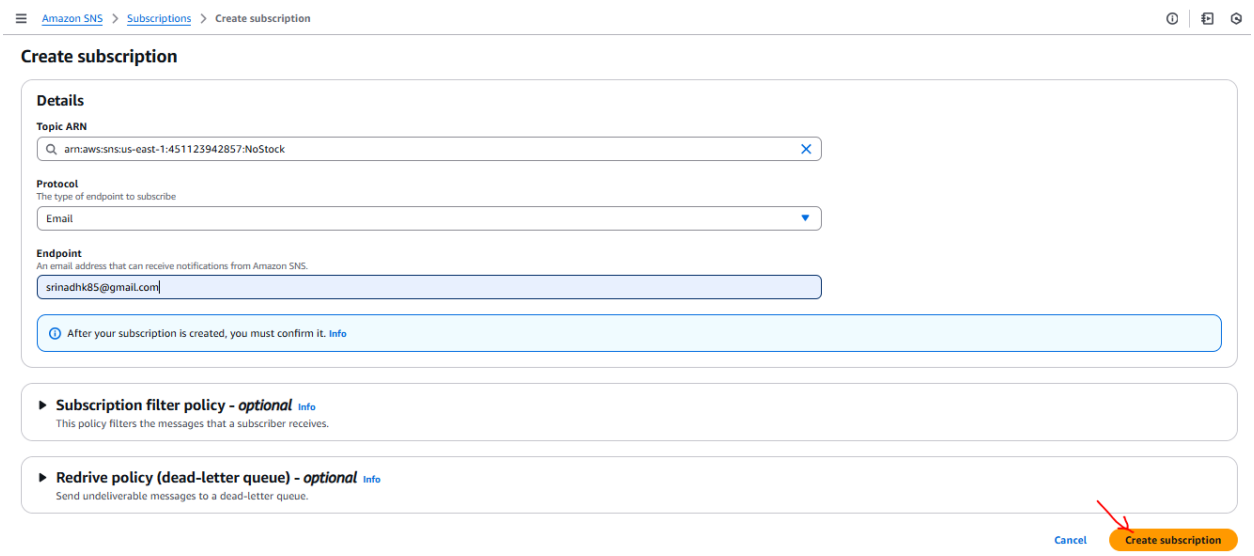
31. Choose Create topic



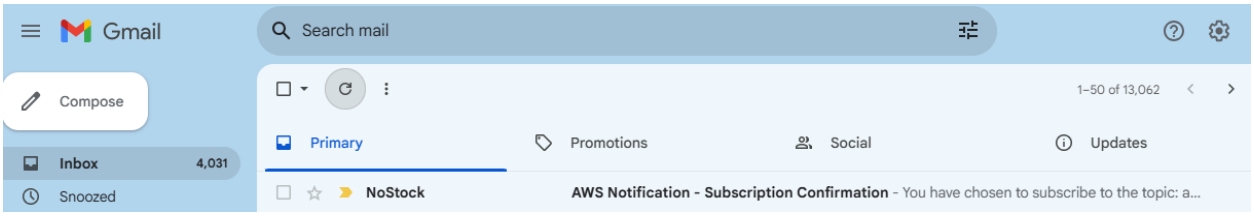
To receive notifications, we must subscribe to the topic. we can choose to receive notifications via several methods, such as SMS and email.

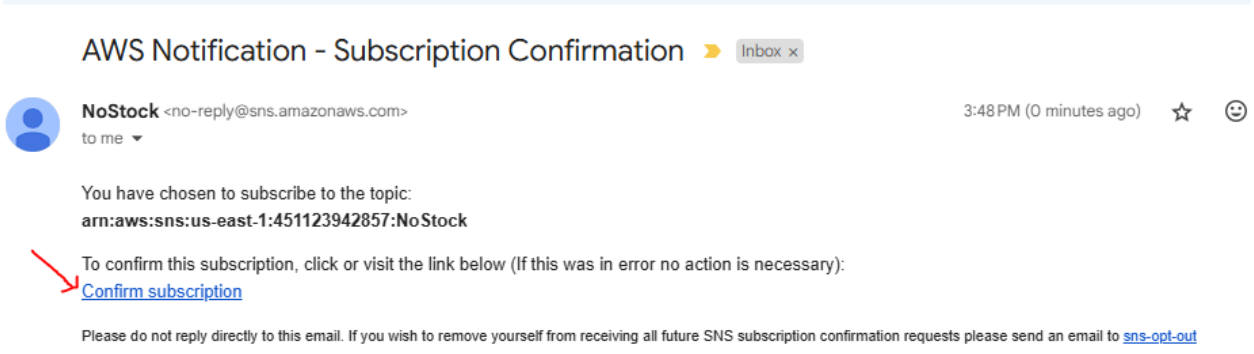


32. In the lower of the page, choose Create subscription and configure these settings:
Protocol: Email
Endpoint: Enter your email address
Choose Create subscription



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.





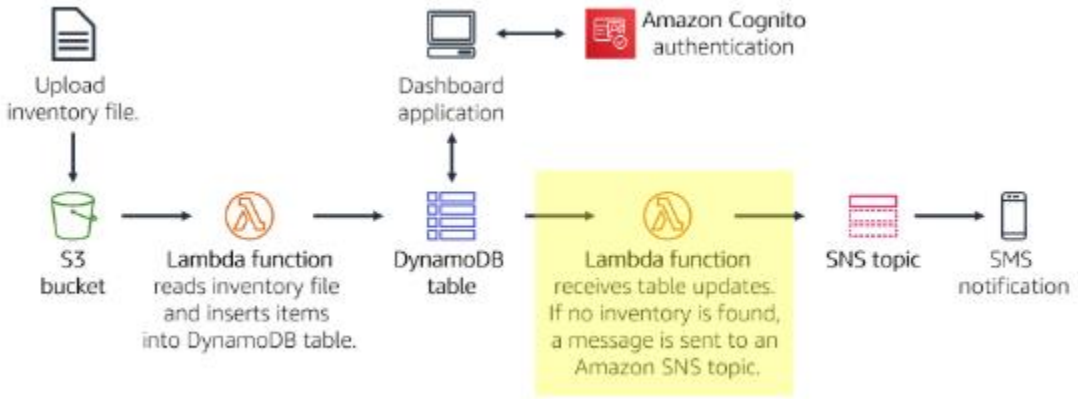
Task 5: Creating a Lambda function to send notifications

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.



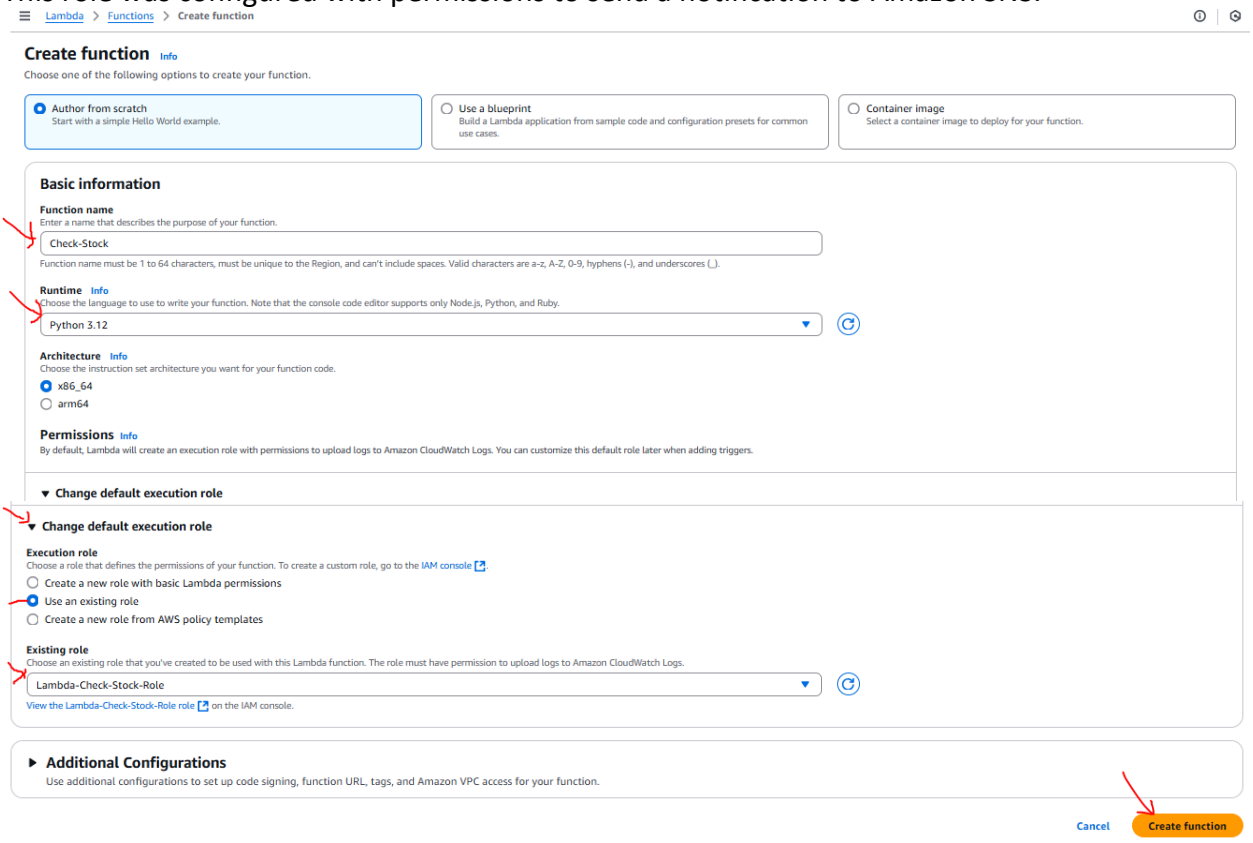
33. On the Services menu, choose Lambda.

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.



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:

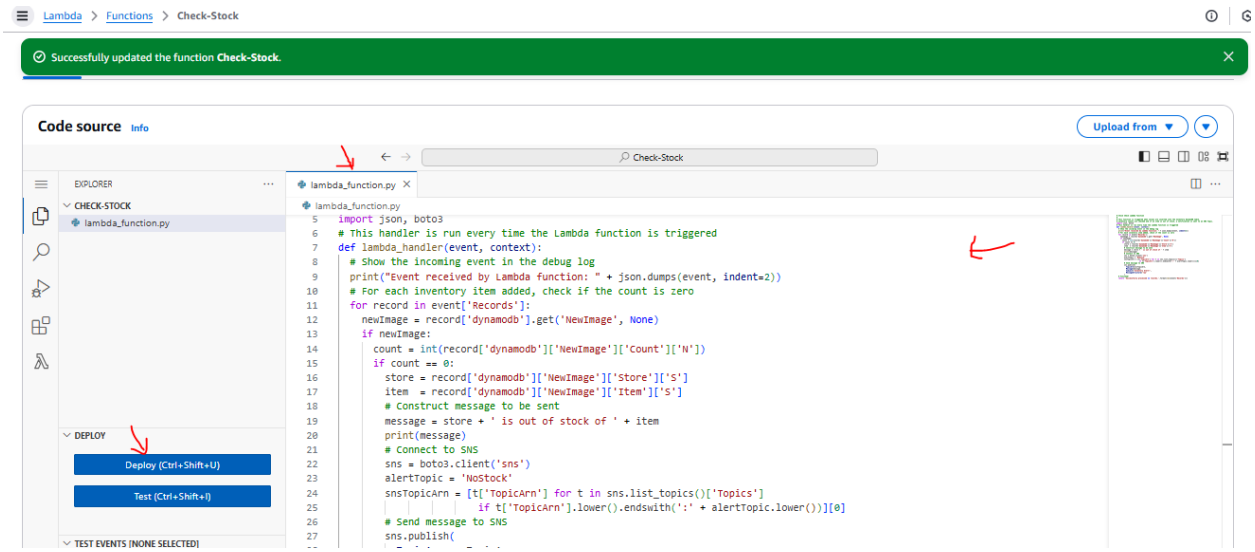
```
# Stock Check Lambda function
#
# This function is triggered when values are inserted into the Inventory DynamoDB table.
# Inventory counts are checked and if an item is out of stock, a notification is sent to an SNS Topic.
import json, boto3
# This handler is run every time the Lambda function is 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:

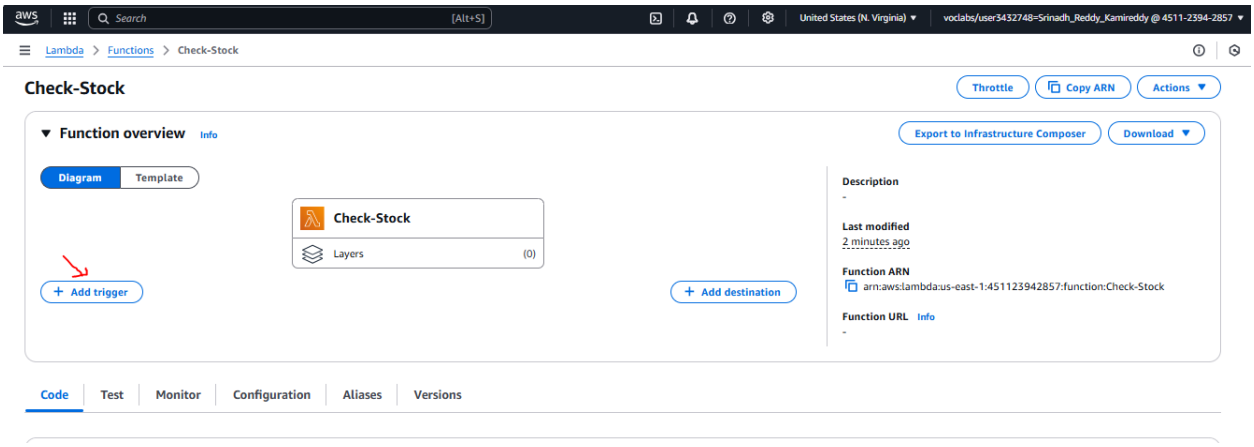
* Loop through the incoming records
* If the inventory count is zero, send a message to the NoStock SNS topic

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



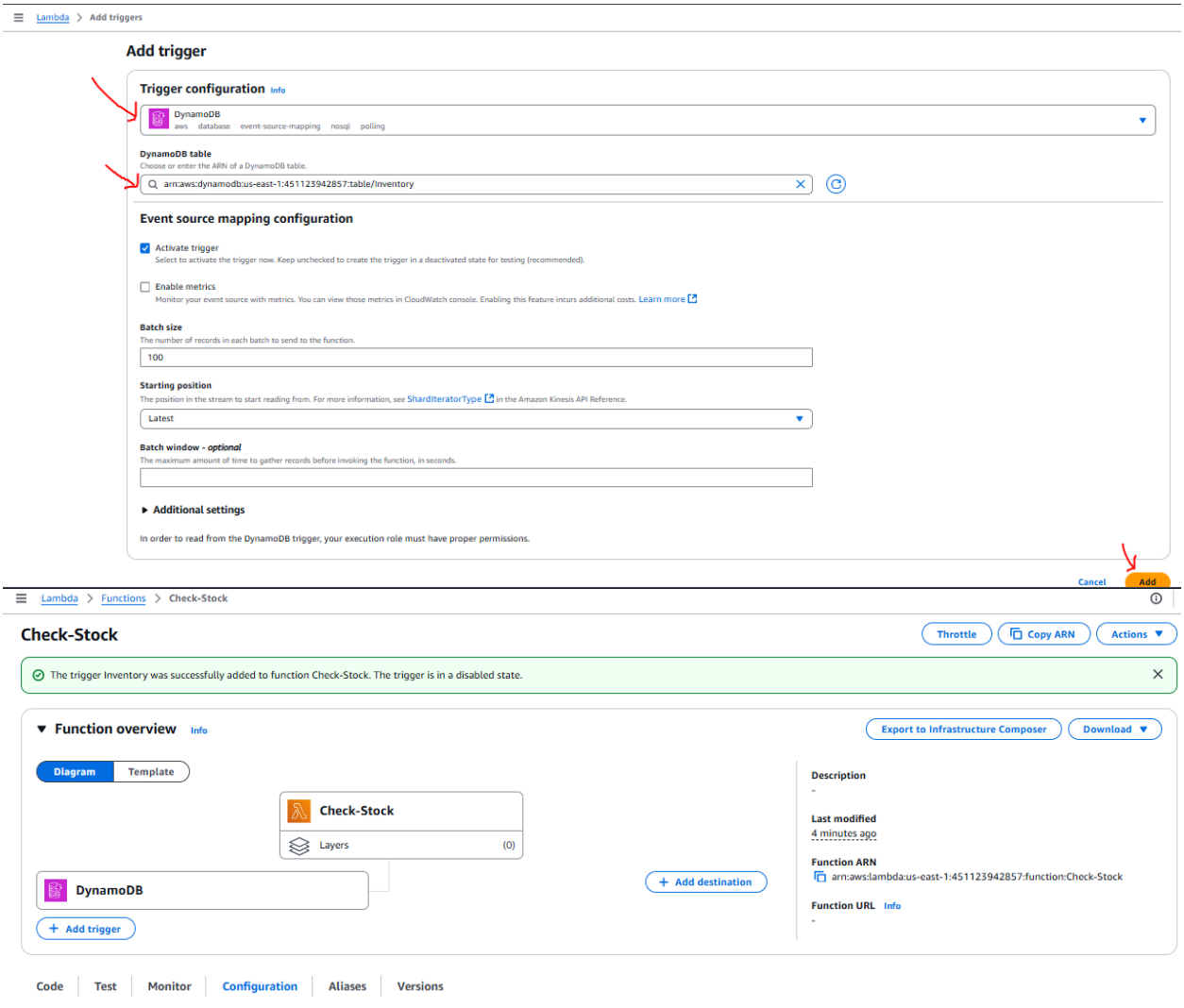
38. Choose Deploy to save your code changes

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



40. Choose Add trigger and then configure these settings:

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



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.

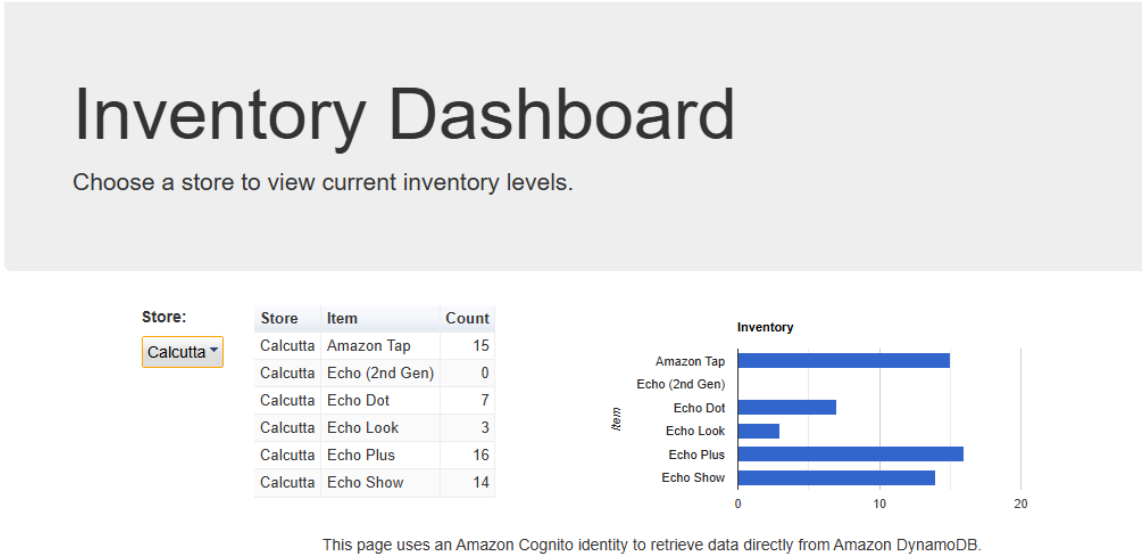
41. On the Services menu, choose S3.

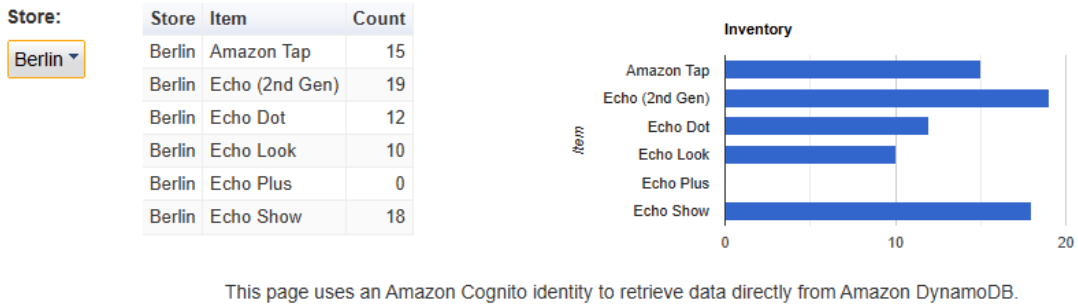
42. Choose the name of your inventory- bucket.

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

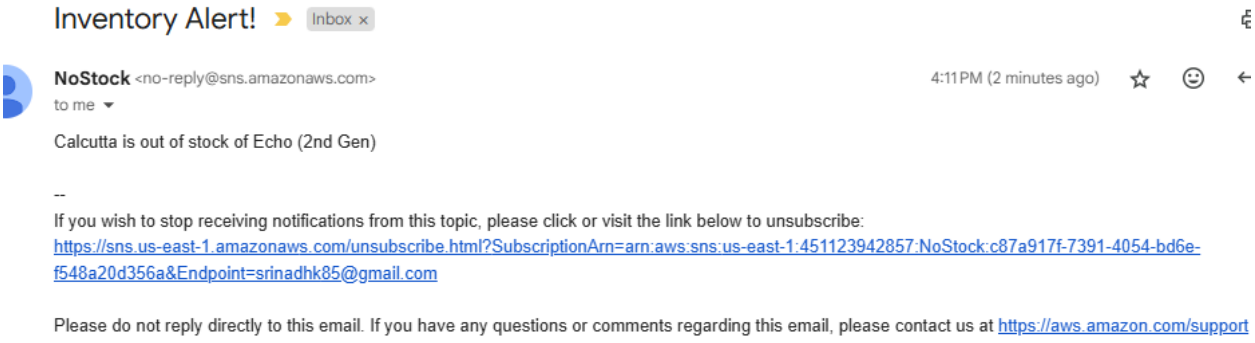
44. Return to the Inventory System Dashboard and refresh the page.

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





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).

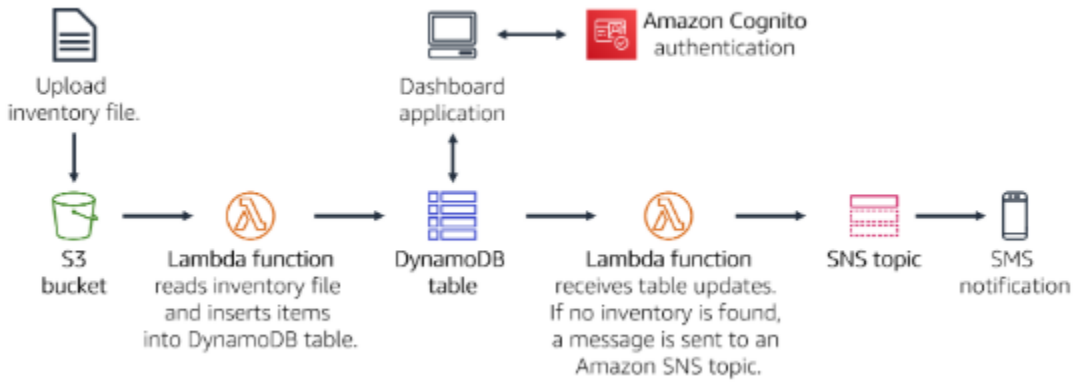


If I did not receive a notification, wait a few minutes and upload a different inventory file. The DynamoDB trigger can sometimes take a few minutes to enable.

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

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

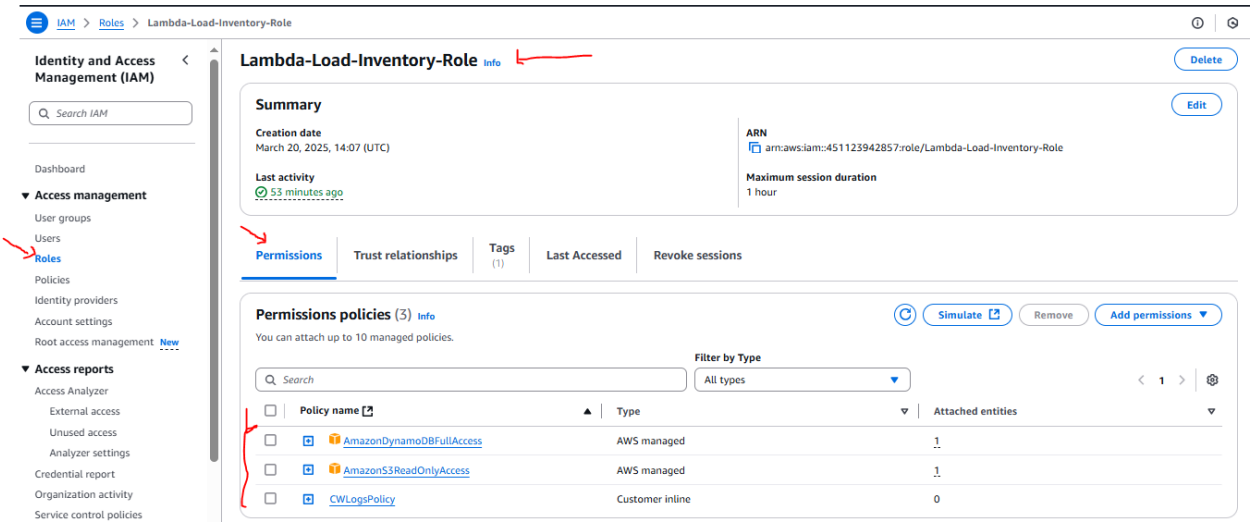
- Implement a serverless architecture on AWS
- Trigger Lambda functions from Amazon S3 and Amazon DynamoDB
- Configure Amazon Simple Notification Service (Amazon SNS) to send notifications



++

Created Required Roles to perform this project

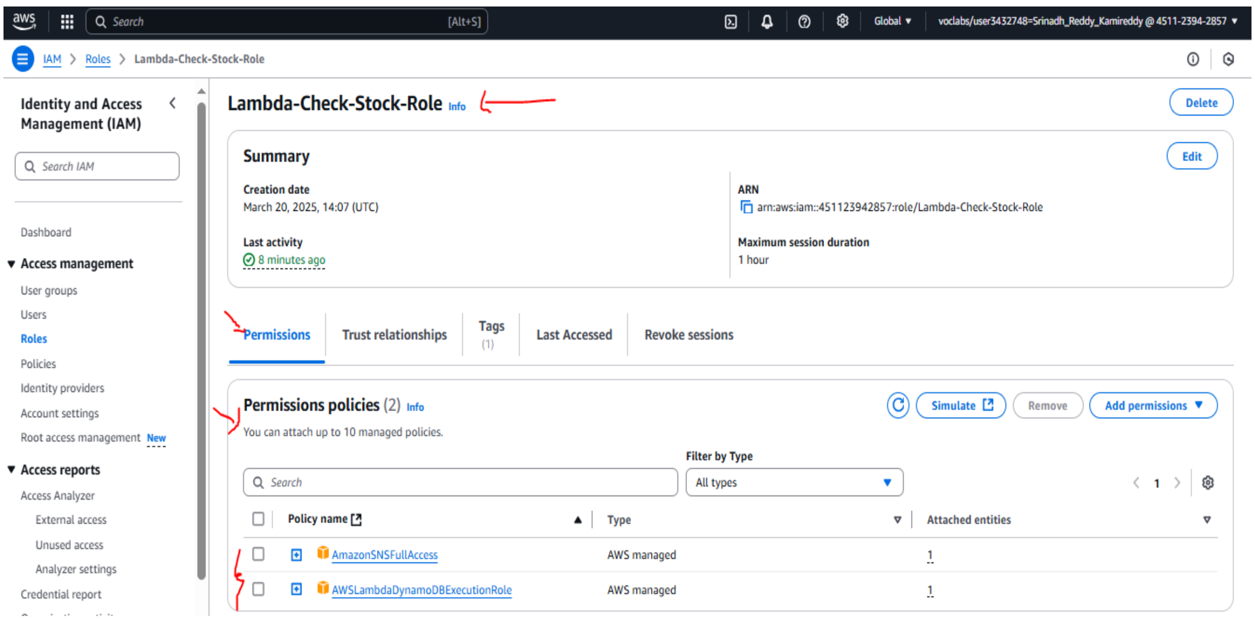
1. Lambda-Load-Inventory-Role



CWLogsPolicy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*",
            "Effect": "Allow"
        }
    ]
}
```

2. Lambda-Check-Stock-Role



++