Automated Receipt ETL System with AWS

A cloud-native ETL solution built using serverless and AI-driven AWS services

I built a serverless ETL system using AWS to automate receipt extraction. I used Textract to extract fields like vendor, total, and items. Then, I transformed and stored the data into DynamoDB using a Lambda function and sent notification emails via SES. This helped me understand how to design cloud-native pipelines and implement data governance practices, something I believe aligns strongly with Deloitte's cloud consulting engagements.

Category: Data Engineering, ETL, Serverless Workflows

Project Objective

Automate receipt processing to eliminate manual data entry, reduce errors, and enable scalable financial data extraction using ETL (Extract, Transform, Load) principles on the cloud.

Services Used & ETL Mapping

ETL Stage	AWS Service	Functionality
Extract	Amazon Textract	OCR-based data extraction from receipts
Transform	AWS Lambda + Python	Cleansing, structuring, and validation
Load	Amazon DynamoDB	Stores structured receipt data
Notify	Amazon SES	Sends processed results via email
Storage	Amazon S3	Securely stores scanned receipt files
Security	IAM Roles & Policies	Controls access to cloud services

Architecture Overview

1. User uploads receipt → S3 Bucket
2. S3 triggers Lambda → Lambda calls Textract
3. Textract extracts fields → Lambda structures data
4. Lambda writes to DynamoDB → SES sends notification

Add diagram screenshot here if available.

Code Sample – Lambda Orchestration

```
import json
import os
import boto3
import uuid
```

```python
from datetime import datetime
import urllib.parse

# Initialize AWS clients
s3 = boto3.client('s3')
textract = boto3.client('textract')
dynamodb = boto3.resource('dynamodb')
ses = boto3.client('ses')

# Environment variables
DYNAMODB_TABLE = os.environ.get('DYNAMODB_TABLE', 'Receipts')
SES_SENDER_EMAIL = os.environ.get('SES_SENDER_EMAIL', 'your-email@example.com')
SES_RECIPIENT_EMAIL = os.environ.get('SES_RECIPIENT_EMAIL', 'recipient@example.com')

def lambda_handler(event, context):
    try:
        # Get the S3 bucket and key from the event
        bucket = event['Records'][0]['s3']['bucket']['name']
        # URL decode the key to handle spaces and special characters
        key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])

        print(f"Processing receipt from {bucket}/{key}")

        # Verify the object exists before proceeding
        try:
            s3.head_object(Bucket=bucket, Key=key)
            print(f"Object verification successful: {bucket}/{key}")
        except Exception as e:
            print(f"Object verification failed: {str(e)}")
            raise Exception(f"Unable to access object {key} in bucket {bucket}: {str(e)}")

        # Step 1: Process receipt with Textract
        receipt_data = process_receipt_with_textract(bucket, key)

        # Step 2: Store results in DynamoDB
        store_receipt_in_dynamodb(receipt_data, bucket, key)

        # Step 3: Send email notification
        send_email_notification(receipt_data)

        return {
            'statusCode': 200,
            'body': json.dumps('Receipt processed successfully!')
        }
    except Exception as e:
```

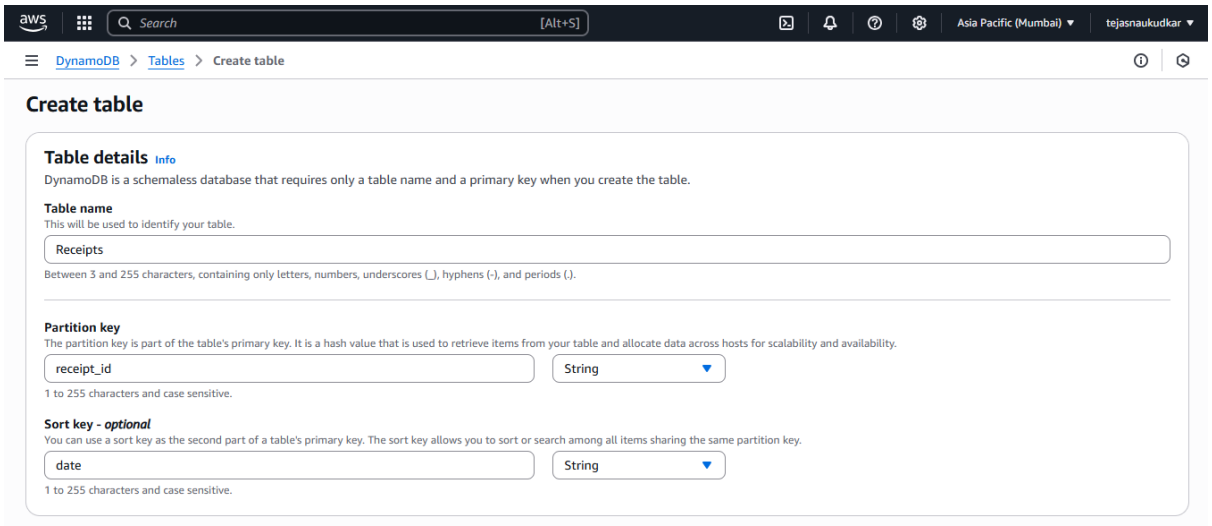
```
        print(f"Error processing receipt: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps(f'Error: {str(e)}')
        }

def process_receipt_with_textract(bucket, key):
    """Process receipt using Textract's AnalyzeExpense operation"""
    try:
        print(f"Calling Textract analyze_expense for {bucket}/{key}")
        response = textract.analyze_expense(
            Document={
                'S3Object': {
                    'Bucket': bucket,
                    'Name': key
                }
            }
        )
        print("Textract analyze_expense call successful")
    except Exception as e:
        print(f"Textract analyze_expense call failed: {str(e)}")
        raise

    # Generate a unique ID for this receipt
    receipt_id = str(uuid.uuid4())

    # Initialize receipt data dictionary
    receipt_data = {
        'receipt_id': receipt_id,
        'date': datetime.now().strftime('%Y-%m-%d'),  # Default date
        'vendor': 'Unknown',
        'total': '0.00',
        'items': [],
        's3_path': f"s3://{bucket}/{key}"
    }

    # Extract data from Textract response
    if 'ExpenseDocuments' in response and response['ExpenseDocuments']:
        expense_doc = response['ExpenseDocuments'][0]

        # Process summary fields (TOTAL, DATE, VENDOR)
        if 'SummaryFields' in expense_doc:
            for field in expense_doc['SummaryFields']:
                field_type = field.get('Type', {}).get('Text', '')
                value = field.get('ValueDetection', {}).get('Text', '')
```

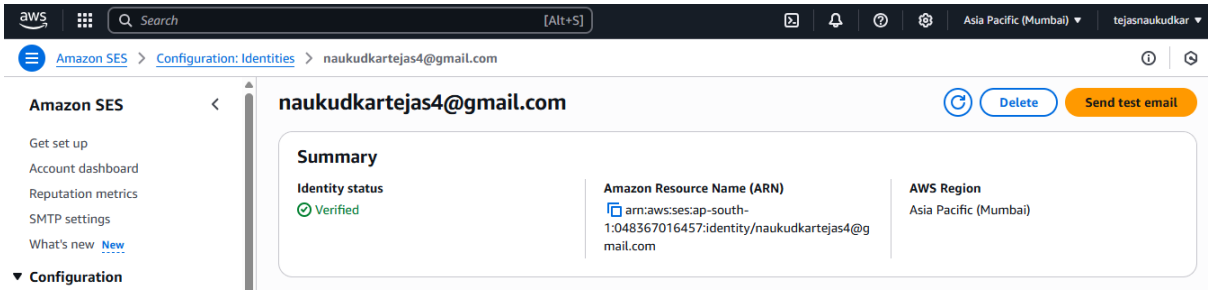
```
                if field_type == 'TOTAL':
                    receipt_data['total'] = value
                elif field_type == 'INVOICE_RECEIPT_DATE':
                    # Try to parse and format the date
                    try:
                        receipt_data['date'] = value
                    except:
                        # Keep the default date if parsing fails
                        pass
                elif field_type == 'VENDOR_NAME':
                    receipt_data['vendor'] = value

        # Process line items
        if 'LineItemGroups' in expense_doc:
            for group in expense_doc['LineItemGroups']:
                if 'LineItems' in group:
                    for line_item in group['LineItems']:
                        item = {}
                        for field in line_item.get('LineItemExpenseFields', []):
                            field_type = field.get('Type', {}).get('Text', '')
                            value = field.get('ValueDetection', {}).get('Text', '')

                            if field_type == 'ITEM':
                                item['name'] = value
                            elif field_type == 'PRICE':
                                item['price'] = value
                            elif field_type == 'QUANTITY':
                                item['quantity'] = value

                        # Add to items list if we have a name
                        if 'name' in item:
                            receipt_data['items'].append(item)

    print(f"Extracted receipt data: {json.dumps(receipt_data)}")
    return receipt_data

def store_receipt_in_dynamodb(receipt_data, bucket, key):
    """Store the extracted receipt data in DynamoDB"""
    try:
        table = dynamodb.Table(DYNAMODB_TABLE)

        # Convert items to a format DynamoDB can store
        items_for_db = []
        for item in receipt_data['items']:
```

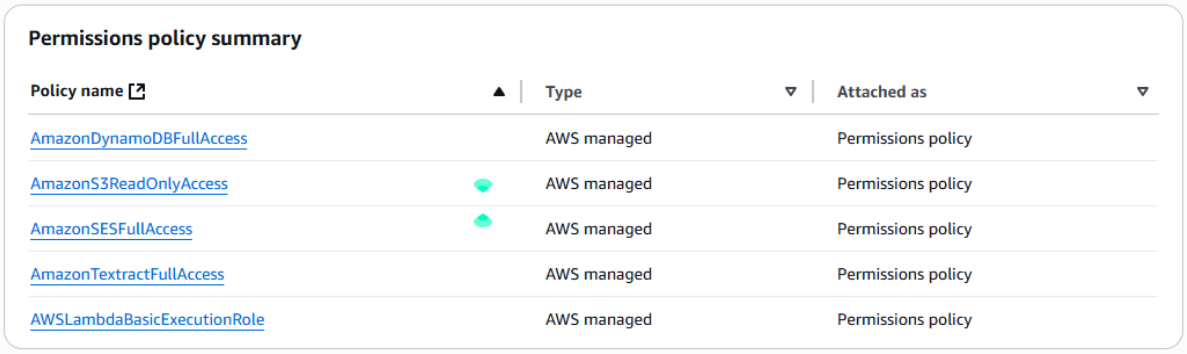
```
            items_for_db.append({
                'name': item.get('name', 'Unknown Item'),
                'price': item.get('price', '0.00'),
                'quantity': item.get('quantity', '1')
            })

        # Create item to insert
        db_item = {
            'receipt_id': receipt_data['receipt_id'],
            'date': receipt_data['date'],
            'vendor': receipt_data['vendor'],
            'total': receipt_data['total'],
            'items': items_for_db,
            's3_path': receipt_data['s3_path'],
            'processed_timestamp': datetime.now().isoformat()
        }

        # Insert into DynamoDB
        table.put_item(Item=db_item)
        print(f"Receipt data stored in DynamoDB: {receipt_data['receipt_id']}")
    except Exception as e:
        print(f"Error storing data in DynamoDB: {str(e)}")
        raise

def send_email_notification(receipt_data):
    """Send an email notification with receipt details"""
    try:
        # Format items for email
        items_html = ""
        for item in receipt_data['items']:
            name = item.get('name', 'Unknown Item')
            price = item.get('price', 'N/A')
            quantity = item.get('quantity', '1')
            items_html += f"<li>{name} - ${price} x {quantity}</li>"

        if not items_html:
            items_html = "<li>No items detected</li>"

        # Create email body
        html_body = f"""
        <html>
        <body>
            <h2>Receipt Processing Notification</h2>
            <p><strong>Receipt ID:</strong> {receipt_data['receipt_id']}</p>
            <p><strong>Vendor:</strong> {receipt_data['vendor']}</p>
```

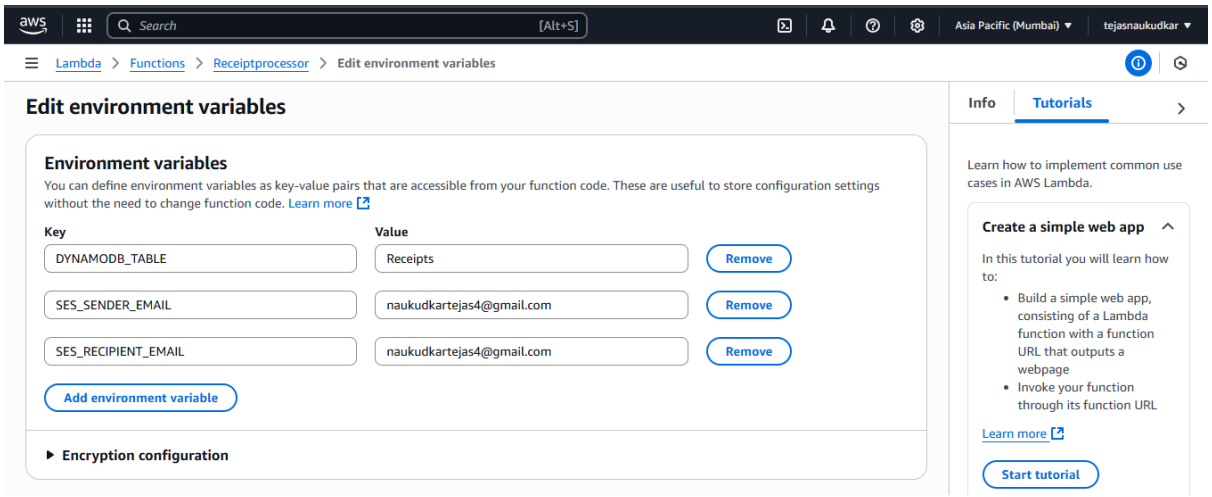
```
            <p><strong>Date:</strong> {receipt_data['date']}</p>
            <p><strong>Total Amount:</strong> ${receipt_data['total']}</p>
            <p><strong>S3 Location:</strong> {receipt_data['s3_path']}</p>

            <h3>Items:</h3>
            <ul>
                {items_html}
            </ul>

            <p>The receipt has been processed and stored in DynamoDB.</p>
        </body>
        </html>
        """

        # Send email using SES
        ses.send_email(
            Source=SES_SENDER_EMAIL,
            Destination={
                'ToAddresses': [SES_RECIPIENT_EMAIL]
            },
            Message={
                'Subject': {
                    'Data': f"Receipt Processed: {receipt_data['vendor']} -
${receipt_data['total']}"
                },
                'Body': {
                    'Html': {
                        'Data': html_body
                    }
                }
            }
        )

        print(f"Email notification sent to {SES_RECIPIENT_EMAIL}")
    except Exception as e:
        print(f"Error sending email notification: {str(e)}")
        # Continue execution even if email fails
        print("Continuing execution despite email error")
```

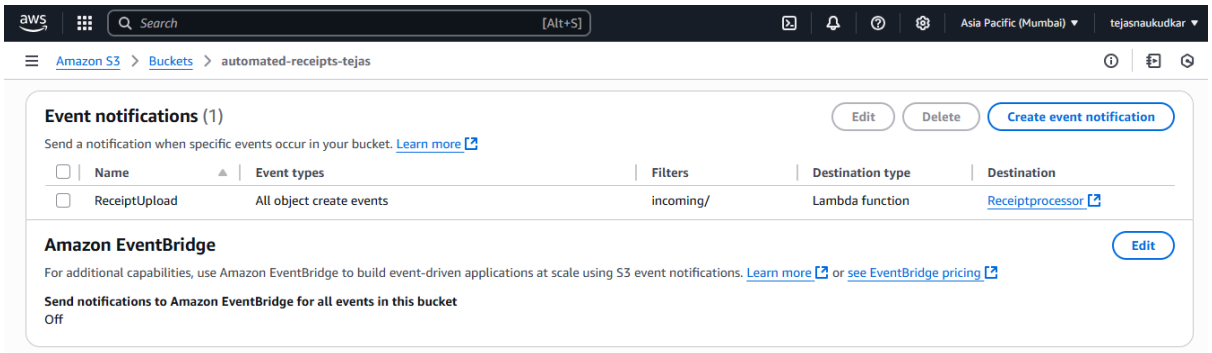
BUILDING PROCESS:



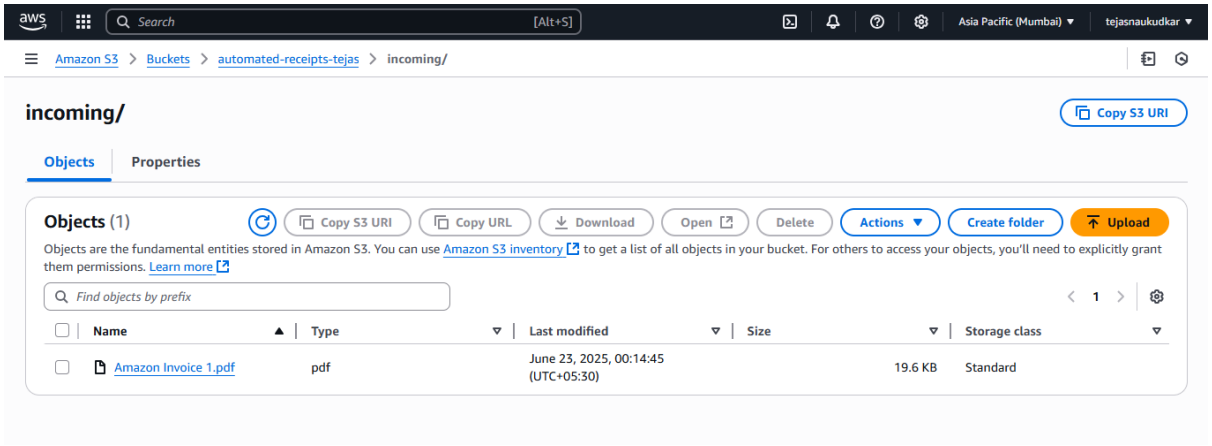


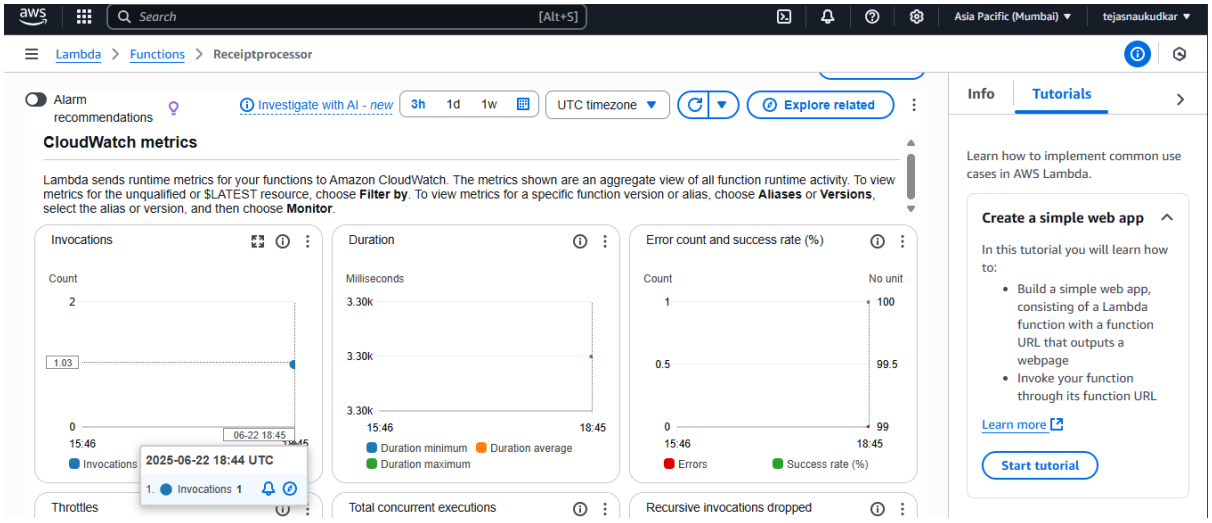


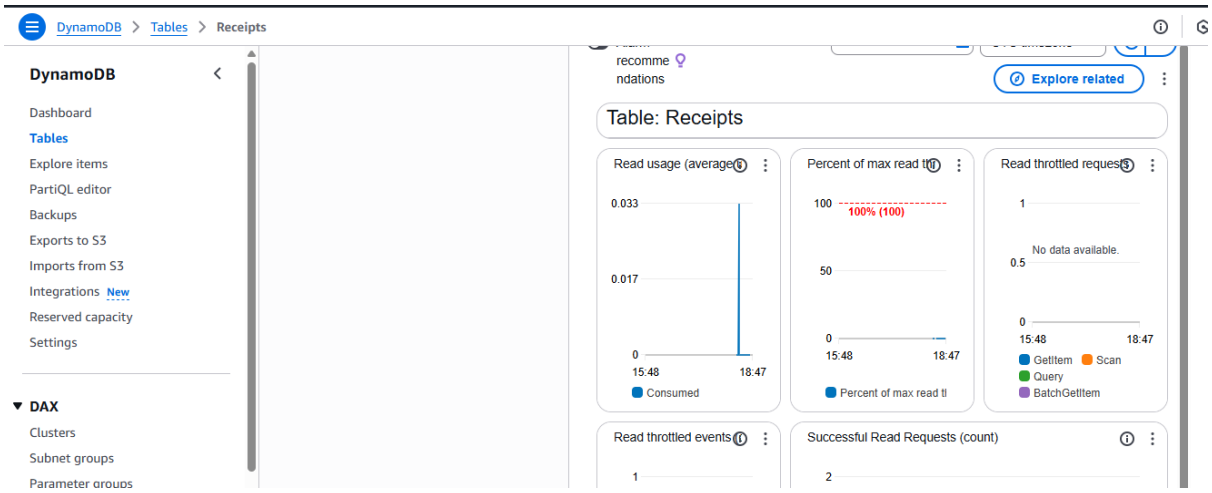

Search [Alt+S]
Asia Pacific (Mumbai)
tejasnaukudkar
Lambda > Functions > Receiptprocessor > Edit environment variables
Edit environment variables
Environment variables
You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more
Key
Value
DYNAMODB_TABLE
Receipts
Remove
SES_SENDER_EMAIL
naukudkartejas4@gmail.com
Remove
SES_RECIPIENT_EMAIL
naukudkartejas4@gmail.com
Remove
Add environment variable
Encryption configuration
Info
Tutorials
Learn how to implement common use cases in AWS Lambda.
Create a simple web app
In this tutorial you will learn how to:
Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
Invoke your function through its function URL
Learn more
Start tutorial


Search [Alt+S]
Asia Pacific (Mumbai)
tejasnaukudkar
Amazon S3 > Buckets > automated-receipts-tejas
Event notifications (1)
Edit
Delete
Create event notification
Send a notification when specific events occur in your bucket. Learn more
Name
Event types
Filters
Destination type
Destination
ReceiptUpload
All object create events
incoming/
Lambda function
Receiptprocessor
Amazon EventBridge
Edit
For additional capabilities, use Amazon EventBridge to build event-driven applications at scale using S3 event notifications. Learn more or see EventBridge pricing
Send notifications to Amazon EventBridge for all events in this bucket
Off

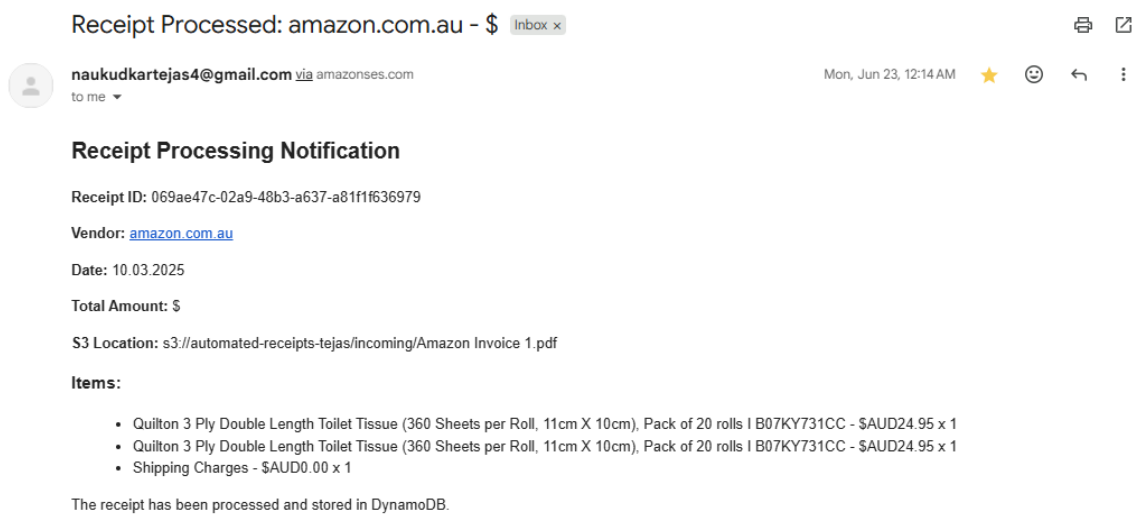
FINAL VIEW:


Search [Alt+S]
Asia Pacific (Mumbai)
tejasnaukudkar
Amazon S3 > Buckets > automated-receipts-tejas > incoming/
incoming/
Copy S3 URI
Objects
Properties
Objects (1)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
1
Name
Type
Last modified
Size
Storage class
Amazon Invoice 1.pdf
pdf
June 23, 2025, 00:14:45 (UTC+05:30)
19.6 KB
Standard


Search
[Alt+S]
Asia Pacific (Mumbai)
tejasnaukudkar
Lambda > Functions > Receiptprocessor
Alarm recommendations
Investigate with AI - new
3h 1d 1w
UTC timezone
Explore related
CloudWatch metrics
Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose Filter by. To view metrics for a specific function version or alias, choose Aliases or Versions, select the alias or version, and then choose Monitor.
Invocations
Count
2
1.03
0
15:46
06-22 18:45
Invocations
2025-06-22 18:44 UTC
1. Invocations 1
Duration
Milliseconds
3.30k
3.30k
3.30k
15:46
18:45
Duration minimum
Duration average
Duration maximum
Error count and success rate (%)
Count
No unit
1
0.5
0
100
99.5
99
15:46
18:45
Errors
Success rate (%)
Throttles
Total concurrent executions
Recursive invocations dropped
Info
Tutorials
Learn how to implement common use cases in AWS Lambda.
Create a simple web app
In this tutorial you will learn how to:
Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
Invoke your function through its function URL
Learn more
Start tutorial


DynamoDB > Tables > Receipts
DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
recommendations
Explore related
Table: Receipts
Read usage (average)
0.033
0.017
0
15:48
18:47
Consumed
Percent of max read tl
100
100% (100)
50
0
15:48
18:47
Percent of max read tl
Read throttled reques
1
No data available.
0.5
0
15:48
18:47
GetItem
Scan
Query
BatchGetItem
Read throttled events
Successful Read Requests (count)
2

Sample Output (Email Notification)



Receipt Processed: amazon.com.au - $ Inbox ×

naukudkartejas4@gmail.com via amazonses.com
to me

Mon, Jun 23, 12:14 AM

Receipt Processing Notification

Receipt ID: 069ae47c-02a9-48b3-a637-a81f1f636979

Vendor: amazon.com.au

Date: 10.03.2025

Total Amount: $

S3 Location: s3://automated-receipts-tejas/incoming/Amazon Invoice 1.pdf

Items:

- Quilton 3 Ply Double Length Toilet Tissue (360 Sheets per Roll, 11cm X 10cm), Pack of 20 rolls I B07KY731CC - $AUD24.95 x 1
- Quilton 3 Ply Double Length Toilet Tissue (360 Sheets per Roll, 11cm X 10cm), Pack of 20 rolls I B07KY731CC - $AUD24.95 x 1
- Shipping Charges - $AUD0.00 x 1

The receipt has been processed and stored in DynamoDB.

Build Time & Cost Estimate

Time Required: 2 hours
Cost: $0 (Fully AWS Free Tier eligible)

Clean-Up Process

Resource	Clean-Up Step
S3 Bucket	Delete receipt objects and bucket
Textract	Ensure no additional API calls
DynamoDB Table	Delete receipt data and remove table
SES	Unverify emails if no longer needed
IAM Roles	Detach policies and delete role