Automated Receipt Processing

Understanding the AWS Lambda Function Code:

- This section provides a detailed explanation of the Lambda function that powers the **Automated Receipt Processing System**, describing each part of the workflow and how it connects with other AWS services.

1. Code Structure Overview

The Lambda function is organized into four main components:

Component	Purpose
Lambda Handler	Entry point that coordinates the overall workflow.
Textract Processing	Extracts structured data from receipt images or PDFs.
DynamoDB Storage	Saves the processed receipt data to a database.
Email Notification	Sends formatted results to the user via Amazon SES.

2. Lambda Handler Function

```
def lambda_handler(event, context):
    try:
        # Get S3 bucket and object key from the event
        record = event['Records'][0]
        bucket = record['s3']['bucket']['name']
        key = urllib.parse.unquote_plus(record['s3']['object']['key'])

        logger.info(f"Processing file: s3://{bucket}/{key}")

        # Verify the file exists
        s3.head_object(Bucket=bucket, Key=key)

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

        # Step 2: Store the extracted data in DynamoDB
        store_receipt_in_dynamodb(receipt_data)

        # Step 3: Send notification email via SES
        send_email_notification(receipt_data)

        logger.info("Processing complete")
        return {"statusCode": 200, "body": json.dumps("Receipt processed successfully!")}

    except Exception as e:
        logger.exception("Error processing receipt")
        return {"statusCode": 500, "body": json.dumps(f"Error: {str(e)}")}
```

What It Does

- Acts as the **central controller** for the receipt processing workflow.
- Reads the uploaded file's **bucket name** and **object key** from the S3 trigger event.
- Verifies that the uploaded object exists before further processing.
- Calls three helper functions:
    1. process_receipt_with_textract() → Extracts text & data.
    2. store_receipt_in_dynamodb() → Stores structured info.
    3. send_email_notification() → Notifies the user.
- Handles errors gracefully using try-except and detailed logging.

---

3. Textract Processing Function

```
def process_receipt_with_textract(bucket, key):
    response = textract.analyze_expense(
        Document={'S3Object': {'Bucket': bucket, 'Name': key}}
    )

    receipt_id = str(uuid.uuid4())
    now = datetime.now().strftime('%Y-%m-%d')

    receipt_data = {
        'receipt_id': receipt_id,
        'date': now,
        'vendor': 'Unknown',
        'total': '0.00',
        'items': [],
        's3_path': f"s3://{bucket}/{key}"
    }

    expense_docs = response.get('ExpenseDocuments', [])
    if not expense_docs:
        logger.warning("No ExpenseDocuments found")
        return receipt_data

    doc = expense_docs[0]

    # Extract key fields like TOTAL, DATE, VENDOR
    for field in doc.get('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 in ('INVOICE_RECEIPT_DATE', 'DATE'):
            receipt_data['date'] = value
        elif field_type in ('VENDOR_NAME', 'SUPPLIER_NAME'):
            receipt_data['vendor'] = value

    # Extract line items (Item name, quantity, price)
    for group in doc.get('LineItemGroups', []):
        for line_item in group.get('LineItems', []):
            item = {}
            for f in line_item.get('LineItemExpenseFields', []):
```

```
            val = f.get('ValueDetection', {}).get('Text', '')
            if f_type == 'ITEM':
                item['name'] = val
            elif f_type == 'PRICE':
                item['price'] = val
            elif f_type == 'QUANTITY':
                item['quantity'] = val
        if 'name' in item:
            item.setdefault('price', '0.00')
            item.setdefault('quantity', '1')
            receipt_data['items'].append(item)

    return receipt_data
```

What It Does

- Uses **Amazon Textract's analyze_expense API** to automatically detect structured information in receipts and invoices.
- Creates a **unique ID** for each processed receipt.
- Extracts key summary fields such as:
  - Vendor name
  - Invoice/receipt date
  - Total amount
- Collects individual **line items** (product name, quantity, price).
- Returns all data in a clean, structured dictionary.

Key Insights

- Textract recognizes **semantic structures**, not just text.
- Missing data is handled gracefully using default values.
- Each receipt is uniquely traceable via its receipt_id.

---

4. DynamoDB Storage Function

```
def store_receipt_in_dynamodb(receipt_data):
    table = dynamodb.Table(DYNAMODB_TABLE)
    table.put_item(Item={
        'receipt_id': receipt_data['receipt_id'],
        'date': receipt_data['date'],
        'vendor': receipt_data['vendor'],
        'total': receipt_data['total'],
        'items': receipt_data['items'],
        's3_path': receipt_data['s3_path'],
        'processed_timestamp': datetime.now().isoformat()
    })
```

What It Does

- Connects to the **DynamoDB** table specified in the environment variable.
- Saves all structured receipt data in a single database record.
- Adds a **timestamp** for when the processing occurred.
- Keeps the **S3 path** for traceability to the original file.

Key Insights

- Data is easily queryable using the receipt_id key.
- The timestamp helps in tracking and debugging.
- The structure allows future analytics (e.g., total spend per vendor).

---

5. Email Notification Function

```
def send_email_notification(receipt_data):
    items_html = "".join(
        f"<li>{i.get('name','Unknown')} - ${i.get('price','0.00')} × {i.get('quantity','1')}</li>"
        for i in receipt_data.get('items', [])
    ) or "<li>No items detected</li>"

    html_body = f"""
    <html><body>
      <h2>Receipt Processed</h2>
      <p><strong>Vendor:</strong> {receipt_data['vendor']}</p>
      <p><strong>Date:</strong> {receipt_data['date']}</p>
      <p><strong>Total:</strong> ${receipt_data['total']}</p>
      <p><strong>Receipt ID:</strong> {receipt_data['receipt_id']}</p>
      <p><strong>S3 Path:</strong> {receipt_data['s3_path']}</p>
      <h3>Items</h3>
      <ul>{items_html}</ul>
    </body></html>
    """

    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}}
        }
    )
```

What It Does

- Creates a well-formatted **HTML email** summarizing the extracted data.
- Lists all identified **line items** (products, quantities, and prices).
- Sends the email using **Amazon SES**.
- Provides a **direct reference** to the S3 location and receipt ID.

Key Insights

- HTML format improves readability.
- The email can serve as an automated audit trail.
- It provides confirmation that processing completed successfully.

---

6. Error Handling and Logging

- The code uses try-except blocks for every major operation.
- Uses AWS CloudWatch logs to capture detailed messages at each step.
- Gracefully handles missing data, malformed receipts, and temporary service issues.
- Ensures that failures in non-critical steps (like email sending) don’t break the overall process.

7. Environment Variables

Variable	Description
DYNAMODB_TABLE	Name of the DynamoDB table to store processed data.
SES_SENDER_EMAIL	Verified sender address for Amazon SES.
SES_RECIPIENT_EMAIL	Email recipient for notifications.
SES_REGION	AWS region where SES is configured.

This allows flexible reconfiguration without modifying the Lambda code itself.

8. Summary

Stage	Service	Purpose
1. Upload	Amazon S3	Stores receipt files (PDF/JPEG).
2. Trigger	AWS Lambda	Processes events automatically.
3. Extract	Amazon Textract	Reads and interprets receipt data.
4. Store	DynamoDB	Saves structured data.
5. Notify	Amazon SES	Sends confirmation and summary email.

Thank you for visiting