Understand function

🔍 Understanding the AWS Lambda Function Code

This section breaks down the AWS Lambda function responsible for automating receipt processing. It highlights each major component, what it does, and why it matters.

📂 Code Structure Overview

The Lambda function is logically divided into four main components:

1. **Lambda Handler** – The entry point that manages the entire workflow.
2. **Textract Processing** – Extracts structured data from uploaded receipt images.
3. **DynamoDB Storage** – Saves the parsed data into a DynamoDB table.
4. **Email Notification** – Sends a detailed email summary of the processed receipt.

🧩 Lambda Handler Function

```
def lambda_handler(event, context):

    try:

        # Extract S3 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'])



        # Confirm the file exists in S3

        s3.head_object(Bucket=bucket, Key=key)



        # Step 1: Analyze receipt using Textract

        receipt_data = process_receipt_with_textract(bucket, key)



        # Step 2: Save data in DynamoDB

        store_receipt_in_dynamodb(receipt_data, bucket, key)
```

```
        # Step 3: Send an 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)}')
        }
```

Purpose:

- Acts as the controller for the whole process
- Decodes the file path and verifies file presence
- Delegates each task to dedicated functions
- Provides clear logging and error handling

🧾 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())
    receipt_data = {
```

```
    'receipt_id': receipt_id,
    'date': datetime.now().strftime('%Y-%m-%d'),
    'vendor': 'Unknown',
    'total': '0.00',
    'items': [],
    's3_path': f"s3://{bucket}/{key}"
}

if 'ExpenseDocuments' in response and response['ExpenseDocuments']:
    expense_doc = response['ExpenseDocuments'][0]

    for field in expense_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 == 'INVOICE_RECEIPT_DATE':
            receipt_data['date'] = value
        elif field_type == 'VENDOR_NAME':
            receipt_data['vendor'] = value

    for group in expense_doc.get('LineItemGroups', []):
        for line_item in group.get('LineItems', []):
            item = {}
            for field in line_item.get('LineItemExpenseFields', []):
                ftype = field.get('Type', {}).get('Text', '')
```

```
            fvalue = field.get('ValueDetection', {}).get('Text', '')
            if ftype == 'ITEM':
                item['name'] = fvalue
            elif ftype == 'PRICE':
                item['price'] = fvalue
            elif ftype == 'QUANTITY':
                item['quantity'] = fvalue
        if 'name' in item:
            receipt_data['items'].append(item)

    return receipt_data
```

Key Functions:

- Leverages **Textract's analyze_expense API** to read structured receipt data
- Assigns a **unique receipt ID** for tracking
- Parses essential fields like **date, vendor, and total**
- Collects individual **line items** with prices and quantities
- Returns a structured and normalized data format

🗃️ DynamoDB Storage Function

```
def store_receipt_in_dynamodb(receipt_data, bucket, key):
    table = dynamodb.Table(DYNAMODB_TABLE)
    items_for_db = [
        {
            'name': item.get('name', 'Unknown Item'),
            'price': item.get('price', '0.00'),
            'quantity': item.get('quantity', '1')
        } for item in receipt_data['items']
    ]
```

```
    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()
    }

    table.put_item(Item=db_item)
```

What it does:

- Connects to the configured **DynamoDB table**
- Transforms receipt data into a compatible format
- Stores the **entire receipt**, including itemized data and timestamp
- Maintains an **S3 reference** for traceability

📧 Email Notification Function

```
def send_email_notification(receipt_data):
    items_html = "".join([
        f"<li>{item.get('name', 'Unknown Item')} – ${item.get('price', 'N/A')} x {item.get('quantity', '1')}</li>"
        for item in receipt_data['items']
    ])

    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>
  """

  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}}
    }
  )
```

Purpose:

- Builds a **clean HTML email** summarizing the processed receipt
- Includes all extracted **line items**
- Sends the notification via **Amazon SES**
- Adds useful context such as **S3 path and receipt ID** for future reference

🛠️ Error Handling and Logging

Across the code, good practices are followed:

- **Try/except blocks** for controlled failure and logging
- **Clear error messages** to aid debugging
- **Default values** ensure the system doesn't break on missing fields
- **Partial recovery:** If non-critical steps fail (like email), the rest still proceeds

⚙️ Environment Variables

Used for flexibility and clean configuration:

- DYNAMODB_TABLE – Target DynamoDB table name
- SES_SENDER_EMAIL – Verified sender email for SES
- SES_RECIPIENT_EMAIL – Email recipient for notifications

Advantage: These settings can be changed without touching the code.