

#textract

sending to textract via s3

Implementation Steps

1. Create SNS Topic & SQS Queue

- SNS topic for Textract completion notifications
- SQS queue subscribed to SNS topic

2. Update IAM Role

- Add Textract, SNS, SQS permissions

3. Create Lambda Functions

- textract-starter : Initiates StartDocumentAnalysis for each PDF
- textract-processor : Triggered by SQS, retrieves results, saves to S3

4. Create Step Functions State Machine (if Option 1)

- Orchestrates batch processing
- Handles retries and errors

5. Set up S3 Output Structure

- processed/batch-1/document-1.json
- processed/batch-2/document-2.json

⚠ Important Limitations

1. **150 documents per bulk upload** - You're already aligned with this!
2. **Job results stored for 7 days** - Make sure to retrieve within this window
3. **Processing time**: Multi-page documents can take 5-30 seconds each
4. **Concurrent job limits**: AWS may throttle if too many concurrent requests

Perfect! Based on the AWS documentation you provided, here's a detailed implementation plan for **Option 2: Lambda + SQS** approach for your batch Textract processing:

📐 Refined Architecture Based on AWS Docs

```
S3 Bucket: important-stuff-20251106-0-018ql4
├── batch-1/ (150 PDFs)
├── batch-2/ (150 PDFs)
└── ...
```



🔑 Key Differences from AWS Docs Example

The documentation shows **DetectDocumentText** (synchronous, text only), but you need:

- **StartDocumentAnalysis** (asynchronous, supports FORMS + TABLES)
- **GetDocumentAnalysis** (retrieves results)



Detailed Implementation Plan

Lambda 1: Batch Initiator

Based on your existing `lambda_pdf_organizer.py` structure:

```

import boto3
import json
from datetime import datetime

textract_client = boto3.client('textract')

```

```

s3_client = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('textract-jobs') # We'll create this

def lambda_handler(event, context):
    """
    Initiates Textract processing for all PDFs in a batch folder.

    Event format:
    {
        "bucket_name": "important-stuff-20251106-0-018ql4",
        "batch_prefix": "batch-1/" # Which batch to process
    }
    """

    bucket_name = event['bucket_name']
    batch_prefix = event['batch_prefix']

    # SNS topic ARN for completion notifications
    sns_topic_arn = event['sns_topic_arn']
    role_arn = event['textract_role_arn'] # IAM role for Textract to
    publish to SNS

    # List all PDFs in the batch folder
    response = s3_client.list_objects_v2(
        Bucket=bucket_name,
        Prefix=batch_prefix
    )

    pdf_files = [obj['Key'] for obj in response.get('Contents', [])
                 if obj['Key'].endswith('.pdf')]

    jobs_started = []

    for pdf_key in pdf_files:
        try:
            # Start asynchronous document analysis
            response = textract_client.start_document_analysis(
                DocumentLocation={
                    'S3Object': {
                        'Bucket': bucket_name,
                        'Name': pdf_key
                    }
                },
                FeatureTypes=['FORMS', 'TABLES'], # Extract key-value pairs
                + tables
                OutputConfig={


```

```

        'S3Bucket': bucket_name,
        'S3Prefix': f'processed/{batch_prefix}'
    },
    NotificationChannel={
        'SNSTopicArn': sns_topic_arn,
        'RoleArn': role_arn
    }
)

job_id = response['JobId']

# Store job mapping in DynamoDB for tracking
table.put_item(Item={
    'JobId': job_id,
    'SourceKey': pdf_key,
    'Bucket': bucket_name,
    'BatchPrefix': batch_prefix,
    'Status': 'IN_PROGRESS',
    'StartTime': datetime.utcnow().isoformat()
})

jobs_started.append({
    'pdf': pdf_key,
    'job_id': job_id
})

except Exception as e:
    print(f"Error processing {pdf_key}: {str(e)}")

return {
    'statusCode': 200,
    'body': json.dumps({
        'message': f'Started {len(jobs_started)} Textract jobs',
        'batch': batch_prefix,
        'jobs': jobs_started
    })
}

```

Lambda 2: Result Processor

```

import boto3
import json

textract_client = boto3.client('textract')
s3_client = boto3.client('s3')

```

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('textract-jobs')

def lambda_handler(event, context):
    """
    Triggered by SQS messages from SNS topic.
    Retrieves Textract results and saves to S3.
    """

    for record in event['Records']:
        # Parse SNS message from SQS
        sns_message = json.loads(record['body'])
        message = json.loads(sns_message['Message'])

        job_id = message['JobId']
        status = message['Status']

        if status != 'SUCCEEDED':
            print(f"Job {job_id} failed with status: {status}")
            continue

        try:
            # Get job details from DynamoDB
            response = table.get_item(Key={'JobId': job_id})
            job_data = response['Item']

            # Retrieve Textract results
            result = textract_client.get_document_analysis(JobId=job_id)

            # Extract blocks (contains text, forms, tables)
            blocks = result['Blocks']

            # Handle pagination if document has many pages
            while 'NextToken' in result:
                result = textract_client.get_document_analysis(
                    JobId=job_id,
                    NextToken=result['NextToken']
                )
                blocks.extend(result['Blocks'])

            # Parse key-value pairs and text
            extracted_data = parse_textract_response(blocks)

            # Save to S3
            filename = job_data['SourceKey'].split('/')[-1].replace('.pdf',
'.json')
            s3 = boto3.client('s3')
            s3.put_object(Bucket='your-bucket-name', Key=filename, Body=json.dumps(extracted_data))

        except Exception as e:
            print(f"Error processing job {job_id}: {e}")

    return {
        'statusCode': 200,
        'body': 'Success'
    }
```

```

        output_key = f"processed/{job_data['BatchPrefix']}{filename}"

        s3_client.put_object(
            Bucket=job_data['Bucket'],
            Key=output_key,
            Body=json.dumps(extracted_data, indent=2)
        )

        # Update DynamoDB
        table.update_item(
            Key={'JobId': job_id},
            UpdateExpression='SET #status = :status, OutputKey = :output',
            ExpressionAttributeNames={'#status': 'Status'},
            ExpressionAttributeValues={
                ':status': 'COMPLETED',
                ':output': output_key
            }
        )

    print(f"Successfully processed {job_data['SourceKey']}")

except Exception as e:
    print(f"Error processing job {job_id}: {str(e)}")

return {'statusCode': 200}

def parse_textract_response(blocks):
    """
    Extracts raw text, key-value pairs, and tables from Textract blocks.
    """

    extracted = {
        'raw_text': [],
        'key_value_pairs': [],
        'tables': []
    }

    # Build block map for relationships
    block_map = {block['Id']: block for block in blocks}

    for block in blocks:
        if block['BlockType'] == 'LINE':
            extracted['raw_text'].append({
                'text': block['Text'],
                'confidence': block['Confidence']
            })

```

```

        elif block['BlockType'] == 'KEY_VALUE_SET':
            if 'KEY' in block.get('EntityTypes', []):
                key_text = get_text_from_relationship(block, block_map)
                value_block = get_value_block(block, block_map)
                value_text = get_text_from_relationship(value_block,
                block_map) if value_block else ''
                extracted['key_value_pairs'].append({
                    'key': key_text,
                    'value': value_text,
                    'confidence': block['Confidence']
                })
        return extracted

def get_text_from_relationship(block, block_map):
    """Helper to extract text from CHILD relationships."""
    text = ''
    if 'Relationships' in block:
        for relationship in block['Relationships']:
            if relationship['Type'] == 'CHILD':
                for child_id in relationship['Ids']:
                    child = block_map.get(child_id)
                    if child and child['BlockType'] == 'WORD':
                        text += child['Text'] + ' '
    return text.strip()

def get_value_block(key_block, block_map):
    """Helper to find VALUE block for a KEY."""
    if 'Relationships' in key_block:
        for relationship in key_block['Relationships']:
            if relationship['Type'] == 'VALUE':
                for value_id in relationship['Ids']:
                    return block_map.get(value_id)
    return None

```

🛠 Infrastructure Components to Create

1. DynamoDB Table (for job tracking)

```

aws dynamodb create-table \
--table-name textract-jobs \
--attribute-definitions AttributeName=JobId,AttributeType=S \
--key-schema AttributeName=JobId,KeyType=HASH \

```

```
--billing-mode PAY_PER_REQUEST \
--profile saba
```

2. SNS Topic

```
aws sns create-topic \
--name textract-completion-topic \
--profile saba
```

3. SQS Queue

```
aws sqs create-queue \
--queue-name textract-results-queue \
--profile saba
```

4. Subscribe SQS to SNS

```
aws sns subscribe \
--topic-arn arn:aws:sns:us-east-1:ACCOUNT_ID:texttract-completion-topic \
--protocol sqs \
--notification-endpoint arn:aws:sqs:us-east-1:ACCOUNT_ID:texttract-
results-queue \
--profile saba
```

5. IAM Role for Textract (to publish to SNS)

This role allows Textract to send completion notifications to SNS.

Batch Processing Flow

To process all 4000 PDFs:

```
# Process each batch sequentially or in parallel
for i in {1..27}; do
  aws lambda invoke \
    --function-name textract-batch-initiator \
    --payload "{\"bucket_name\": \"important-stuff-20251106-0-018ql4\", \
    \"batch_prefix\": \"batch-$i/\", \"sns_topic_arn\": \"arn:...\", \
    \"extract_role_arn\": \"arn:...\"}" \
    --profile saba \
  response.json
```

```
sleep 60 # Wait 1 minute between batches to avoid throttling
done
```

IAM Permissions Summary

Your Lambda functions need:

Lambda 1 (Initiator):

- `textract:StartDocumentAnalysis`
- `s3:GetObject , s3>ListBucket`
- `dynamodb:PutItem`

Lambda 2 (Processor):

- `textract:GetDocumentAnalysis`
- `s3:PutObject`
- `dynamodb:.GetItem , dynamodb:UpdateItem`
- `sqs:ReceiveMessage , sqs>DeleteMessage`

Textract Service Role:

- `sns:Publish` (to send completion notifications)