# Serverless Invoice Scanner on AWS Project Documentation

This document explains end-to-end implementation of a Serverless Invoice Scanner using AWS services. It includes frontend, backend, permissions, common errors, and fixes.

---

## 1. Project Overview

The Serverless Invoice Scanner allows users to: - Upload an invoice from a web frontend - Store the invoice securely in S3 - Extract text from the invoice using Amazon Textract - Save extracted data into DynamoDB

### Architecture Flow

Frontend (S3 Static Website) → API Gateway (HTTP API) → AWS Lambda (Python) → S3 (Invoice Upload Bucket) → Amazon Textract → DynamoDB

---

## 2. AWS Services Used



- Amazon S3 (Static Website + Invoice Storage)
- API Gateway (HTTP API)
- AWS Lambda (Python)
- Amazon Textract (OCR)
- Amazon DynamoDB (Data storage)
- IAM (Permissions)
- CloudWatch (Logs & debugging)

---

## 3. S3 Buckets Setup

### 3.1 Frontend Bucket (Static Website)

Purpose: Host HTML, CSS, JavaScript

Example bucket name:

invoice-frontend-ui-001

Steps: 1. Create S3 bucket 2. Enable Static website hosting 3. Upload index.html 4. Keep bucket public read only for website access

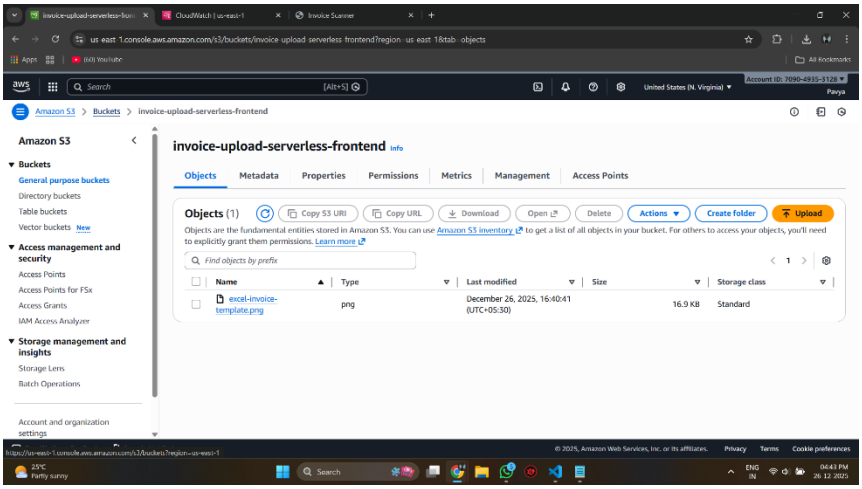
---

## 3.2 Invoice Upload Bucket (Backend)

Purpose: Store uploaded invoice files

Example bucket name: invoice-upload-serverless-frontend

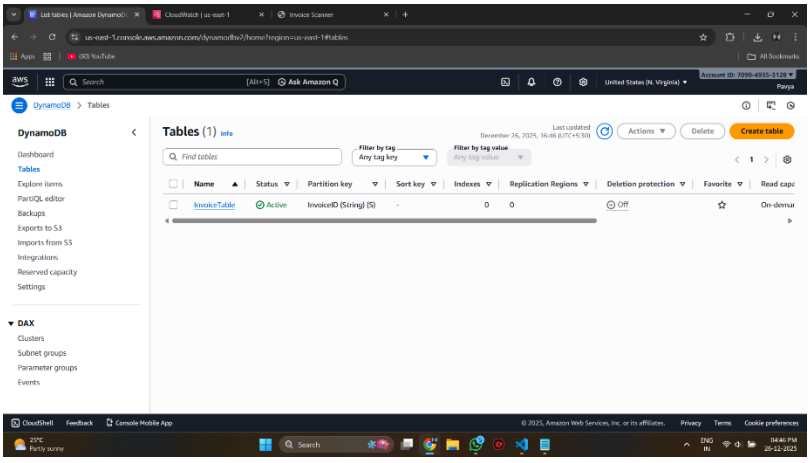
Steps: 1. Create S3 bucket 2. Keep it private 3. Same region as Lambda and API Gateway






# 4. DynamoDB Setup

1. Create table
2. Table name: InvoiceTable
3. Partition key: InvoiceID (String)



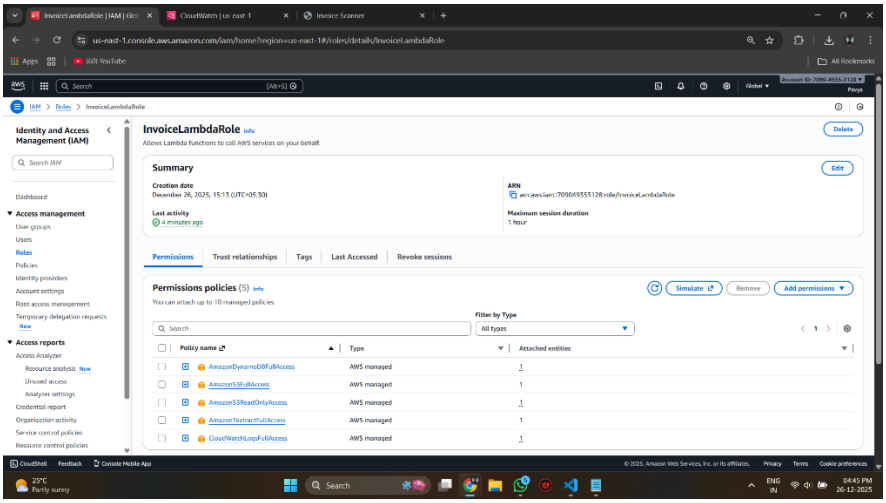

# 5. IAM Role & Permissions

## 5.1 Lambda Execution Role

Go to: Lambda → Configuration → Permissions → Execution role

Attach the following policies:

- AmazonS3FullAccess
- AmazonTextractFullAccess
- AmazonDynamoDBFullAccess
- AWSLambdaBasicExecutionRole




## 5.2 Minimal S3 Upload Permission (Optional Custom Policy)



```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "s3:PutObject",
   "Resource": "arn:aws:s3:::invoice-upload-serverless-frontend/*"
  }
 ]
}
```

---

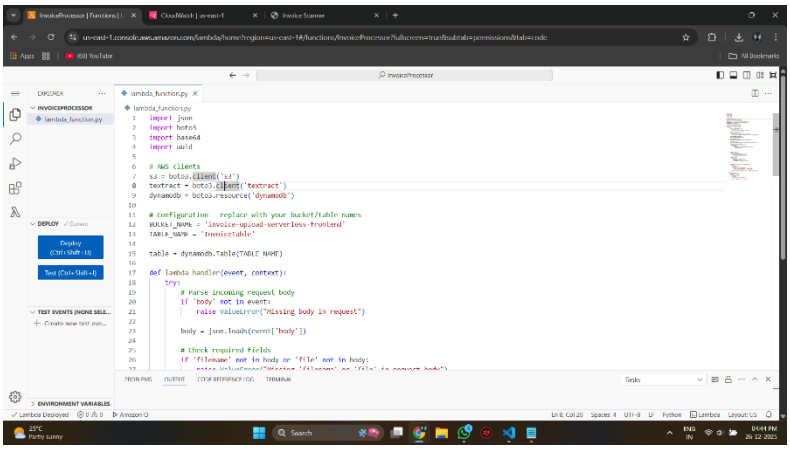
# 6. Lambda Function Setup

## Runtime

Python 3.11

## Lambda Code

The Lambda function: - Accepts base64 file - Uploads to S3 - Uses Textract - Saves result in DynamoDB - Returns CORS headers




---

# 7. API Gateway Setup

## 7.1 Create HTTP API

1. Go to API Gateway
2. Create HTTP API
3. Integration: Lambda function
4. Route:

POST /upload



## 7.2 Enable CORS

- Allow origins:

*

- Allow headers:

Content-Type

- Allow methods:

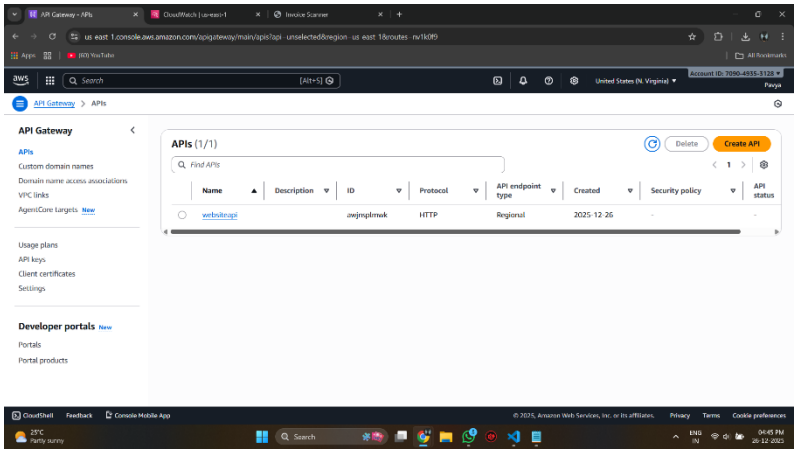
POST, OPTIONS

## 7.3 Invoke URL

Example:

https://abcd1234.execute-api.us-east-1.amazonaws.com/upload

API endpoint URL = Invoke URL




---

# 8. Frontend Integration

## index.html (Simplified Upload Logic)

- Reads file using FileReader
- Converts to base64
- Sends POST request to API Gateway /upload

Make sure the API URL is correct and current.



---

# 9. Common Errors & Fixes

## 9.1 CORS Error

Error:

No 'Access-Control-Allow-Origin' header

Fix: - Enable CORS in API Gateway - Return CORS headers from Lambda

---

## 9.2 Internal Server Error

Cause: - Lambda crash

Fix: - Check CloudWatch logs - Fix syntax / permission issues

---

## 9.3 Runtime.UserCodeSyntaxError

Cause: - Python syntax error in Lambda

Fix: - Use clean, tested Lambda code - Deploy after every change

---

### 9.4 API ID Not Found

Error:

The API with ID xxxx could not be found

Fix: - Use correct Invoke URL - Update frontend with new API ID

---

## 10. Debugging & Logs

- Go to CloudWatch → Log groups
- Open:

/aws/lambda/<FunctionName>

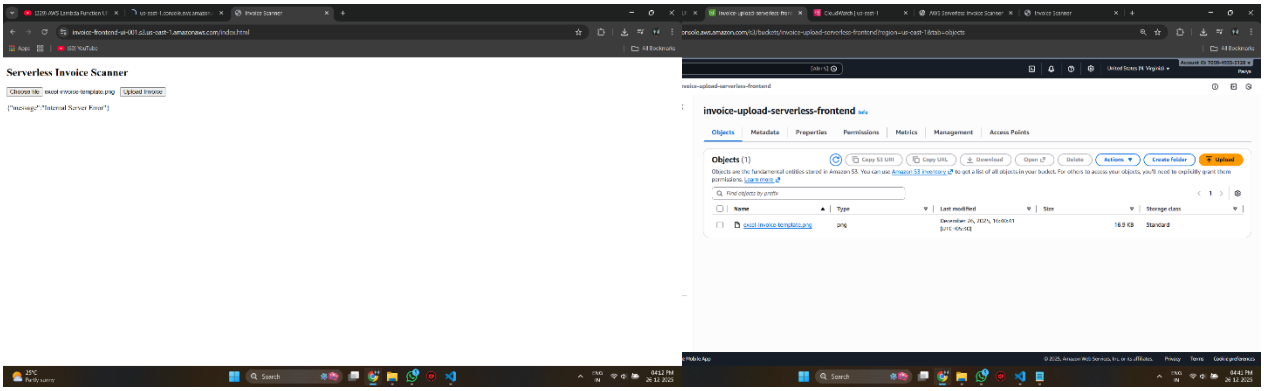
- Look for Traceback / Exception

---

## 11. Final Verification Checklist



- Frontend loads from S3
- API Gateway invoke URL works
- Lambda executes without error
- Invoice appears in upload S3 bucket
- Extracted text saved in DynamoDB




---

## 12. Interview Explanation

> “I built a serverless invoice scanner using AWS S3, API Gateway, Lambda, Textract, and DynamoDB. The frontend uploads invoices to an API endpoint, Lambda stores them in S3, extracts text using Textract, and saves results in DynamoDB. The entire system is fully serverless, scalable, and cost-efficient.”

---

## 13. Conclusion

This project demonstrates: - Serverless architecture - AWS IAM & security - OCR using Textract - Full-stack cloud integration