AWS Serverless File Processing System

Automated File Uploads → Metadata Extraction → DynamoDB Storage

Author: Shalini Baghel

1. Project Overview

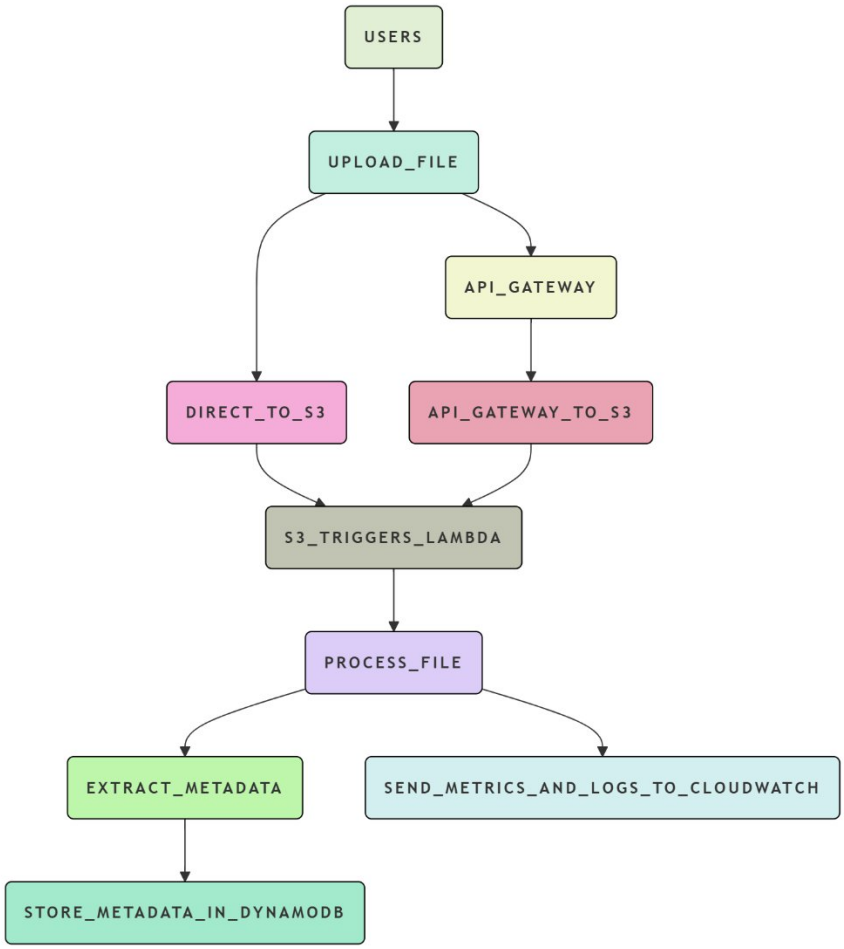
A serverless architecture that automates:

- **File uploads** via API Gateway
- **Storage** in Amazon S3
- **Metadata extraction** using AWS Lambda
- **Structured storage** in DynamoDB
- **Monitoring** via CloudWatch

Use Case:

- Document management systems
- Asset tracking for media files
- Audit logging for uploads

2. Architecture Diagram



3. Step-by-Step Implementation

3.1 S3 Bucket Setup

Purpose: Store uploaded files and trigger Lambda.

Steps:

1. **Create Bucket:**
   - Name: **my-serverless-uploads** (Globally unique)
   - Region: `ap-south-1` (Example)
2. **Enable Event Notifications:**
   - **Event Type:** `PUT` (Trigger on uploads)

- **Destination:** Lambda function `FilesProcessed`

3.2 IAM Role for Lambda

Purpose: Grant Lambda permissions to access S3, DynamoDB, and CloudWatch.

Steps:

1. **Create Role:**
   - **Trusted Entity:** `AWS Lambda`
2. **Attach Policies:**
   - `AmazonS3FullAccess`
   - `AmazonDynamoDBFullAccess`
   - `CloudWatchLogsFullAccess`

Sample Policy (JSON):

json

Copy

Download

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::your-bucket-name/*"
    }
  ]
}
```

3.3 Lambda Function

Purpose: Process S3 uploads and save metadata to DynamoDB.

Configuration:

- **Runtime:** Python 3.10
- **Handler:** `lambda_function.lambda_handler`
- **Environment Variable:**
  - Key: `DYNAMODB_TABLE_NAME`
  - Value: `FileMetadata`

Code (`lambda_function.py`):

python

Copy

Download

```python
import json
import boto3
import os
from datetime import datetime

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table(os.environ['DYNAMODB_TABLE_NAME'])

def lambda_handler(event, context):
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    file_name = event['Records'][0]['s3']['object']['key']
    file_size = event['Records'][0]['s3']['object']['size']
    upload_time = datetime.now().isoformat()

    try:
        table.put_item(
            Item={
                'filename': file_name,
                'size': file_size,
                'upload_time': upload_time,
                'bucket_name': bucket_name
            }
        )
```

```
        return {'statusCode': 200, 'body': 'Metadata saved!'}
    except Exception as e:
        return {'statusCode': 500, 'body': str(e)}
```

3.4 DynamoDB Table

Purpose: Store file metadata.

Configuration:

- **Table Name:** `FileMetadata`
- **Primary Key:** `filename` (String)

Sample Item:

json

Copy

Download

```
{
  "filename": "example.pdf",
  "size": 1024,
  "upload_time": "2025-05-04T12:00:00.000Z",
  "bucket_name": "my-serverless-uploads"
}
```

3.5 API Gateway Setup

Purpose: Accept HTTP uploads and forward to S3.

Steps:

1. **Create REST API:**
   - Name: `FileUploadAPI`
2. **Add Resource:** `/upload`

3. **Add POST Method:**
   - **Integration Type:** AWS Service
   - **Service:** S3
   - **Action:** PutObject
   - **Bucket ARN:** arn:aws:s3::: my-serverless-uploads/*

3.6 CloudWatch Monitoring

Purpose: Track Lambda executions and errors.

Key Metrics:

- Invocation count
- Error rates
- Execution duration

Alarms (Optional):

- Trigger email/SMS if errors exceed 5/minute.

4. Testing the System

1. **Upload a File:**
   - Open Postman
   - Select POST method

   #In the URL, paste your API Gateway Invoke URL (something like):

   bash

   "https://abcd1234.execute-api.us-east-1.amazonaws.com/dev/upload?filename=test.txt"

- Click on Body tab
- Select binary
- Click Select File and choose a sample .txt, .png, etc.
- Hit Send

2. **Verify:**

- File appears in S3.
- Lambda Processed.
- DynamoDB has metadata entry.
- CloudWatch logs show Lambda execution.

5. Troubleshooting

Issue	Solution
Lambda not triggered	Check S3 event notification setup
DynamoDB write fails	Verify IAM permissions
API Gateway 403 error	Ensure `s3:PutObject` permission

6. Cleanup (Optional)

To avoid AWS charges:

1. Delete S3 bucket.
2. Remove Lambda function.
3. Delete DynamoDB table.
4. Delete API Gateway created.

5. (Optional) Delete Cloudwatch “Lambda log Groups”.

Attachments (In Word):

1. Full IAM policy JSON.
2. Postman collection for API testing.