**Title**: Automated Data Extraction Pipeline using AWS Services and Python

**Introduction:**

In today's fast-paced world, manual extraction of valuable data from documents is time-consuming. To address this challenge, we are developing an automated pipeline that harnesses AWS services and Python scripts to extract essential information from documents. The entire process aims to be seamless, utilizing the power of AWS to efficiently extract loan numbers and policy numbers from documents and store them in a database.

**Aim and Objectives:**

The project's primary goal is to establish an Automated Pipeline that triggers in response to specific events. It will employ AWS services and Python scripts to extract loan numbers and policy numbers from documents and store them in a database. The key objectives are as follows:

1. Implement a process to push data from DMS/Internal boss portal to an S3 bucket using a PUT API.

2. Design a Step Function to be invoked by S3 events, which will process the data using a series of Lambda functions.

3. Develop Lambda functions to extract data from documents, create separate pages from PDFs, and identify relevant fields.

4. Enable a GET API to check the status of the process and retrieve stored data.

**Architecture:**

A diagram of software development

Description automatically generated

1. PUT API: Created using API Gateway, this API allows data to be pushed to the S3 bucket. A Lambda function (File\_Uploader\_Lambda) is responsible for creating a folder in the S3 bucket, uploading the file, generating a unique ID, and storing it in DynamoDB.

2. Step Function: Triggered by S3 events, the Step Function comprises two Lambdas. The first Lambda extracts data from the uploaded file and creates separate pages from PDFs using the PyPDF4 library. The second Lambda iterates through these pages, analyze the text, identifies relevant fields using regular expressions, and stores the data in DynamoDB.

3. GET\_Status API: This API is used to query the status of the extraction process and retrieve stored data. The GET\_Status\_Lambda processes the request and provides responses based on the status of the process and the extracted fields.

**Implementation:**

**Step 1**: Pushing Data to S3 Bucket via PUT API

• A PUT API was created using API Gateway and tested using Postman.

• The API push the data to the Lambda.

• The File\_Uploader\_Lambda creates a folder(filename) in the bucket, uploads the file, generates a unique ID, and stores it in DynamoDB along with the current processing stage.

DEMO Picture of input and output: -

A screenshot of a computer

Description automatically generated

**Step 2**: Step Function Processing

• AWS Event Bridge is enabled on the S3 bucket to trigger the Step Function upon file upload.

• The Step Function orchestrates two Lambdas.

• First Lambda: Extracts data from the uploaded file and creates separate PDF pages using PyPDF4.

• Second Lambda: Iterates through the pages, analyze text, identifies fields using regular expressions, and stores them in DynamoDB.

**Step 3**: GET\_Status API

• A GET\_Status API is created to query the process status and retrieve stored data.

• The GET\_Status\_Lambda processes the request, retrieves data associated with the unique ID, and responds with the status of the process and stored fields.

DEMO Picture of input and output: -

A screenshot of a computer

Description automatically generated

**Conclusion:**

The Automated Data Extraction Pipeline seamlessly integrates AWS services and Python scripts to streamline the process of extracting loan numbers and policy numbers from documents. By utilizing the power of AWS Lambda, DynamoDB, and other services, the pipeline effectively automates a previously time-consuming task, saving valuable time and resources while ensuring accuracy and efficiency in data extraction.