Enterprise Content Management and Document AI Processing in the Cloud

By Kevin Ng, Akash Patel, Harshil Sharma, Gagan Shrivastava, Kusuma Yalaka

A project done for NJIT's CS491 Capstone Program with New Jersey Courts.

The objective is to create a content management system for documents with AI solutions to aid in the process. A user would upload documents and be able to select functions to perform on these documents.

These functions include:

- Document conversion from a word document or PDF to PDF/A format
- Strip metadata from a PDF/Word file
- Identify and Redact personally identifiable information from documents
- Extract text and store into ElasticSearch
- Virus Scanning
- Sentiment Analysis on Documents with Judge comments

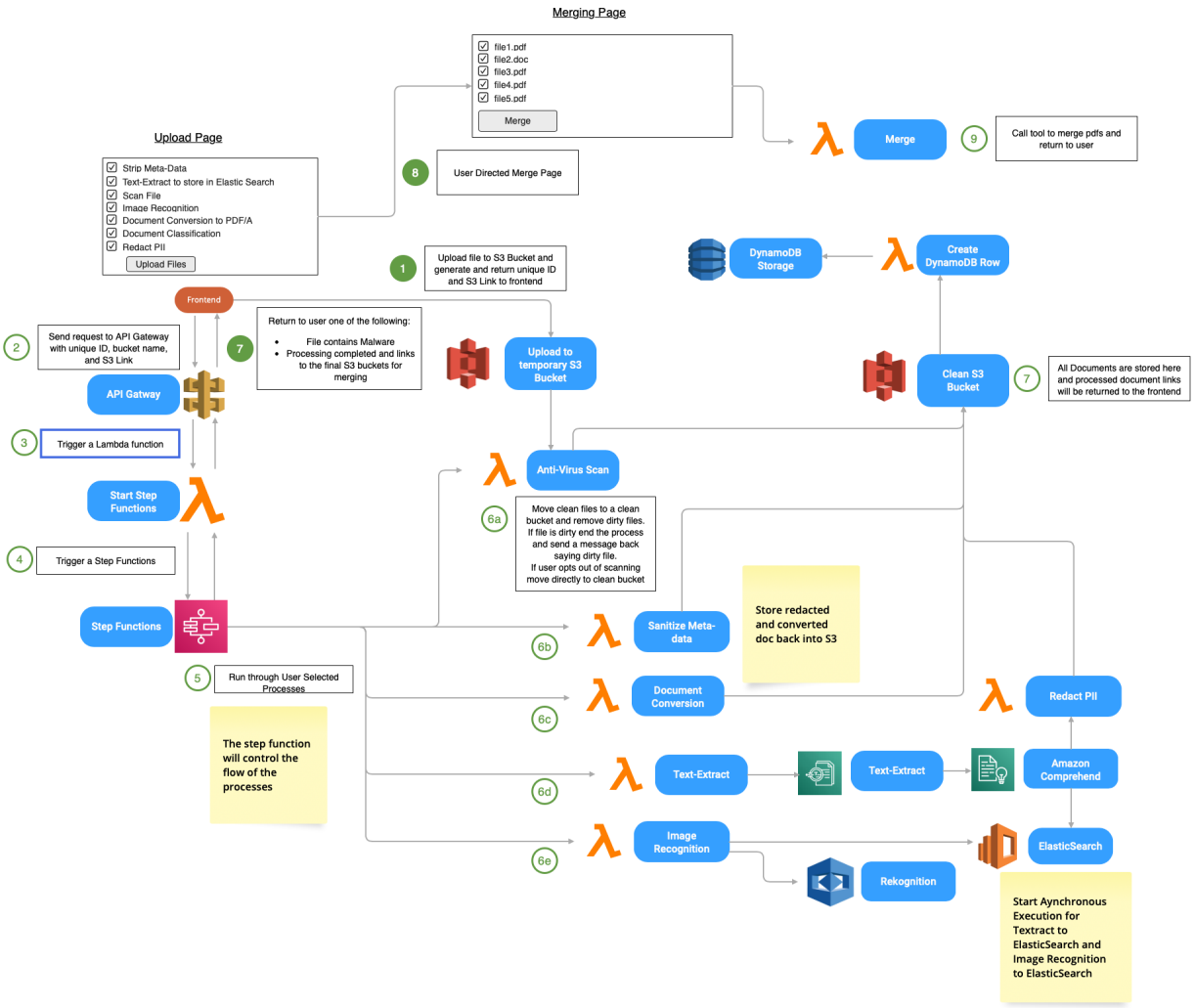
Textract Limitations: Only PDF, elasticsearch, sentiment analysis, redaction

Overview

Using this system consists of two main parts.

1) Upload Documents
2) Running Functions on Documents

As seen in Figure 1, the frontend first uploads the documents to an unsanitized S3 bucket (Step 1). After upload, the frontend makes an API call through API Gateway which triggers running functions on the documents (Step 2). This API and the process of running functions on the documents will be explained further on "API Gateway and Step Functions".



Architecture Overview. Figure 1

Uploading Files

User ID: an unique identifier for each user generated randomly using npm uuid library and store in Localstorage in user browser.

Video Overview:

File Upload Video

Front-End Overview:

The user interface is deployed on AWS Amplify that is built when new changes are pushed to a repository on AWS CodeCommit. The front-end can be access using following link: Front End

Home Page: Users can select up to five files per upload with each file being less than 10MB in size. Users can then select multiple options or functions to perform on their selected files. First, the application will try to upload files to Amazon AWS S3 (Object Storage) using the **UNSECURE** way that exposes user Access ID and Access Key in the front-end. This user has access to upload files to only one bucket and no other access that might jeopardize the security of the organization. Upon failure to upload the files, the application will alert the user with failed status code and status text. If the upload was successful, it will invoke an API endpoint (AWS API Gateway) that has no authentication to perform the user-selected options on the files.

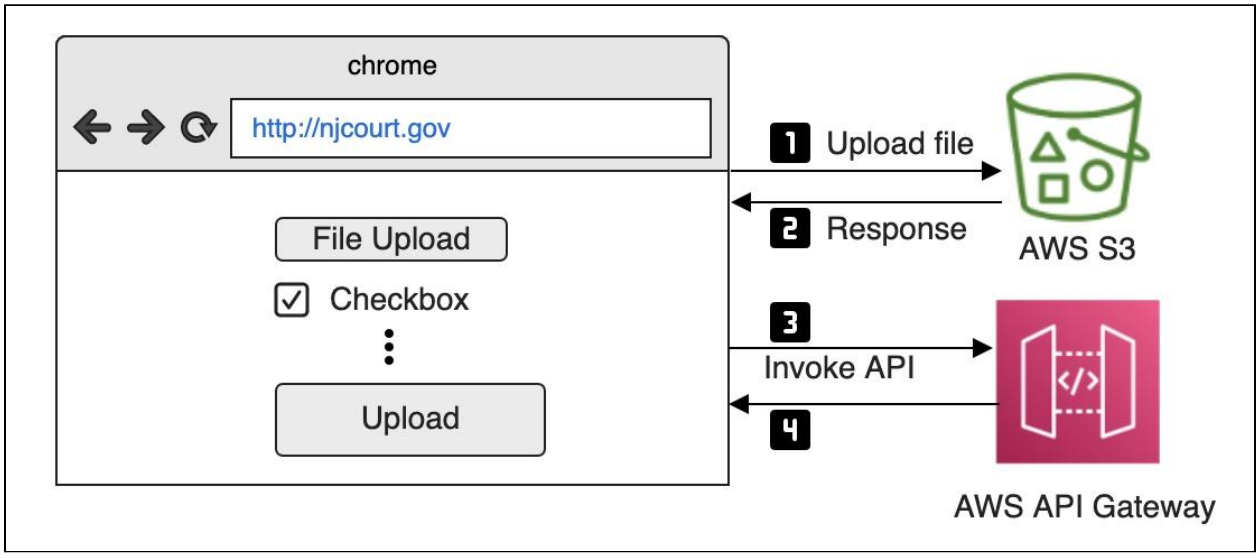


Figure 2. Broad overview of Application Architecture.

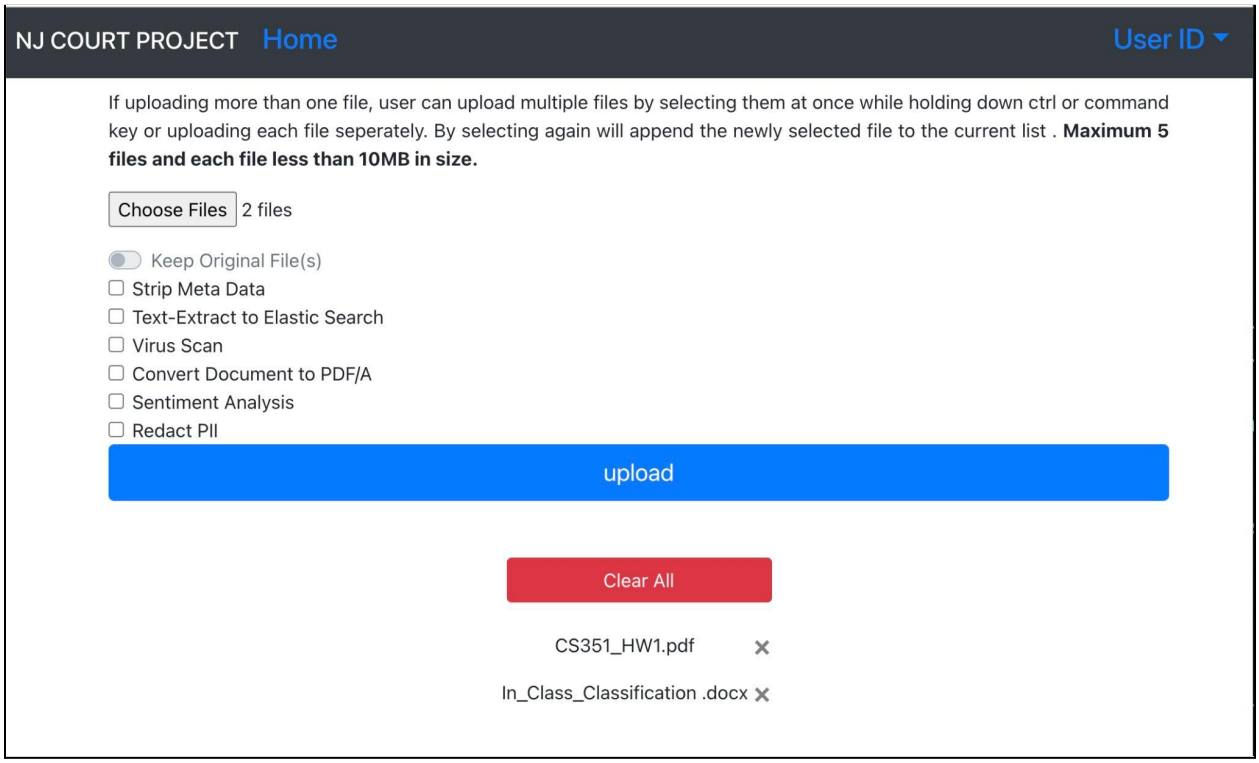


Figure 3. Main page with file upload option along with actions to perform on upload file(s).

After file upload and API invocation, the process takes place in the AWS cloud with different microservices and it looks something like the following architecture:

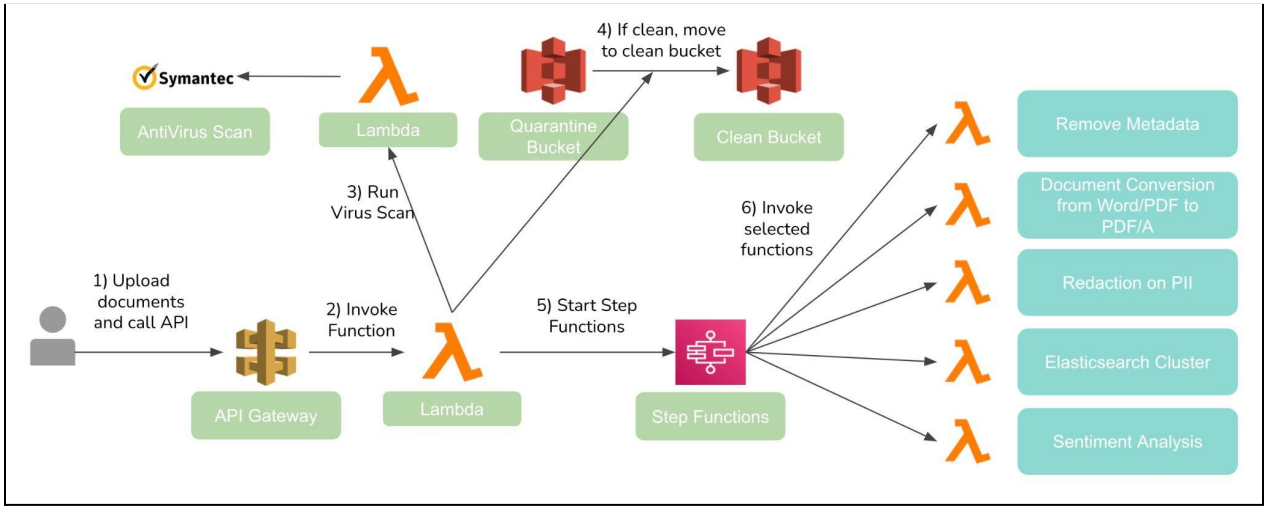


Figure 4. Application Architecture.

Merge Page: Once the user-selected functions have been executed successfully on their files, the user can select files to merge them or decide to continue to the **home page**. The user will be notified only about Virus Scan and Strip Metadata

results while all other error handlings will be done in the AWS Step Function. If merging was successful then the user will be presented with a **presigned URL** to download the merged files. The lifespan of a **presigned URL** is one hour so after that user won't be able to download the merge files using the same link presented on the merge page.

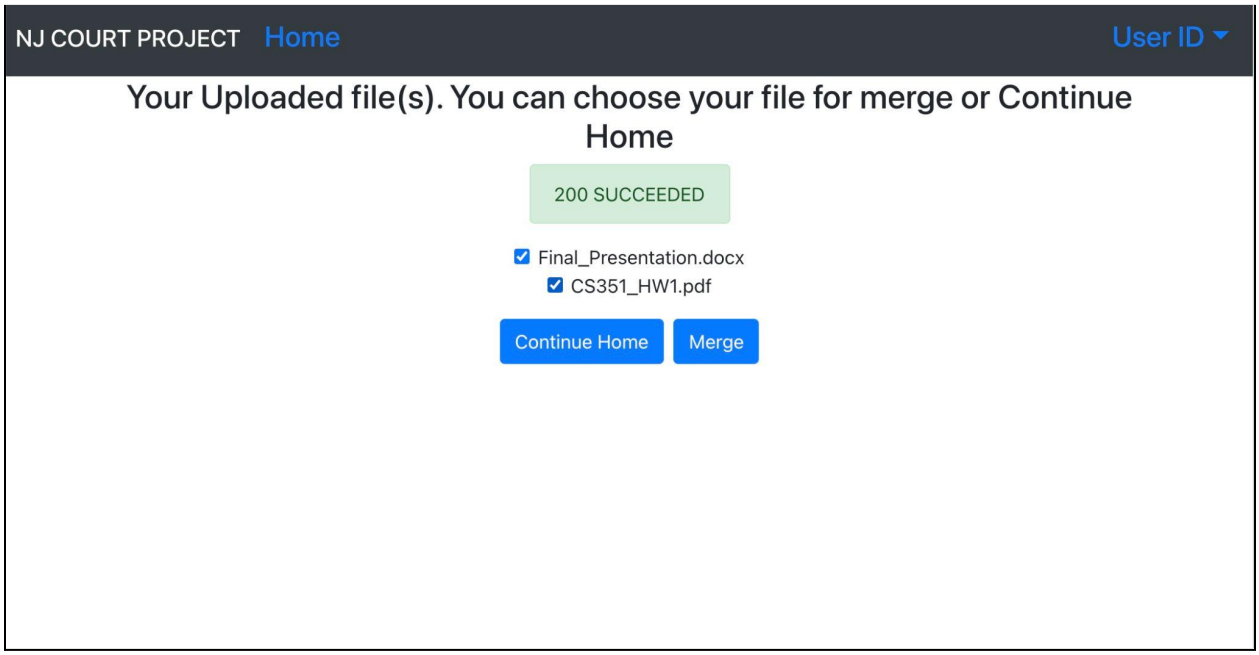


Figure 5. Merge Page with Merge Button.

API Endpoints

There are two api endpoints deployed to serve different purposes for the application including executing the user selected actions on their file(s) and merging requested files. The details about this endpoints are as follows:

Actions

Actions resource to perform user selected actions on their files.

Perform Actions

POST /actions

trigger chain of user selected actions on their file(s)

Body Parameters

---

files REQUIRED TYPE array

List of user file(s) object to perform actions upon

Example:

```
“files”: [
        {
                “displayName”: “Human Readable Name”,
                “key”:”S3 Object key”,
                “location”:”S3 file location”
```

}
]

---

selectedOptions REQUIRED TYPE object

Options that defines actions to be perform on file(s)

Example:

“selectedOptions”: {

“stripMetaData” : 0,

“elasticSearch”: 0,

“virusScan”: 0,

“convertDocument”: 0,

“sentimentAnalysis”: 0,

“redactPII”: 0

}

---

userID REQUIRED TYPE string

Unique user ID that is randomly generated on the front-end

---

submitTime REQUIRED TYPE string

Epsilon time of user submission of actions

Success Response

HTTP Status 200

Merge files

Merge resource to merge user selected files

Merge operation

POST /mergefiles

Merge requested files and return secure presigned url to download the merge file

Body Parameters

---

files REQUIRED TYPE array

List of user file(s) object to perform actions upon

Example:

“files”: [
”S3 Object key”, “another S3 Object Key”,....
]

Success Response

HTTP Status 200

---

statusCode TYPE string

Status of the action weather successful(200) or unsuccessful(NA)

---

preSignedURL TYPE string

Merged file S3 presigned url that has lifespan of 1 hour.

Step Functions

Synchronous Step Functions

Overview

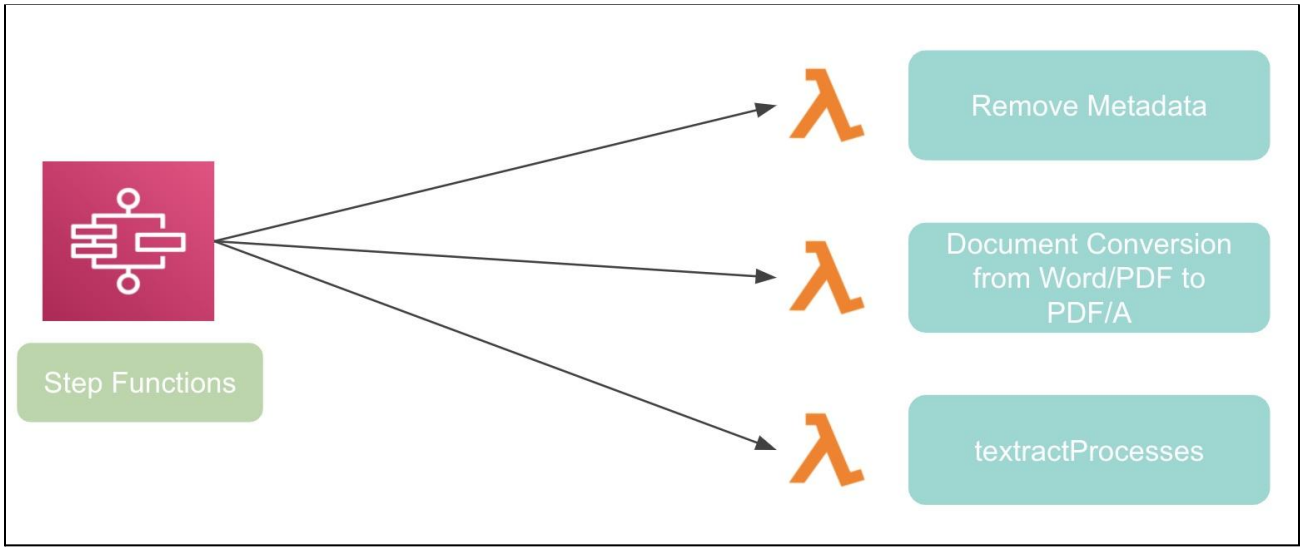


Figure. Overview of Asynchronous Step Functions

This section will discuss the asynchronous part of the application that carries out the main processes such as removing metadata, document conversion to PDF/A, and a process that triggers long-running machine learning algorithms on files such as redact PII (personally identifiable information) and sentiment analysis. AWS Step Functions is a serverless function orchestrator that makes it easy to sequence AWS Lambda functions and multiple AWS services into business-critical applications. The AWS Step Functions uses Amazon States Language that is a json-based, structured language used to define your state machine, a collection of states, that can do work (`Task` states), determine which states to transition to next (`Choice` states), stop an execution with an error (`Fail` states), and so on.

Pros:

- Build and update apps quickly
- Write Less Code

Implementation

The process is implemented using Express Workflows because it supports asyronochous workflows that were required for this implementation. The further information about Standard vs Express Workflows can be found using the Standard Vs. Express Workflows.The Step Function is implemented using AWS Management Console by defining our state machine using Amazon State Language. The definition of the Step Machine looks like following images:


```
▼ object {3}
    Comment : A Hello World example of the Amazon States Language using Pass states
    StartAt : ChoiceState
  ▼ States {6}
    ▼ ChoiceState {3}
        Type : Choice
      ▼ Choices [3]
        ▼ 0  {3}
            Variable : $.selectedOptions.stripMetaData
            NumericEquals : 1
            Next : stripMetaData
        ▼ 1  {3}
            Variable : $.selectedOptions.convertDocument
            NumericEquals : 1
            Next : convertDocument
        ▼ 2  {3}
            Variable : $.selectedOptions.textract
            NumericEquals : 1
            Next : textractProcesses
        Default : SuccessState
```


Figure. These are choice states that decides next transition based on the input variables

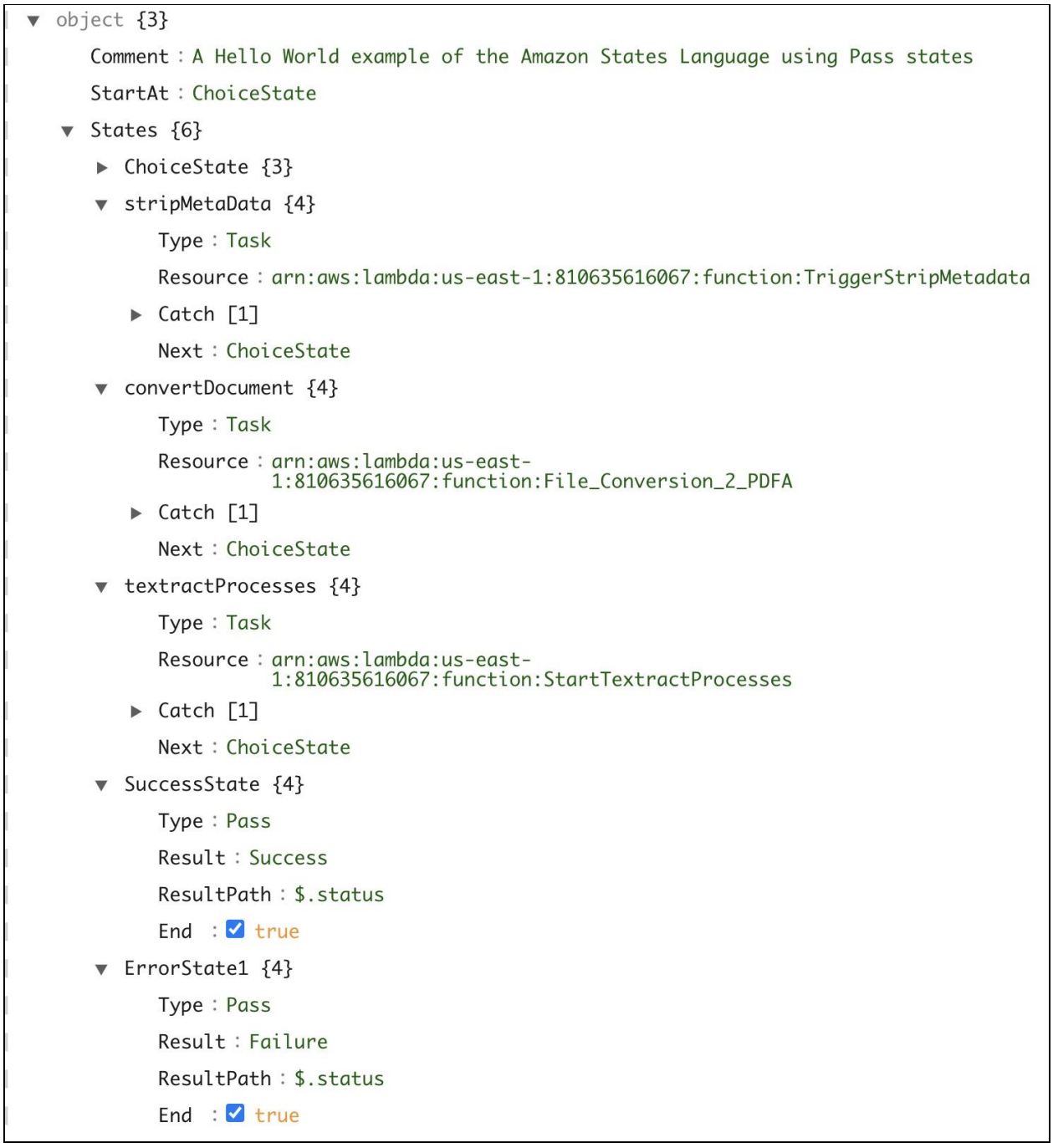


Figure. These are Task states that triggers Lambda Functions

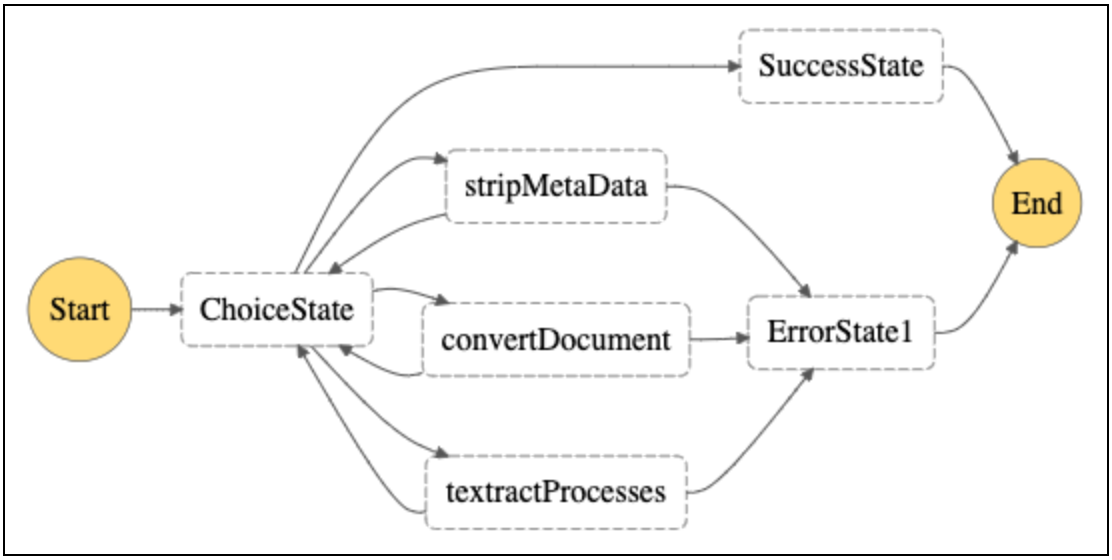


Figure. Visualization of Step Functions Work Flow

Asynchronous Textract Step Functions

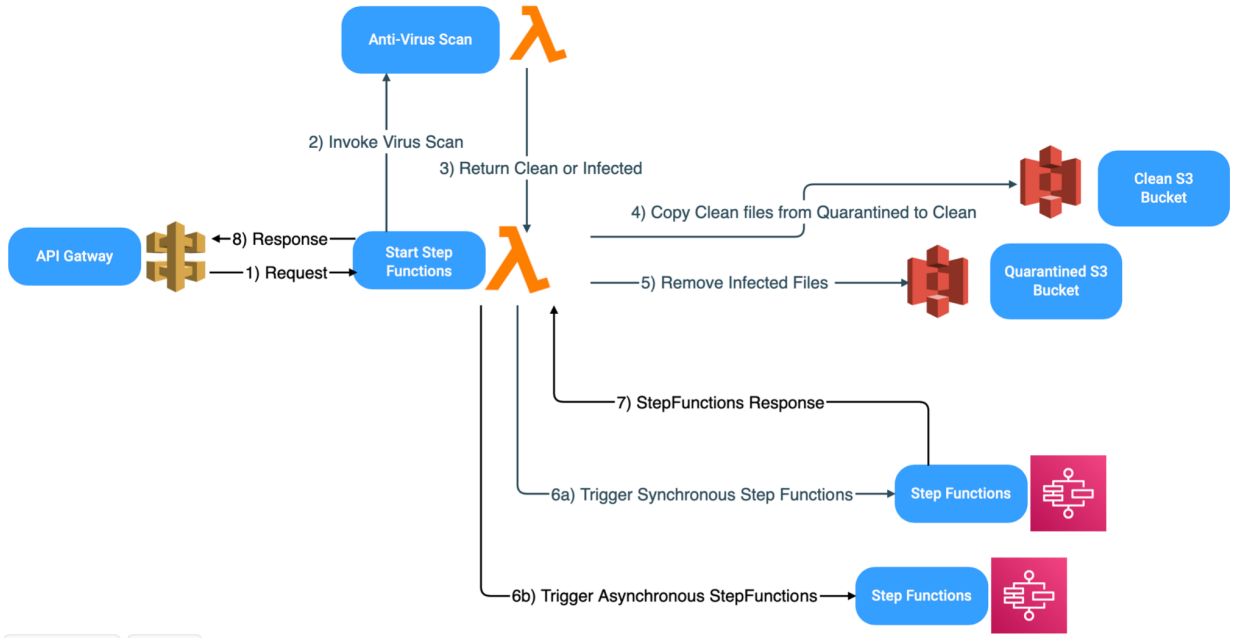
Lambda Functions

Invoke Virus Scan and Start StepFunctions

Overview

Runtime: Node.js 12.x

This section will discuss the Lambda function that invokes the antivirus scan and starts the step functions. For Anti-Virus Scan, check the VirusScan section. For Step Functions, check the Step Functions section.



The process for this Lambda is as follows.

1) API Gateway sends a request to Lambda containing file information, bucket key, etc.
2) This Lambda will invoke the Virus Scan Lambda on given files
3) The Virus Scan will respond with clean or infected for each file
4) For files, that are clean. They will be moved from the quarantine bucket to the clean bucket.
5) The files will be removed from the Quarantine bucket
6) Both the Synchronous and Asynchronous Step Functions will be triggered at this point
7) Only the Synchronous Step Functions will return a response
8) The response is passed back to API Gateway for the user.

Usage

```
Input: {
       Files: [
              {
              Key: keyName,
              OriginalName: origin
```

```
                },
                {
                Key: keyName,
                OriginalName: origin
                }
        ],
        QuarantineBucket: QuarantineBucketName,
        CleanBucket: CleanBucketName,
        Overwrite: 1,
        Use: {
                Name: CapstoneStepFunctions,
                'selectedOptions': {
                        'stripMetaData': 0,
                        'textExtract': 0,
                        'virusScan': 0,
                        'imageRecognition': 0,
                        'convertDocument': 0,
                        'documentClassification': 0,
                        'redactPII': 0,
                        'merge': {
                                'status' : 1,
                                'post-modification' : 1
                        }
}
}
}
```

Output:
TBD

Implementation

Development

When developing this Lambda, I utilized the Serverless Framework to help with deployment.
[This video helped me get started.]
[Download Serverless here.]

With serverless, deployment to Lambda is very simple. With the serverless.yml that is created upon setting up the serverless project, AWS configurations can be made easily. Setting up IAM Role Permissions and API Gateway Endpoints are made easy.

The command to deploy with serverless is `serverless deploy`
The deployment will be made to AWS Lambda, API Gateway, CloudFormation.

Code

This lambda was written in NodeJS for its faster start up times.
AWS-SDK is used for communication with S3, Lambda, and StepFunctions.

Step Functions requires that each execution made must have a name and the name must be unique. In order to circumvent this issue, I utilized a ‘uuid’ package to generate a random number that will be assigned to each execution.

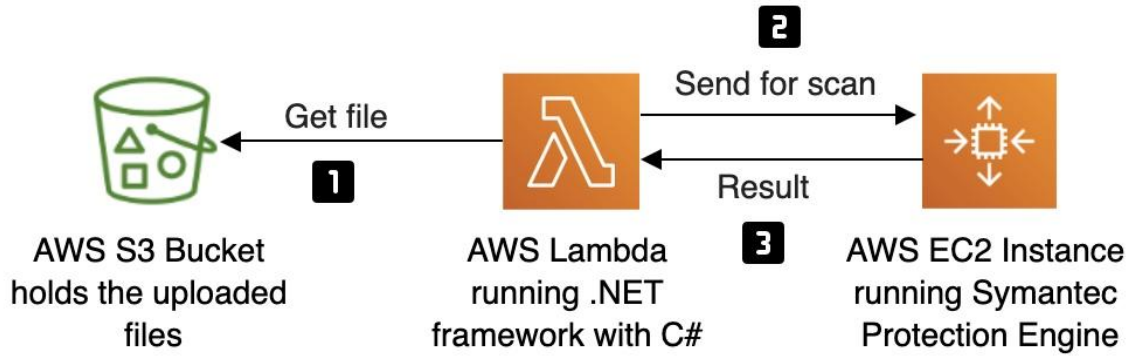
Two functions were made in handler.js.

1. cleanBuckets(files, scannedResults)
    a. Will compare the file with the scanned results. Based off whether the file is clean or infected. The function will have S3 copy the file from the quarantine bucket to the clean bucket (For clean files). The file will be deleted from quarantine bucket regardless.
2. startStepFunctions
    a. Will simply start StepFunctions Execution

VirusScan

Runtime: .NET Core 3.1 (C#)

Overview



1) Download File from S3
2) Send file bytes with ICAP Protocol to EC2 instance for scanning
3) EC2 instance returns ICAP Response with results from scanning

An AWS Lambda Function works as the ICAP Client that sends file data over to an ICAP Server (EC2 for Symantec Protection Engine running with hibernation enabled). The server sends back data on whether violations were found in the files.There is a great article that talks about different states of AWS EC2 Instance.

The lambda function assumes that the input files have already been uploaded to the S3 bucket. The function will download the file from S3 and send the bytes over to the ICAP server for scanning.

The SPE server was set up using the Symantec Protection Engine for Cloud Services for Linux. Using this will automatically create an EC2 with the Engine from an Amazon Machine Image. All that's needed is to launch the instance and wait for it to start.
This AWS Lambda Function performs virus scanning on files with Symantec Protection Engine. Acts as an ICAP Client, sending ICAP requests to the Symantec Protection Engine server. The server responds with an ICAP response with scanning information on the file.

Usage

Can be invoked with the AWS-SDK.
IMPORTANT: Change the IP Address of the server.
Input:

```
{
```

```
      bucketName: <s3-bucket-name>,
      fileKeys: [ <file-1>, <file-2>, ...]
}
```

Output:
Returns an array of boolean values corresponding to the order of input files. True denotes a clean file. False denotes an infected file.

Implementation

Development

To get started with C# development, I utilized .NET Core CLI as instructed here. Following these instructions will provide a basic setup for an empty AWS Lambda function. With this, you'll be able to deploy changes to the lambda function. There's more info on the commands here, including deployment, invoking, deleting, updating, and more. I suggest deploying and then developing. Most development took place on the deployed lambda.

Deploy Command:

```
dotnet lambda deploy-function MyFunction
```

Invoke Function:

```
dotnet lambda invoke-function MyFunction --payload "The Function Payload"
```

--payload is the input parameter

From here, I copied ICAP.cs and the ICAPException.cs and added the appropriate dependencies and packages.
Once the appropriate IP address and port are given. Invoking the lambda function should allow for communication to the ICAP Server that is made with the IP.

IMPORTANT: The IP Address given by the EC2 instance will be needed to be provided for use. Everytime the EC2 is restarted, the IP address is changed.For our project purposes, it is currently left hardcoded.

Resources:
Getting started with .NET and AWS Lambda
.NET Core CLI and AWS Extension Documentation

Code

The class files ICAP.cs and ICAPException.cs were mostly taken from this repository. Slight modifications were made to have the class work with Symantec Protection Engine (SPE)

specifications, most of the code remains unchanged. This repository will be able to help with some questions.
The Repository contains both a C# and Java implementation. I decided to work with C# because the latter is notorious for long cold starts when it comes to AWS Lambda. Communication to the server is facilitated by Socket.Net over TCP.

The ICAP method used is RESPMOD which is meant specifically for server anti-virus scanning. The service is called SYMScanResp-AV. More information for ICAP requests, response, methods, services, error codes, and more can be found on Symantec Protection Engine SDK.
Resources:
AWS Marketplace Symantec Protection Engine for Cloud Services on Linux
https://github.com/Baekalfen/ICAP-avscan
Symantec Protection Engine 8.2 Documentation (SDK and more)

Testing

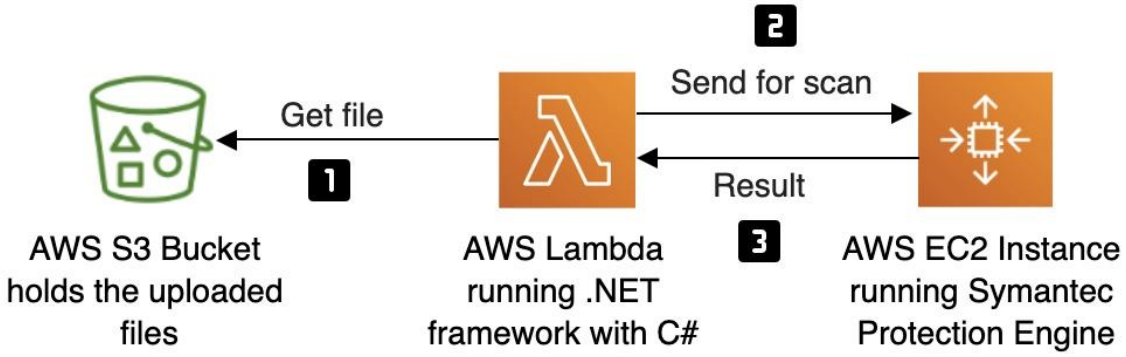
To test that the virus scan can detect viruses, I used the eicar string as input. It can be downloaded here. This is a txt file.

For the PDF File, I used this.
Resources:
Eicar Scanning test string
Eicar Scanning PDF

Alternatives:

Option 1:

This option utilizes the ICAP to communicate with Symantec Protection Engine running in a virtual machine in the cloud. We are using three different services that include AWS S3 (Object Storage), AWS Lambda (event driven serverless computing), and AWS EC2 Instance (virtual machine). Current implementation can be visualized as the diagram below.



Pros:

- Able to implement ICAP protocol in cloud

Cons:

- Not scalable
- Using C# to communicate with Symantec Protection Engine (SPE)
- EC2 Instance + SPE Licensing can be expensive

Option 2:

AWS Marketplace provides a couple of 3rd party S3 virus scan solutions that can be implemented by subscribing to their solutions. I am listing a couple of options that can further explore to find the best solution with good pricing. The list as follow:

1. Widdix VirusScan for AWS S3 Pricing: click here Website
2. Cloud Storage Security Antivirus for AWS S3 Pricing: click here Website

---

Widdix Pros:

- Virus definitions updated every 3 hours
- Dashboard for Full Visibility
- Scalable
- Data never leaves your AWS Account

- Delete or quarantine infected files
- Tag files with scan results

Widdix Cons:

- Powered by ClamAV

Cloud Storage Security Pros:

- Object upto 15 GB with wide ranges of file types
- Dashboard for Full Visibility
- Scalable using Containers
- Data never leaves your AWS Account
- Delete or quarantine infected files
- Tag files with scan results

Cloud Storage Security Cons:

- Powered by multiple virus detection engines including Sophos and ClamAV

Merge Documents

Overview

This lambda function combines the input word document file(DOCX) or PDF files into a single PDF file. Input is passed into the lambda as a JSON format. Functionality of this lambda function is divided into 4 parts.

1. Download all the input files from S3 bucket into the local repository.
2. Check the extension of the file. If it's a word file then convert it into a pdf file.
3. Now, at this point all the files are in PDF format. Call the Merge file function to combine all the files into a single PDF file.
4. Once all the files have been successfully merged then create a pre signed URL of that file and return it in the response of the lambda function.

Implementation

Coding Language : Python 3.7
Library/Tools used : VSCode, AWS Cloud9, PDFTron, AWS S3, AWS Lambda

PDFTron is an open source library which we can use to perform almost all the operations on a PDF file. It supports multiple languages like Node.js, Java, Python etc. I have used this library in merging docx/ pdf files into a single PDF.

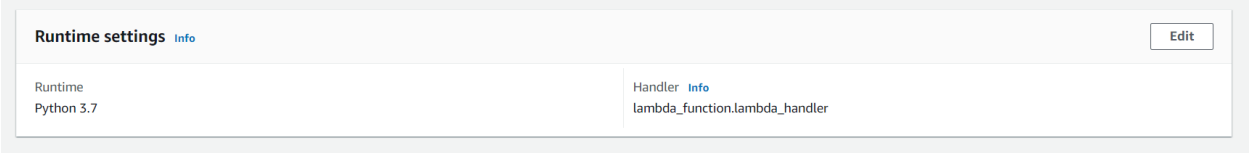
In python it supports upto version <3.8.

Create a Lambda function, add the required roles to it from the Permission setting inside Configuration. Role likeAmazonS3FullAccess.

After creating the lambda function setup the code in local.
Command to install the library:
> python -m pip install boto3 -t .
> python -m pip install PDFNetPython3 -t .

To upload the code on Lambda, we have to zip all the libraries and code together.
Command to zip the code and library into the current directory:
> zip -r lambda_function.zip .
(Period means all)

To run and test the lambda function, cross check the runtime setting of lambda function.
In Handler the first parameter should be the file_name of the code file. Like for example my code file name is "lambda_function.py", so lambda_function should come first. Then function name which invoked our lambda function and which takes the event and context as an input.



Input

```
{
  "files": [
    "a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/NJIT Intern Packet Spring 2021.pdf",

"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b93ffb116176465405640.pdf",

"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b93ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document"
  ]
}
```

Output

```
{
  "statusCode": 200,
  "Pre-Signed-URL": "URL of the Final Merge File"
}
```

Issues/Error Occurred While Developments

1. Searching for the best library to implement the use case. After exploring a couple of libraries I got the PDFTron which is suitable for all use cases. But while installing it I got error like:
   >python -m pip install PDFNetPython3
   Could not find a version that satisfies the requirement PDFNetPython3
   ERROR: No matching distribution found for PDFNetPython3

   Reason:: IT only supports the python version upto 3.8 and I was using 3.9. SO I have to downgrade the version to install it locally.

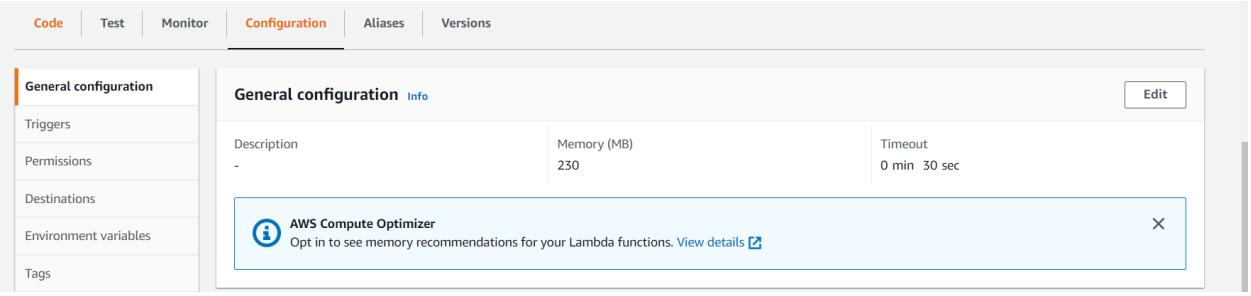
2. Error while using the lib in AWS lambda.
   ModuleNotFoundError: No module named '_PDFNetPython'
   File "/usr/lib64/python3.7/importlib/__init__.py", line 127,
   Reason:: I did not exactly find out the reason for this but I think its because of the OS difference. I am using windows and lambda support unix.
   When I used the Cloud 9, Import all the libraries and compressing and uploading the code from Cloud 9 the error got resolved and it get working.

3. In lambda when we download the files, we always have to download into the “tmp” folder always. I was not giving the “/tmp/” location when downloading the file into local and reading those from local. After correcting the path the code was working fine.
   s3.Bucket(BUCKET_NAME).download_file(KEY,'/tmp/' + KEY)

4. Also Check the permission which you have given for the lambda. I got error like
   [ERROR] OSError: [Errno 30] Read-only file system: 'Assignment1_Sol.pdf.CE332DEE'
   Traceback (most recent call last):
     File "/var/task/lambda_function.py", line 130, in lambda_handler
       s3.Bucket(BUCKET_NAME).download_file(KEY, KEY)

   Reason: I didn’t give the correct permission and role policy to the lambda.

5. TimeOut error while executing the lambda function.
   Cross check the General configuration setting and increase the timeout in the lambda function. By default it’s set to 3 sec.Aslo give the memory as needed for your code.



6. Access denied while uploading the file to S3. Error like S3UploadFailedError: Failed to upload . for that we have to add the permission and policy. https://aws.amazon.com/premiumsupport/knowledge-center/lambda-execution-role-s3-bucket/

Conversion from PDF/Word to PDF/A

Overview

This lambda function converts the word document (docx) or pdf document into a PDF/A document file. PDF/A is the standard document file which can't be edited further and can be stored for a longer period of time. Input is passed into the lambda as a JSON format.
Functionality of this lambda function is divided into 4 parts.

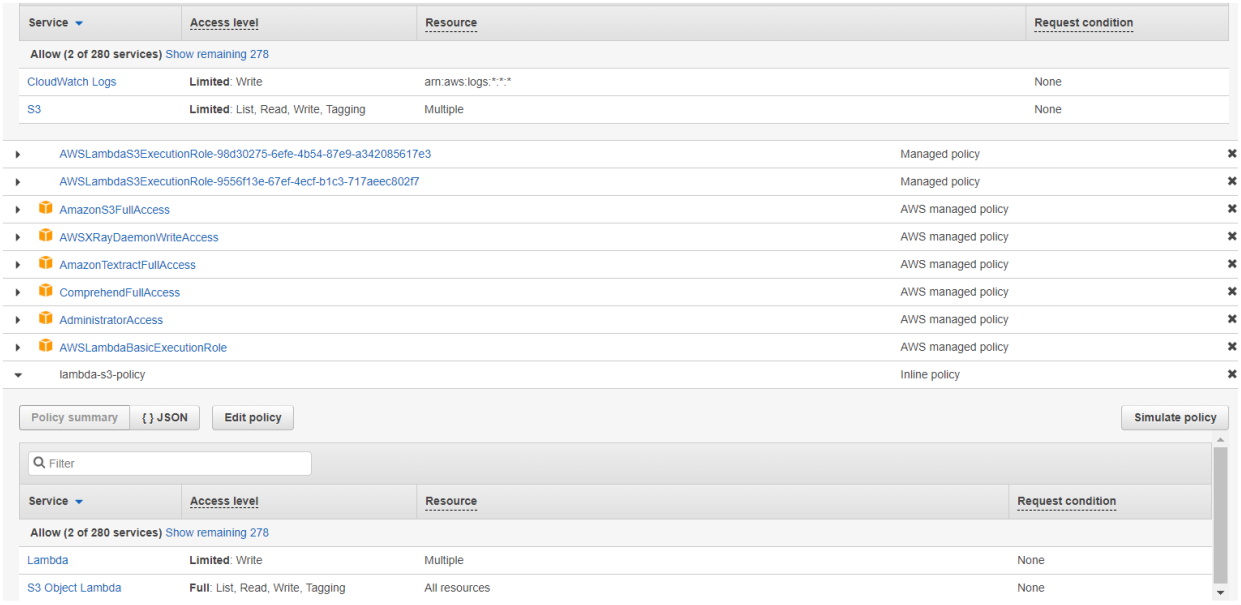
1. Download the input file from S3 bucket into the local repository.
2. Check the extension of the file. If it's a word file then convert it into a pdf file.
3. Now, at this point all the files are in PDF format. Call the ConvertToPdfa function to convert the file into a PDF/A file.
4. Once all the files have been successfully converted then upload the file into S3 bucket.

Implementation

Coding Language : Python 3.7
Library/Tools used : VSCode, AWS Cloud9, PDFTron, AWS S3, AWS Lambda

PDFTron is an open source library which we can use to perform almost all the operations on a PDF file. It supports multiple languages like Node.js, Java, Python etc. I have used this library in merging docx/ pdf files into a single PDF.
In python it supports upto version <3.8.

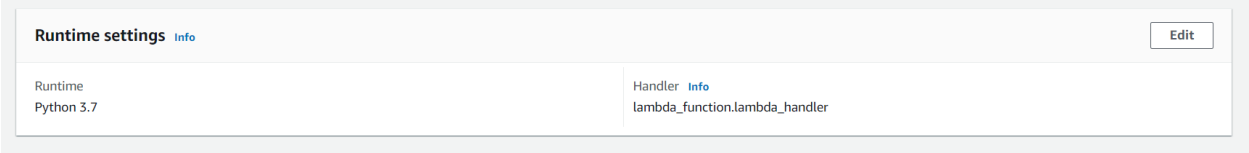
Create a Lambda function, add the required roles to it from the Permission setting inside Configuration. Role likeAmazonS3FullAccess.



After creating the lambda function setup the code in local.
Command to install the library:
> python -m pip install boto3 -t .
> python -m pip install PDFNetPython3 -t .

To upload the code on Lambda, we have to zip all the libraries and code together.
Command to zip the code and library into the current directory:
> zip -r lambda_function.zip .
(Period means all)

To run and test the lambda function, cross check the runtime setting of lambda function.
In Handler the first parameter should be the file_name of the code file. Like for example my code file name is “lambda_function.py”, so lambda_function should come first. Then function name which invoked our lambda function and which takes the event and context as an input.



Input

```
{
  "files": [
    {
      "displayName": "CS351_HW1.pdf",
```

```
      "key":
"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b9
3ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document",
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617
215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.pdf"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 0,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 1,
    "documentClassification": 0,
    "redactPII": 0
  },
  "userID": "userID",
  "submitTime": "epsilonTime"
}
```


Output


```
{
  "statusCode": 200,
  "files": [
    {
      "displayName": "CS351_HW1.pdf",
      "key":
"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b9
3ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document",
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617
215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.pdf"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 0,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 0,
    "documentClassification": 0,
```

```
    "redactPII": 0
  }
}
```

Issues/Error Occurred While Developments

1. After exploring a couple of libraries I got the PDFTron which is suitable for all use cases.The libraries which didn't work because of errors are docx2pdf ("errorMessage": "docx2pdf is not implemented for linux as it requires Microsoft Word to be installed"), LibreOffice (it needs an Libreoffice software to be present on the running machine)

Note: Once the file is converted into PDFA, we can't edit it. So the text extract function won't work on the PDFA file.

Redact PII

Overview

This lambda function takes PDF and PDF/A format files from JSON input. The code uses two comprehend methods of AWS named Textract and Detect PIIs. The Textract helps in extracting texts from images along with the confidentiality of each word and text. The text generated from textract is then given as input to the Detect PIIs method to find out PIIs present. The detected PIIs are then hidden, using their coordinates, on the input image. The Functionality of this lambda function is divided into 5 parts.

1. Download the input file from S3 bucket into the local repository.
2. Check the extension of the file for .pdf.
3. The input files are divided into individual pages and these individual pages are then converted to .jpg files.
4. Once all the files have been successfully converted to .jpg, the textract and redaction is done.
5. These redacted .jpg files are converted to .pdf files and then merged as one .pdf file. The finally merged file then replaces the original file on S3 bucket.

Packages Used

After trying a series of different packages, **PDFNetPython3**, is the most efficient and compatible package for achieving the desired target on Linux environment. Outside of the Linux environment, there are few more python packages that can be used. I have also included **psutil** and **logging** packages to make sure all the system exceptions are handled properly. The **pillow** (PIL) package has played a vital role in redacting PIIs using the coordinates.

Implementation

The code is most compatible with python 3.7 due to package dependencies on AWS lambda platform. The platforms used were VSCode, Cloud9 and AWS Lambda. All the platforms support upto python 3.9 but **PDFNetython3** and **pillow** are compatible with python 3.7 in the Linux environment. The VSCode was set up by providing AWS credentials but Cloud9 and Lambda functions do not require one. The Boto3 and Botocore packages were also required to be imported in VSCode and Cloud9 but these packages are in-built on lambda.

Input


```
{
  "files": [
    {
      "displayName": "In_Class_Classification.docx",
      "key": "a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/NDA-Capstone(3).pdf",
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617
215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638660.vnd.openxmlformats-offi
cedocument.wordprocessingml.document"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 0,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 0,
    "documentClassification": 0,
    "redactPII": 1
  },
  "userID": "userID",
  "submitTime": "epsilonTime"
}
```


Output

```
Page: 1
Detected 1 languages.
Detected 3 PII entities.
Page: 2
Detected 1 languages.
Detected 1 PII entities.
```

Read Permission: 1
Write Permission: 1

Issues/Errors Occurred while Development
Most of the errors were faced during transitions from VSCode to Cloud9 to AWS Lambda. The issues were majorly related to the compatibility of the platforms with the python version and its inter-dependencies of the packages. Many packages like **PyMuPDF, ImageFont** from **PIL** and **ImageDraw** from **PIL** were dropped even though these are more effective than the one currently being used.

The JSON input format that the AWS Lambda takes was also a bit difficult to understand. While the code runs, several files are created, stored in the local space of Lambda and once the final file is uploaded to S3 bucket, the files in the local are deleted. But lambda needs us to specify the local directory by adding “/tmp/” in front of the file names. Rest there were few runtime errors which were rectified by going through the logging information on Cloud Watch.

Sanitize Metadata

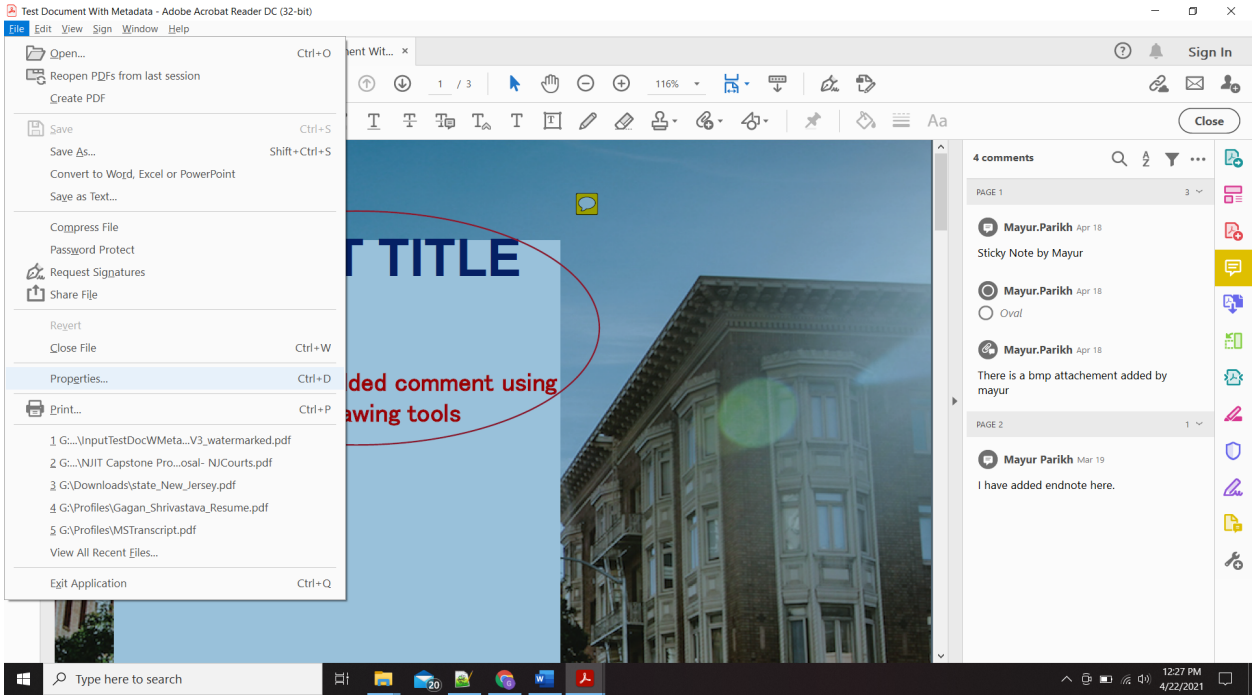
Overview
This lambda function removes the Metadata from the word document (docx) or PDF Document. The Metadata are the properties like (Author, Tittle, Revision, created by, etc..). To remove the metadata I have created two separate functions, one for word file and another one for the PDF files.

Below is the *list of different metadata properties* in the PDF file. From which except Advance properties sections, I am able to remove all.

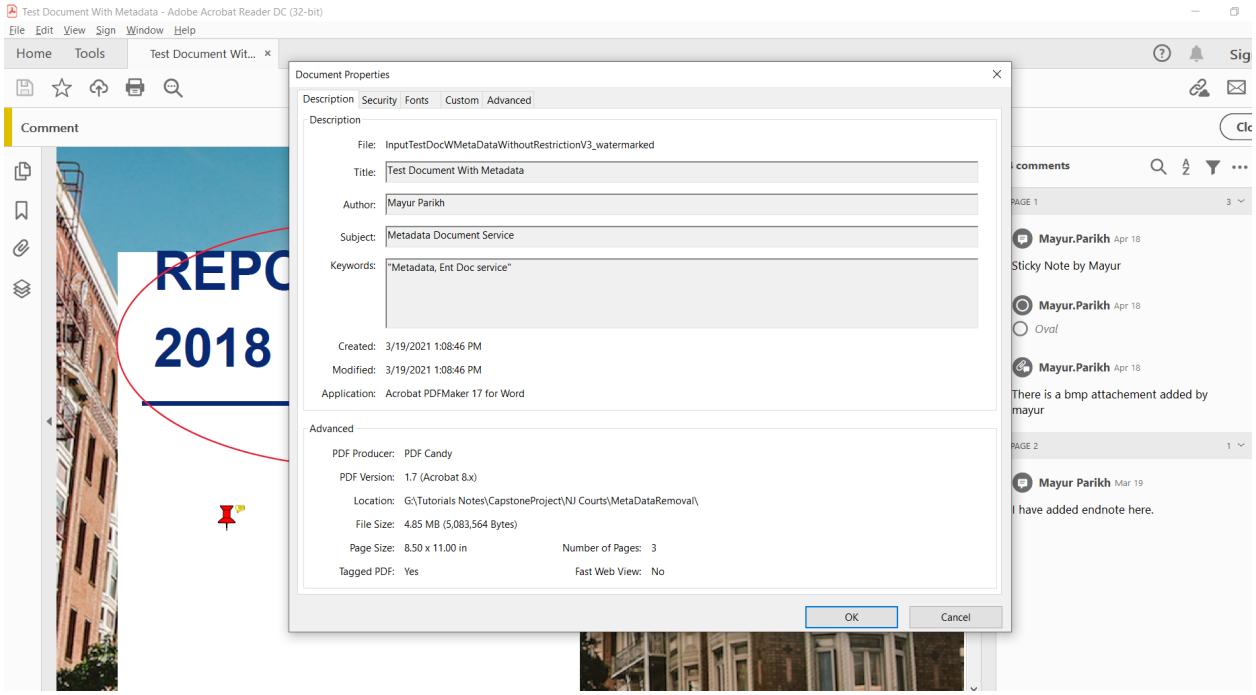
	PDF Metadata
	Metadata Type
File Properties	Title
	Author
	Subject
	Keywords
	Created
	Template ID
	Company
	Manager
	Source Modified
	Company
Comments	Annotations Via Drawing tool
	Sticky
	Comments
	Annotations
Advanced Properties	XML Data
	Base URL
	Copy Right
	Watermark

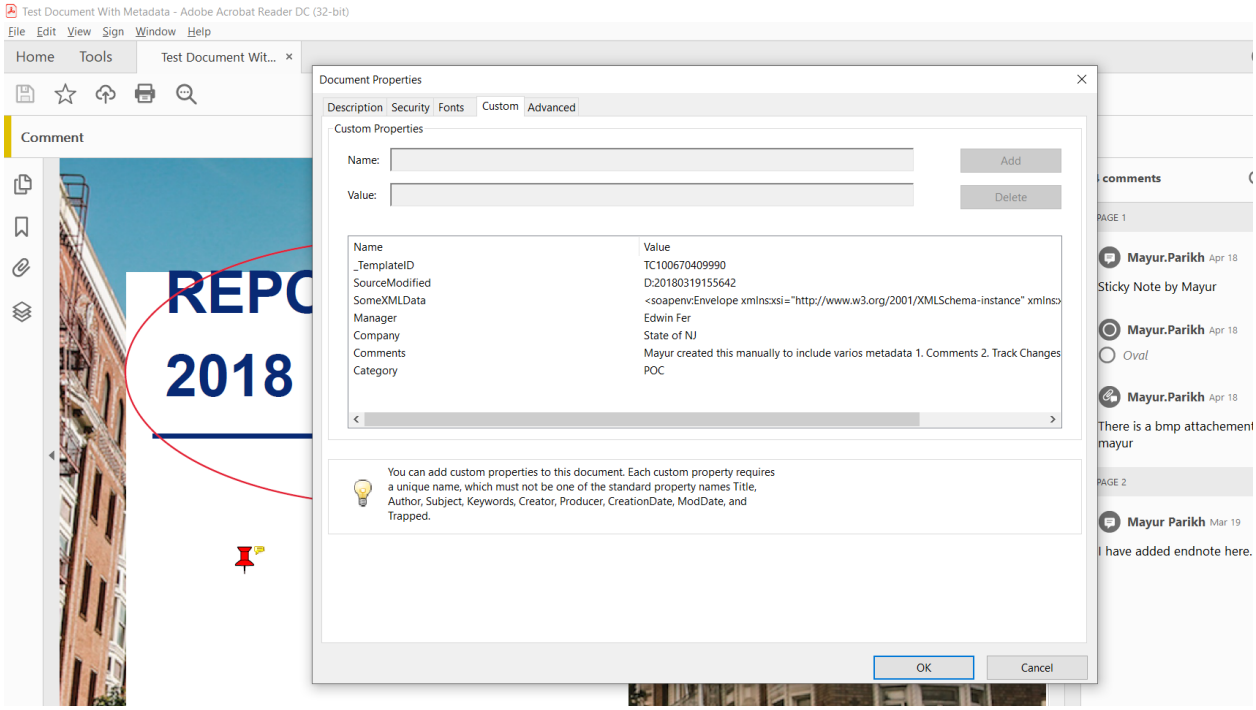
We Can see the metadata from PDF file using below steps:

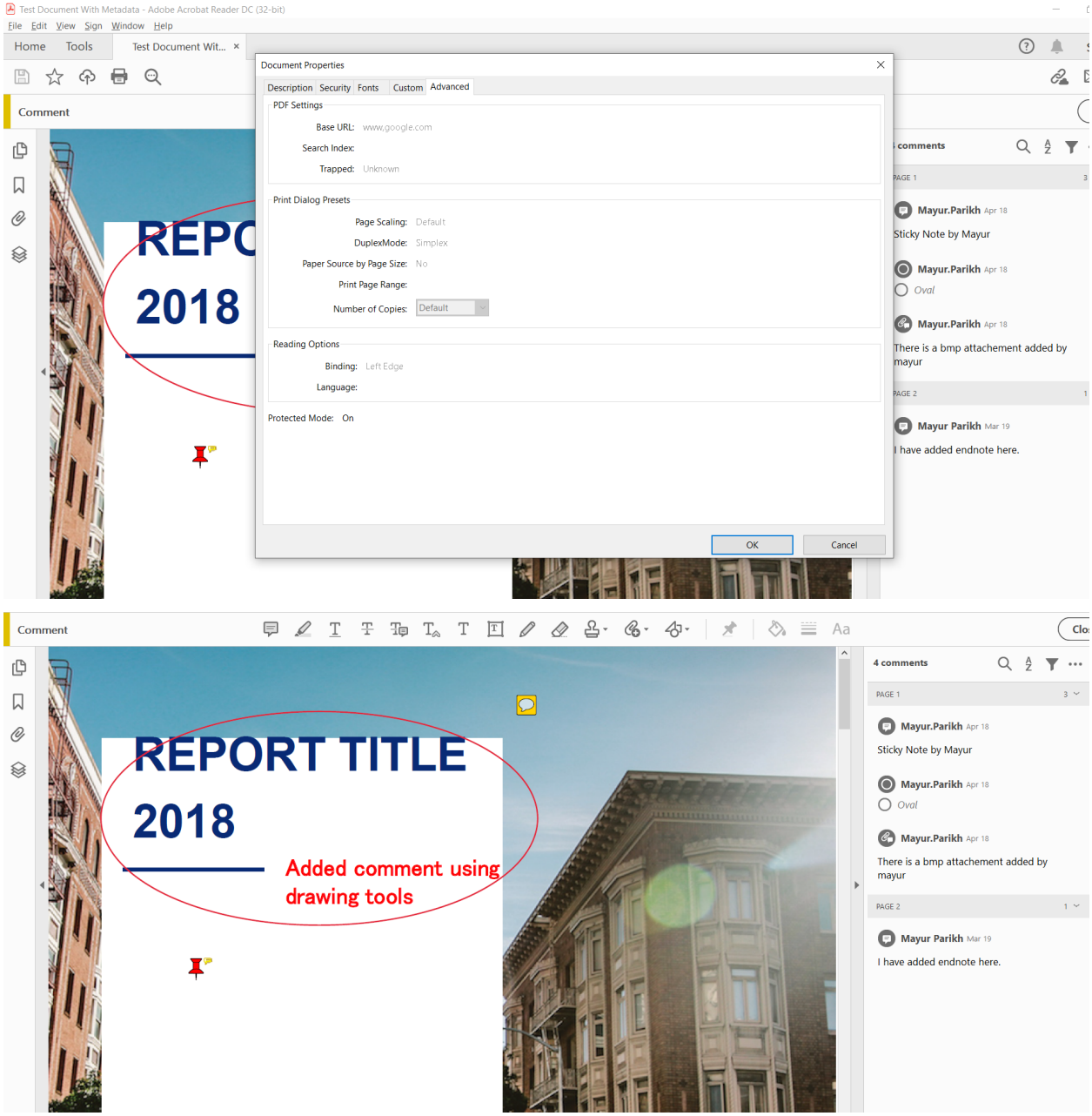
1. Open the file in Adobe Acrobat Reader.
2. Right click on the document. Choose the Document Properties (CTRL + D) OR at the top click on File, then choose properties.



3. Check the various properties.







Below is the *List of different properties in Word File*. I am able to remove the items which are in the green background.

Word Metadata	
Risk Level	**Metadata Type**
High Risk	Track Changes
	Comments
	Text smaller than 5pt
	Hidden text
	White text on any background
	previous authors
	Versions (Word 2003)
	Versioning(word 2003)
Medium Risk	Custom properties
	Document variable
	Macros
	Document reviewers
	Ink annotation
	Routing Slips (Word 2003)
Low Risk	Footnotes
	Field codes
	Document statistics
	Built-in properties
	Attached templates
	Smart tags words (2003/2007)

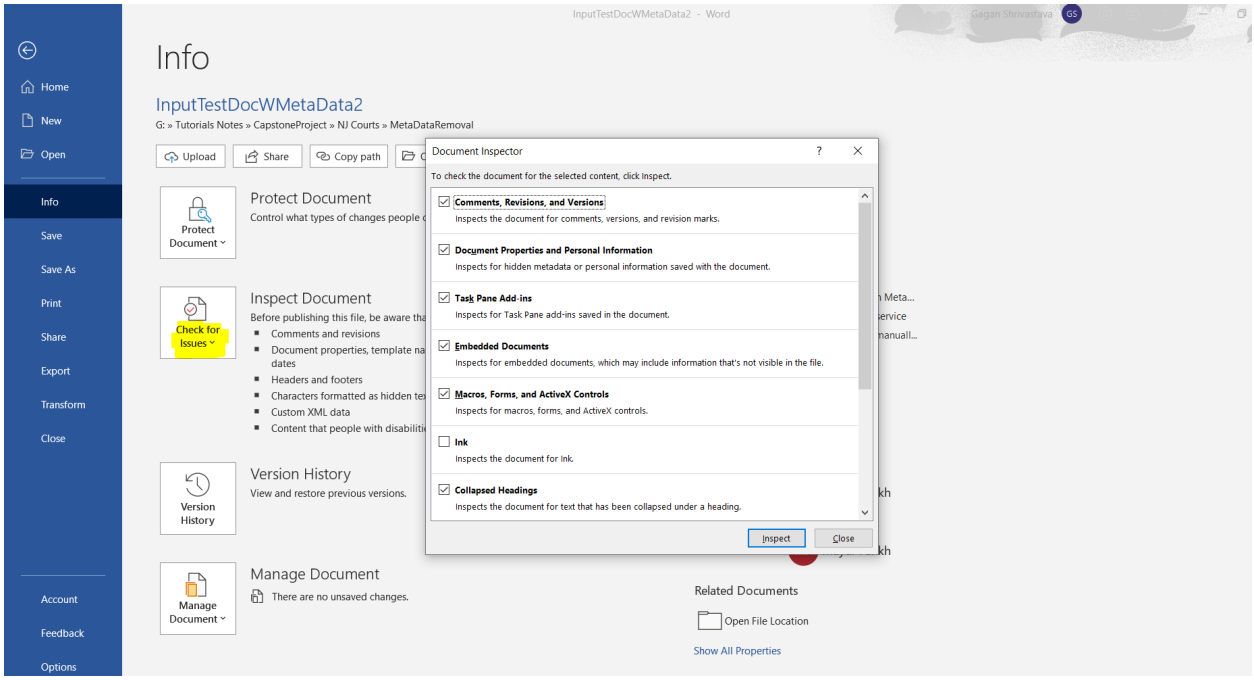
Fields Removed Successfully = Green

Fields not able to identify = Pink

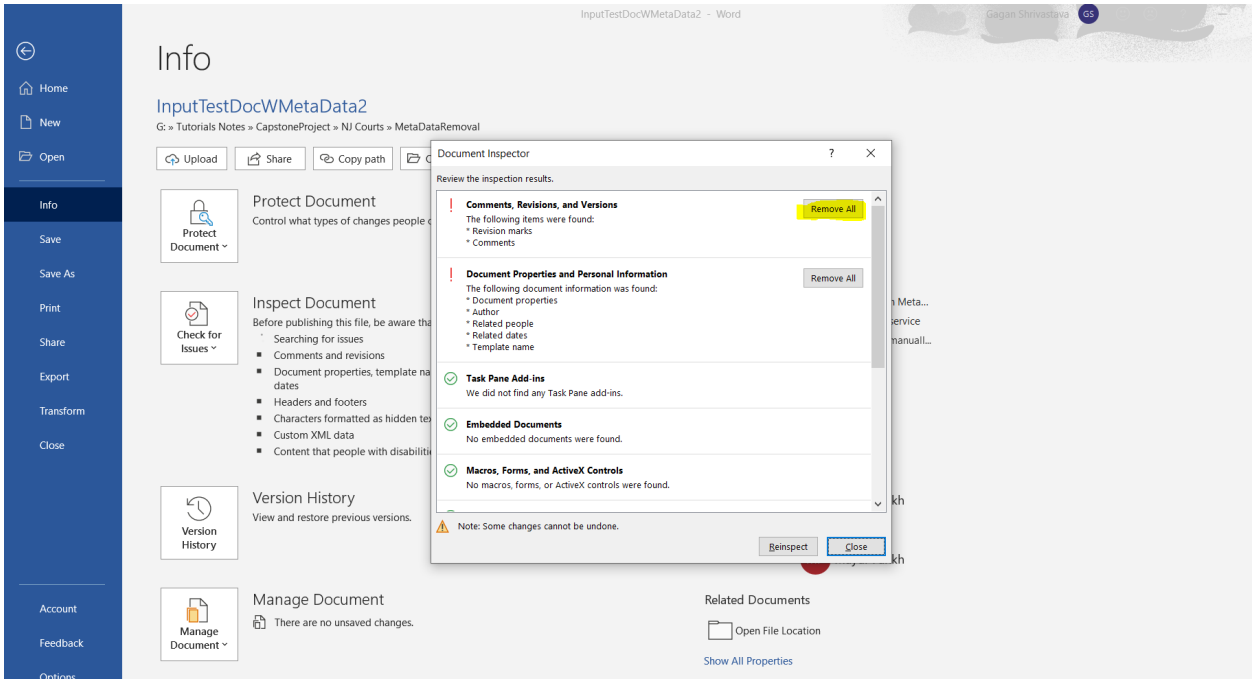
Rest of the other attributes = WIP

Steps to See and remove the metadata in word docx manually:

1. Open file in Microsoft word.
2. Go to File > Info > Inspect Document(check for issues) > click on Inspect



3. TO remove the metaData manually click on removeAll. Then Save the file.



Sanitize Metadata from PDFs

I used PDFTron to remove the metadata from the PDFs file. Lambda function take the Input in JSON format, validate the input JSON then download the file from S3 bucket into local

repository (‘/tmp/’) folder, remove the metaData properties from the file and upload it on S3 bucket(‘sanitize bucket’).

Implementation
Coding Language : Python 3.7
Library/Tools used : VSCode, AWS Cloud9, PDFTron, AWS S3, AWS Lambda

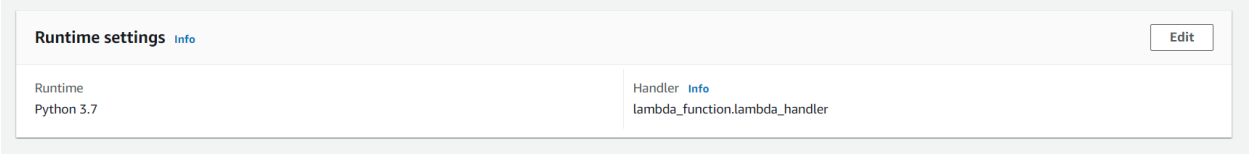
PDFTron is an open source library which we can use to perform almost all the operations on a PDF file. It supports multiple languages like Node.js, Java, Python etc. I have used this library in merging docx/ pdf files into a single PDF.
In python it supports upto version <3.8.

Create a Lambda function, add the required roles to it from the Permission setting inside Configuration. Role likeAmazonS3FullAccess.

After creating the lambda function setup the code in local.
Command to install the library:
> python -m pip install boto3 -t .
> python -m pip install PDFNetPython3 -t .

To upload the code on Lambda, we have to zip all the libraries and code together.
Command to zip the code and library into the current directory:
> zip -r lambda_function.zip .
(Period means all)

To run and test the lambda function, cross check the runtime setting of lambda function.
In Handler the first parameter should be the file_name of the code file. Like for example my code file name is “lambda_function.py”, so lambda_function should come first. Then function name which invoked our lambda function and which takes the event and context as an input.



Input

```
{
  "files": [
    {
      "displayName": "CS351_HW1.pdf",
      "key":
"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b9
3ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document",
```

```
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617
215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.pdf"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 1,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 1,
    "documentClassification": 0,
    "redactPII": 0
  },
  "userID": "userID",
  "submitTime": "epsilonTime"
}
```


Output


```
{
  "statusCode": 200,
  "files": [
    {
      "displayName": "CS351_HW1.pdf",
      "key":
"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b9
3ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document",
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617
215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.pdf"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 0,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 0,
    "documentClassification": 0,
    "redactPII": 0
  }
}
```

Issues/Error Occurred While Developments

1. After exploring a couple of libraries I got the PDFTron which is suitable for all use cases.The libraries which didn't work because of errors are

    a. Pikepdf("Not able to find the metadata properties in a tree structure path."), Basic Properties can be easily removed but not get enough documentation for advanced properties.

    Reference :
    pip install pikepdf
    pdf = pikepdf.open('NJIT Intern Packet Spring 2021_Gagan.pdf')
    print(pdf)
    meta = pdf.open_metadata()
    print(meta)

    https://pikepdf.readthedocs.io/en/latest/topics/metadata.html
    https://pikepdf.readthedocs.io/en/latest/topics/metadata.html#accessmetadata

    b. Pdfrw::
    This seems to work but have to set all th properties individually to null.
    And have to find the exact path for all the properties. Which seems to be difficult task.

    c. PYPDF2 : Use to extract the metadata only, can't remove from the file.
    import PyPDF2

    f = open('NJIT Intern Packet Spring 2021_Gagan.pdf','rb')
    pdf = PyPDF2.PdfFileReader(f)
    print(pdf.getDocumentInfo())

    d. 'pdf-parse' :: node js
    In this we can extract the meta data but not able to remove it from the file.'

Sanitize Metadata from Word Files

I used the Aspose tool to remove the metadata from the word document file. Lambda function take the Input in JSON format, validate the input JSON then download the file from S3 bucket into local repository (‘/tmp/’) folder, remove the metaData properties from the file and upload it on S3 bucket(‘sanitize bucket’).

Implementation

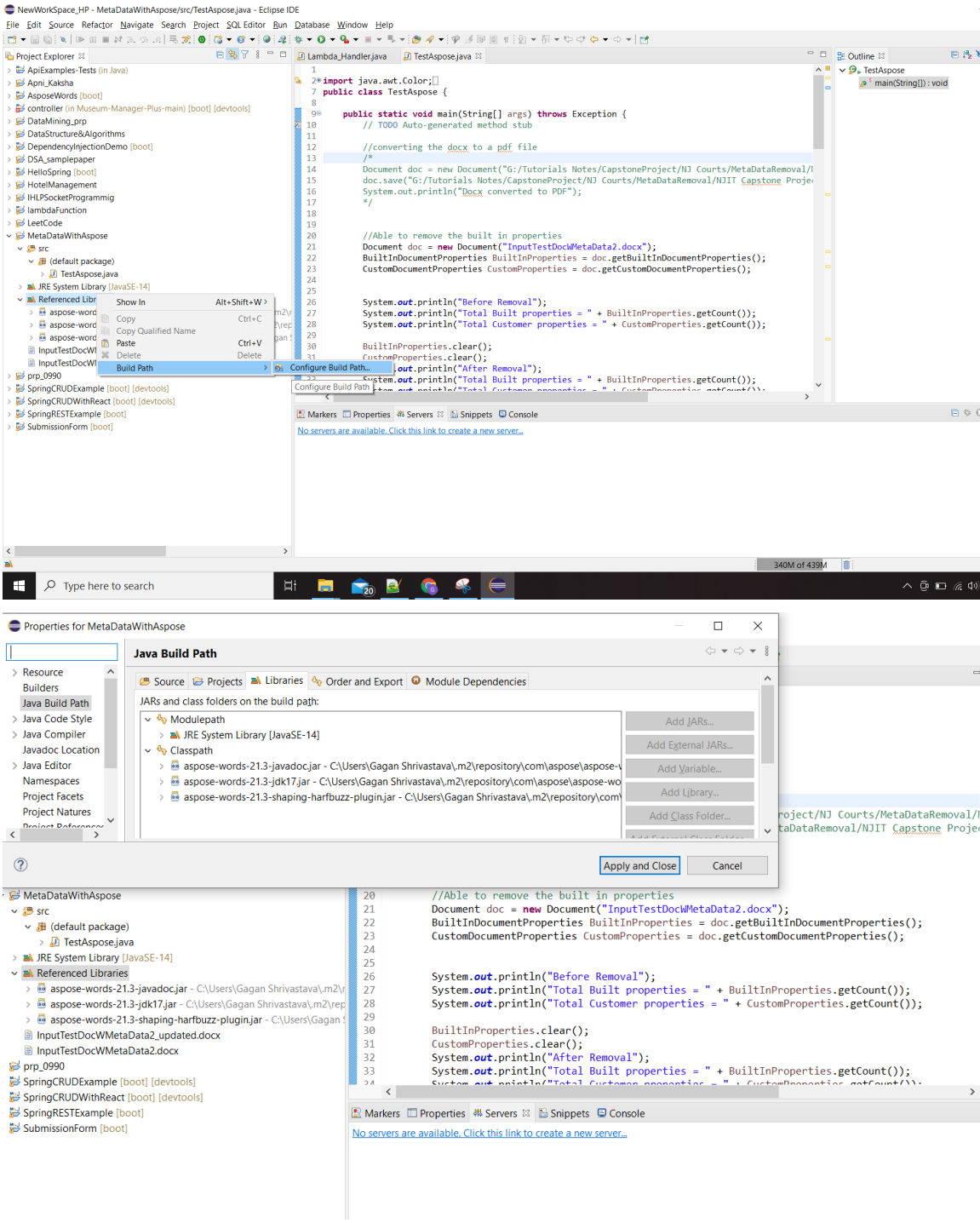
Coding Language : Java
Library/Tools used : Eclipse,Aspose, AWS S3, AWS Lambda

Aspose is an open source tool which is used to perform multiple tasks on the words and PDF files. We can use DotNet, Java, C# for the implementation. I have used Java to implement metaData removal functionality.

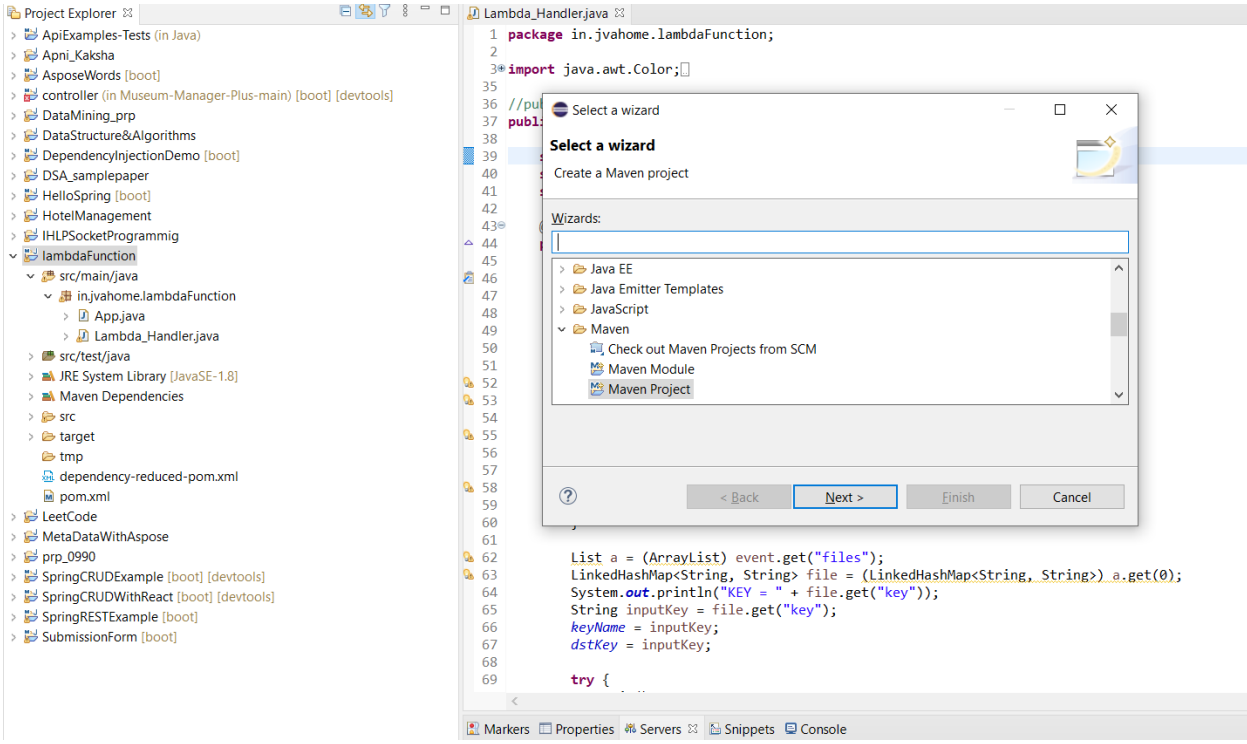
Create a Lambda function, add the required roles to it from the Permission setting inside Configuration. Role likeAmazonS3FullAccess.

After creating the lambda function setup the code in local. Steps to use the Aspose Library in java are:

1. One way to use this Tool is import the library in our project. Steps are :
    a. Open Eclipse > Project Explorer > (right click) New > Java Project (project)
    b. Download the jar of Aspose from https://downloads.aspose.com/words/java/new-releases/aspose.words-for-java-21.4/
    c. Open the project which you created > Reference Library > (right click) Build Path>Configure Build Path> Add external Jars >”get the jar from the downloaded location” > Apply

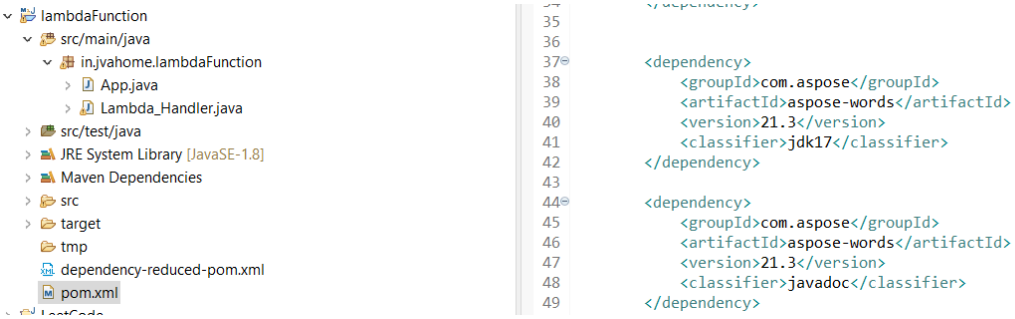


2. TO Use the Apsoe tool using Maven Project and in Lambda function:
    a. Open Eclipse > Project Explorer > (right click) New > Maven Project



b. Get the latest version of Aspose from: https://downloads.aspose.com/words/java

c. In pom.xml add the dependencies.



d. Add the AWS lambda dependencies.

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-s3</artifactId>
    <version>1.11.327</version>
</dependency>


<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-lambda-java-core -->
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-core</artifactId>
    <version>1.2.1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-lambda-java-events -->
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-events</artifactId>
    <version>3.8.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>


<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.4</version>
```

Reference to set up the code and use Java in Lambda Function.
For Aspose
https://repository.aspose.com/webapp/#/artifacts/browse/tree/General/repo/com/aspose/aspose-words/21.3

https://apireference.aspose.com/words/java/com.aspose.words/DocumentPropertyCollection
https://apireference.aspose.com/words/java/com.aspose.words/BuiltInDocumentProperties

To add the aspose library use :: have to add the repository and dependencies in pom.xml
https://docs.aspose.com/total/java/configuration-and-using-aspose-total-java-for-maven/

https://stackoverflow.com/questions/36463310/how-to-read-a-file-in-aws-lambda-function-written-in-java

To download the file and read from the S3 bucket

https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example-deployment-pkg.html#with-s3-example-deployment-pkg-java

Making the JAVA Lambda function
https://www.youtube.com/watch?v=JeJ46YIpPqw

To upload the code on Lambda, we have to zip all the libraries and code together. Sep is :

a. Right click on the project > Run As > Maven build
b. Check the console logs and see the build is successful.
c. Copy the jar location , take the jar which has a bigger size.
d. In AWS lambda, upload the jar.

Input


```
{
  "files": [
    {
      "displayName": "CS351_HW1.pdf",
      "key":
"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b93ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document",
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.pdf"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 1,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 1,
    "documentClassification": 0,
    "redactPII": 0
  },
  "userID": "userID",
  "submitTime": "epsilonTime"
}
```

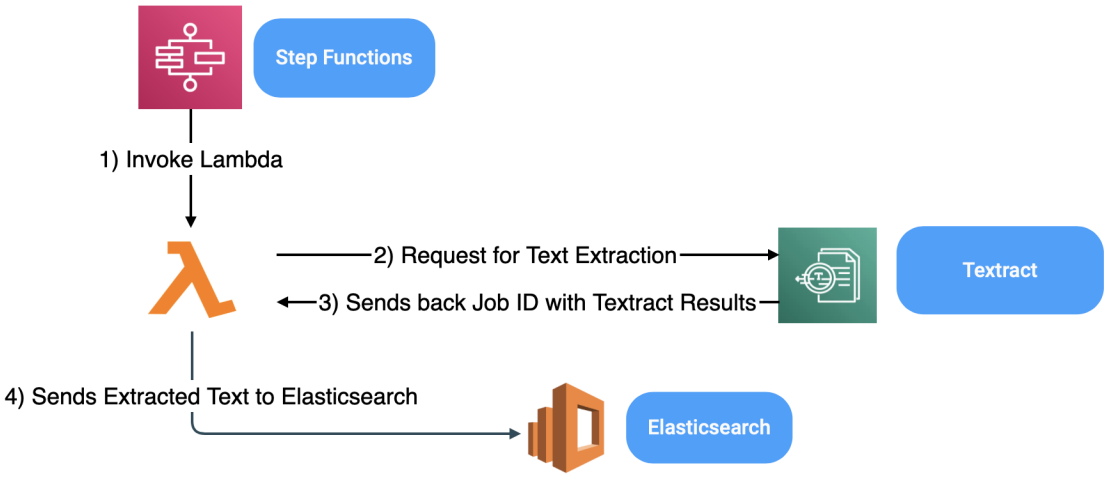

Output

```
{
  "statusCode": 200,
  "files": [
    {
      "displayName": "CS351_HW1.pdf",
      "key":
"a0aa0f2c-da69-4420-b5ea-bcd06b93ffb1/1617646540564/a0aa0f2c-da69-4420-b5ea-bcd06b9
3ffb116176465405641.vnd.openxmlformats-officedocument.wordprocessingml.document",
      "location":
"https://unsanitized-bucket.s3.amazonaws.com//bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617
215163866/bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.pdf"
    }
  ],
  "selectedOptions": {
    "stripMetaData": 0,
    "textExtract": 0,
    "virusScan": 0,
    "imageRecognition": 0,
    "convertDocument": 0,
    "documentClassification": 0,
    "redactPII": 0
  }
}
```

Elasticsearch Storage

Overview

Runtime: Python 3.6



1) Step Functions will invoke the lambda function
2) The lambda will make requests to Textract for file text extraction
3) A job ID with reference to extracted text will be sent back to user.
4) Extracted text will be stored to Elasticsearch along with data on filename, bucket location, link to object.

Usage

Input:

```
{
      "Bucket": <s3-bucket-name>,
      "files": [
            {
                  "key": <key-name1>,
                  "displayName": <display-name1>
            },
            {
                  "key": <key-name2>,
                  "displayName": <display-name2>
            }
      ]
}
```

Implementation

https://docs.aws.amazon.com/lambda/latest/dg/python-package-create.html

This webpage contains code to help implement Elasticsearch indexing.

Textract doesn't work on word documents currently.
Currently, Textract has a limitation where it cannot directly perform text extraction on a pdf synchronously.
Following this repo, we found a way to have the function return to us when the textract job has been finished and the extracted text.

1) Start the job, which will return a jobId
2) Check every 5 seconds to see if Job is finished
3) When job is finished, return the extracted text

For AWS Elasticsearch, the setup was made with this video. The video utilizes public access for domain access. Depending on user needs, the access may be changed to VPC. When ES gets created, a Kibana Visualizer will also be created.

For Kibana Visualization, Amazon Cognito Authentication can be used to manage access to kibana.

Resources:
Getting Started and Deploying Lambda with Python
Textract with Elasticsearch Indexing (Search and Indexing)
Performing Textract on PDF "Synchronously"
Performing Textract on PDF "Synchronously" Stack Overflow
Amazon Textract Limits

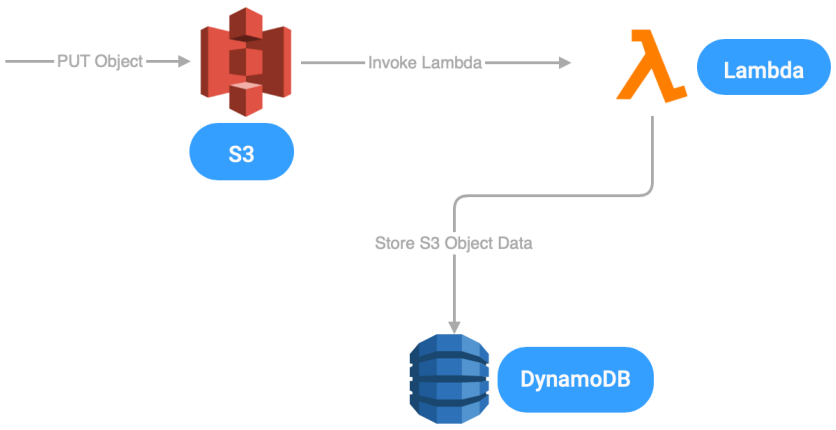
Sentiment Analysis

S3 Bucket to DynamoDB Table

Overview

After each function is run on the S3 object in the sanitized bucket, a corresponding DynamoDB row will be created for each S3 object with the help of a lambda function. This is done in order to make it easier to query for an S3 object using the UniqueUserId+EpsilonTime. Searching done on S3 is iterative, making it much slower than querying with DynamoDB. The primary key consists of a partition key, which is the folder hierarchy ({UserId}/{EpsilonTime}). The sort key is the file's name

({UserId}{EpsilonTime}{number}{file format}). The table consists of other attributes including the S3 BucketName, S3 Object Key Link, and the epsilonTime.



UniqueUserId	FileName	BucketName	Key	epsilonTime
bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617215163866	bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638662.vnd.openxmlf...	s3-transfer2-d...	https://s3-transfer2-dynamodb.s3.amazonaws.com/bb41fc75-4e2...	1617215163866
bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617215163866	bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638661.vnd.openxmlf...	s3-transfer2-d...	https://s3-transfer2-dynamodb.s3.amazonaws.com/bb41fc75-4e2...	1617215163866
bb41fc75-4e28-4c9b-862e-fe79393aa8a2/1617215163866	bb41fc75-4e28-4c9b-862e-fe79393aa8a216172151638660.vnd.openxmlf...	s3-transfer2-d...	https://s3-transfer2-dynamodb.s3.amazonaws.com/bb41fc75-4e2...	1617215163866

As of right now, there is no implementation for querying S3 objects using DynamoDB. Theoretically, a user would give the UniqueUserId+EpsilonTime as input and query would be made to DynamoDB returning the rows of documents uploaded together with the epsilon time. The rows include information on the corresponding S3 Object Key. The user would either get the link to S3 object or have the file displayed to them.

Resources:
https://aws.amazon.com/blogs/big-data/building-and-maintaining-an-amazon-s3-metadata-index-without-servers/
https://stacks.wellcomecollection.org/creating-a-data-store-from-s3-and-dynamodb-8bb9ecce8fc1

Implementation

The Lambda Function was created using this guide:
https://docs.aws.amazon.com/lambda/latest/dg/with-s3-tutorial.html
In addition to the policies mentioned in the documents,

AWS CLI was used to interface with AWS with creating the Lambda Function.
https://docs.aws.amazon.com/cli/latest/reference/lambda/index.html
This documentation has the commands needed to work with a Lambda function using AWS CLI

Image Recognition