AWS Configuration

GREYLIT PROJECT

March, 2019

Table of Contents

Chapter 1

SEARCH ENGINE ENHANCEMENTS AND
NATURAL LANGUAGE PROCESSING

INTRODUCTION

In this chapter we define the details of the functionality that was implemented to improve the current document's search engine in the actual web application. The current search process only takes into consideration the title of the documents, that in most of the cases, is not significant at all with their content. Nowadays, the users have high expectations about response times of the applications letting them search, find, and use the data in an efficient manner.

Several potential solutions had been analyzed, taking into consideration several aspects such as costs and implementation complexity. Also, we attempted to optimize our solutions by taking advantage of the current cloud solutions and the current functionality that is already in place.

PROBLEM DEFINITION

GreyLit is offering services to people who want to publish, read, and share information about several health topics. Using this web application, the users are able to sign up and create a profile to access the platform. The current existing roles identified could be classified in the following:

1. Publisher: this role is related to the person that will be signed up, complete their personal information, and upload (publish) documents. Typically, this role will not use the search engine, but should have the functionality available.
2. Member: a member is a main user of the search functionality. This type of user will search documents related to specific terms and topics.
3. Newsletter creator: this user works for GreyLit but has to use the functionality to create weekly newsletter.

Currently, the search engine uses keywords that the user enters, and these words are compared against the titles of the documents. The limitation with this approach is that sometimes there are documents that contains significant information about some specific topics, and this may not be reflected in the title. Another limitation is that the documents have no relationship among them. For example, if several publishers write about a specific disease there is no way to link them and use this information to improve the user experience.

Finally, and most importantly of all, when the size of the database grows and the number of documents stored increases, a robust platform to deal with it is needed, especially for expansion of their business opportunities in the future. The solution must be scalable, flexible, and easy to manage.

SCOPE

The scope of the project includes the following:

- Create a new screen (User Interface) to interact with the search engine. The user would select the search parameters and submit the key words to do the search. The options provided to search will be:
    - By relevant topic (index): the documents will be classified in different topics.
    - By relevant content: the user will insert one or more key words that will be used to look into the document and bring back those that match, in order of the overall relevance. In this case, every word will have a value that reflects how frequent it is in all the text.
    - Additional options to search, based on the catalogs of entities and sub-entities that the cloud platforms provide to classify the natural language.
    - Search by publisher or Author.
- Implement search engine (back-end), in one of the current functionalities available in Amazon Web Services (AWS). The solution will be created using the current build-in components, orchestration components, and programing elements when needed.

RESTRICTIONS

Restrictions defined about the functionality and also about security and management of the data were defined as follows:

1. The solution must be integrated with the current platform of GreyLit.
2. There is no preference for any platform in the cloud, but AWS was selected based on the evidence and support provided at the beginning of the project.
3. All personal information about the portal users must remain in the Canadian territory. That means the physical location of the server in any cloud service must be in Canada.

4. The solution uses the current databases with the current data available, including user's profiles, documents, and any other data required.
5. New UI uses the current sign in and sign up functionalities
6. All security policies and validation schemas were provided by GreyLit for new UI.
7. The searching functionality only supports English language for this project, but the components and architecture used to build the solution are scalable, and the capability for other languages could be added in the future with relative ease.

TIMELINE

The original estimated plan to develop all the functionality is detailed in Table 1. The estimated time did not include the tasks and efforts related to GreyLit technical staff. At the end of the project, we had 3 week delay caused by the time that the client expends to decide about the cloud solution to be used to implement the solution, including technical meetings, cost estimation, and supported services. The following table contains the final dates and expected time used in each stage.

Table 1 Project Phases and timelines

Phase	Name	Description	Start Date	Finish Date
1	Analysis	Analyze the current process, validate the changes and design the solution	September 17, 2018	September 30, 2018
2	Development	Build forms, analytics, tables and processes	November 2, 2018	February 1, 2019
3	Test	Test new processes, interfaces and integrations	February 4, 2019	February 22, 2019
4	Deploy	Move the solution from development environments to production	February 25, 2019	February 27, 2019
5	Transition	Monitoring the solution, support and transition to support team	February 28, 2019	March 1, 2019

PROPOSED SOLUTION

Based on the scope, restrictions, and requirements of the project, the following solution fulfills and covers every important aspect for GreyLit. The solution is described in the following three levels:

1. The business process and how it will change with the enhancements in the search process.
2. The interaction about the different components that conform the technical solution.
3. About the search engine and how all the components and services provided by AWS are orchestrated to solve the problem stated.

The selection of the cloud provider was supported by information researched and provided by the team. Three of the most prominent cloud service providers (Amazon, Microsoft, and Google) were considered in the analysis. After several meetings with the client and analyzing the information, the final decision was made. We had taken into consideration several factors to compare the cloud solutions that aligned with the business goals:

1. Current platform in place: to reduce the cost and use the current functionalities and modules that the current solution offers.
2. Interfaces: all new steps in the process and new functionalities must be integrated with the current process and components, reducing the communication steps and the interfaces between these components.
3. Use cloud build-in functionalities: most of the current cloud platforms offer built-in modules and components for natural language analysis, so there is no need to build them from scratch.
4. Cost: one of the most important aspects of the solution, every platform in the cloud has a cost and always depends on the resources that the solution will be or is using.
5. Future functionalities: when a new solution is created or extended, we have to be very clear on what is expected from the solution in the near future, planning ahead for future versions and projects, aligned with the continuity of business.

6. Scalability of the solution: the solution will be flexible to extend and grows accordingly with the business and provide the capability to integrate new solutions and platforms at any time.
7. Complexity: taking into consideration the complexity of build and maintain the desired functionality in the different options of cloud providers.

BUSINESS PROCESS

This analysis is focused on the search process only, we are not taking into consideration any other business process. We only validated the dependencies within other processes bud we could not find any affectation on them.

The process was mapped using a Pool Lane diagram (Figure 1), representing the interaction of the entities involved in all the process. We used orange arrows to represent the current steps and green arrows to illustrate the new steps that will be included in order to deploy the solution. This process involved 6 entities:

1. Publisher: user that uploads the documents into the web portal.
2. Search Engine: the portion of the solution that performs the search and analytics of the documents
3. Member: user that will interact directly with the search engine
4. Newsletter Creator: Greylit's employee that creates regular newsletters.
5. Google Analytics: in-place solution that currently stores the historic data related to the searches that every user has done.
6. Azure: in place solution, that has all the database of the current solution and all the data related to the documents.

We assumed that the users are already signed in and have valid rights to access the search form. The process will be as follows step by step:

1. The publishers upload one or more documents. These documents will have a "pending" status, and they will not be available to appear yet in the search results, until they are processed by the analytics tool.
2. The search engine will extract the data from the new documents, run the analytics over the text, and index the documents with the resulting data. All the analytics results will be stored in 2 additional tables (AZURE) that will be related to every document. These two tables provide the insights to search efficiently the content by topic and index.
3. It is important to clarify that this step is asynchronous. All documents will be processed several times a day with a job or something similar.
4. Once the documents are processed, their status will change to make them available to appear in searches.
5. For Members and Newsletter creators, the process follows the same. They could search by all the different ways mentioned above in the Scope section.
6. Every time these users perform a search or view a document, the relevant information will be stored in the current Google Cloud database.
7. For future versions of the application, the historic data could be used to do some analytics related to users' preferences and create connections with readers and publishers.

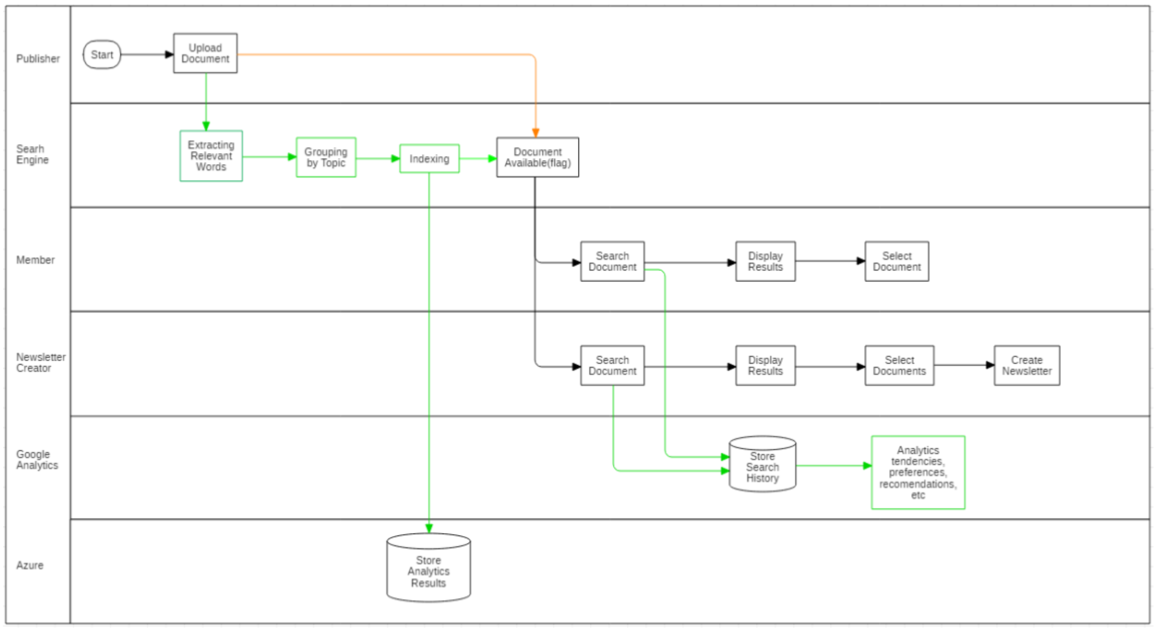


Figure 1 Business flow after the changes in the search engine

TECHNICAL SOLUTION

The solution that was implemented has two main components, the front end and the back-end. Both will interact with the current web portal, providing the enhancements in the search engine allowing the user explore and select documents more efficiently. The diagram (Figure 2) use colors to group the components that are related to the same process when the user is searching for documents. These different processes are:

1. Purple processes: these happen both at the front end and at the back-end. It describes how the relevant data that is needed to do the search will be retrieved and loaded into the form to let the users select their searching parameters.
    a. Load filter values: list catalog related to the type of filter that will be applied.
    b. Load publisher names: the searching tool could provide search by author.
    c. Load topic categories and sub categories: loading the catalogs provided by the cloud platform.
    d. Request values: represents the process that lives in the back-end and basically attends the requests from the front end to retrieved the data.

2. Pink processes: these happen at the front end and at the back-end. These boxes represent the new search engine solution.
    a. Search by topic (index): this box represents the request to search by the topic related to the documents.
    b. Search by Publisher: using the list of names provided by the application itself.
    c. Search by relevant words: one or several key words input by the user.
    d. Search Database: any request will be handled by the back-end to look for and retrieve the data stored in the tables related to the natural language analysis results created earlier.
    e. Show results: this portion runs at the front end, and displays the results of the search.
3. Orange processes: this portion of the process is the interaction with the web application that lets users upload the documents.
    a. Upload documents: front end functionality that lets the user select the documents from the local drive or any other sources.
    b. Save documents at storage unit: back-end functionality that will upload the documents and store them in the current drive already used in the cloud. This process will trigger the analytics steps (green boxes).
4. Green processes: analytics process will take the documents that are uploaded and identify the topics and relevant word in all the text for each document.
    a. Extract text: each document will be transformed into simplest and smallest version to be processed.
    b. Text Analytics: analytics models run over each document's text, identifying the most relevant words and topics then store this information in the database. After this process the document is available for search.

We have to highlight that all the documents that are already stored in the database have to be processed by analytics tools. This process will run just one time for every document, avoiding costs and reprocessing all the time that one user searches the data.

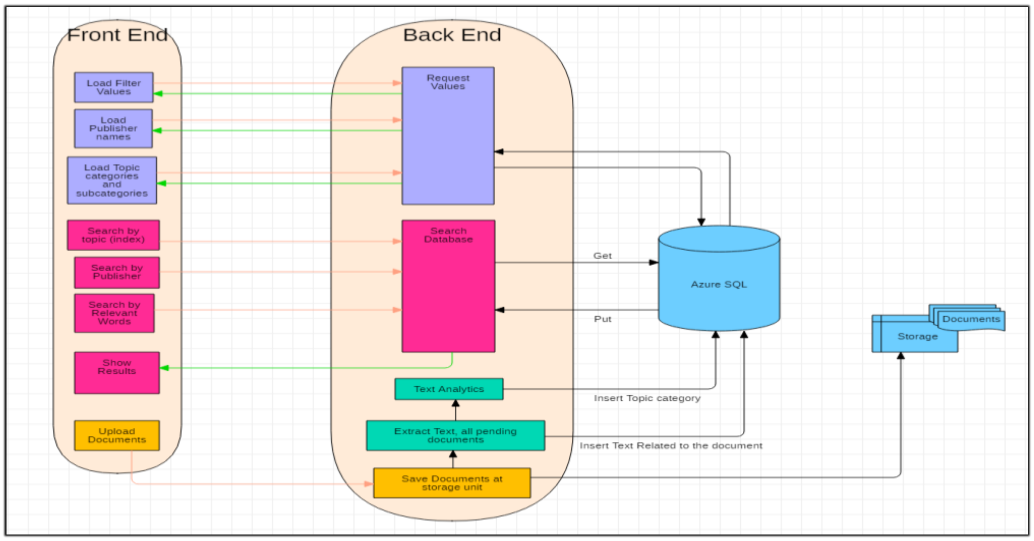


Figure 2 Diagram solution proposed

PLATFORM SELECTION

As a part of this analysis, a comparative table was provided with several aspects about the 3 different platform options selected by the client to build the solution. This table contains relevant information for making a comparison on the 7 features mentioned in the scope section (Table 2). Based on the information we provided, the client decided to create the solution in AWS. One additional topic was included which is the location of the servers that has the services needed to build the solution (natural language analysis). Following the comparative table with the analysis of the providers

Table 2 Aspects analyzed to evaluate the three main providers of Cloud Solutions

	Current Platform	**Interfaces**	**Build-in features provided**	**Cost**	**Future functionalities**	**Scalability**	**Complexity**	**Service Location**
Google Cloud	High	Med	Very Good	Med	Very Good	Low	Med	Out Canada
Amazon Comprehend	High	Med	Best	Low	Best	Low	Med	Out Canada
Microsoft Azure	Low	Low	Very Good	High	Very Good	Low	Med	In Canada

IMPLEMENTATION OF THE SOLUTION

With the information provided in the analysis, GreyLit team chose AWS to create and deploy the backend, mainly because the modules provided by them are easy to use and implement. AWS also is a low-cost service on storage and database hosting. This section will describe the back-end solution and details about the different components of AWS, the configuration that was made in each of them, and the details of the code created in Python language used in Lambda functions.

The following diagram (Figure 3) shows the solution in detail about the backend and the interactions between all of them, in order to create the final solution and to understand better how the integration with the front end solves the established problems.

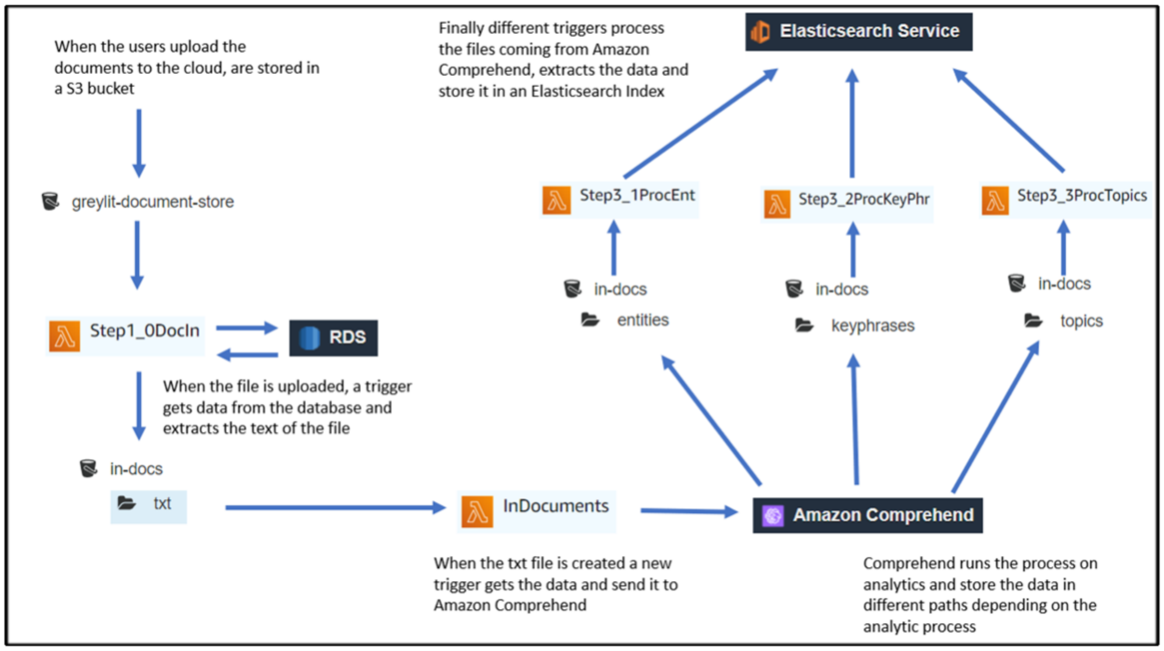

When the users upload the documents to the cloud, are stored in a S3 bucket
greylit-document-store
Step1_0DocIn
RDS
When the file is uploaded, a trigger gets data from the database and extracts the text of the file
in-docs
txt
InDocuments
When the txt file is created a new trigger gets the data and send it to Amazon Comprehend
Amazon Comprehend
Comprehend runs the process on analytics and store the data in different paths depending on the analytic process
in-docs
entities
in-docs
keyphrases
in-docs
topics
Step3_1ProcEnt
Step3_2ProcKeyPhr
Step3_3ProcTopics
Finally different triggers process the files coming from Amazon Comprehend, extracts the data and store it in an Elasticsearch Index
Elasticsearch Service

Figure 3 Diagram Solution AWS

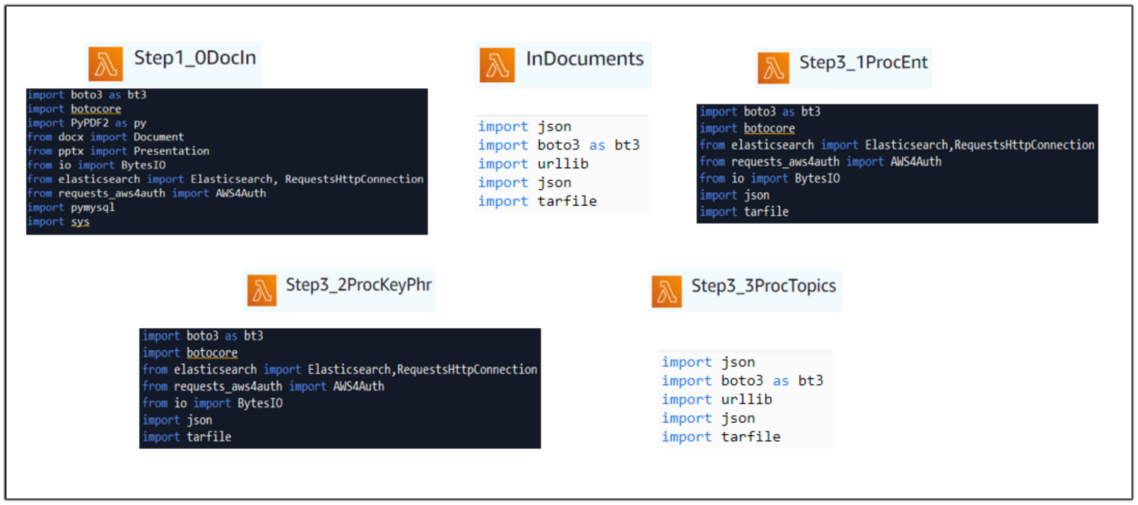
About Python integration

All the algorithms that were created and run in every lambda function were developed in Python 3.6, using Anaconda framework. Figure 4 shows each Lambda function with their related python libraries. Using the username ***greylit-comprehend*** in combination with the key and secret key, we were able to develop and test Python code from our local development environments.

The following libraries were used to create the final solution. Figure 4 shows the libraries used in each of the Lambda Functions:

- **Boto3**: this is the library provided by python to interact with AWS services. Easily you can create an object and make reference to an object or service in AWS.
- **Botocore**: is a set of complementary libraries requested by AWS Lambda services.
- **Elasticsearch**: is the library that provides the interactions with Elasticsearch service. RequestHttpConnection is an object needed as a parameter when the connection with Elasticsearch is stablished, providing the credentials to sign in.
- **Request_aws4auth**: the library provides an object that allows to use the same credentials used in the connection to AWS to establish the connection using http protocol to Elasticsearch.
- **IO**: library used to interact with the files, to create and extract data in and out from the s3 bucket.
- **Json**: needed to manipulate the data, create the indexes, and store it in Elasticsearch, that, in a nut shell, is a Json file.
- **Tarfile**: all outputs given by Amazon Comprehend are compressed files so this library provides the functionality to un-compress the files and extract the results from them.

- **PyPDF2**: library used to extract the text from the pdf files. We also used pdfminer, that is a great library but currently AWS has some portability issues with the library.
- **Docx**: library used to extract the text from .docx files.
- **Pptx**: library used to extract the text from .pptx files.
- **Pymysql**: this library provides the tools needed to connect to MySQL database. We have to extract the Id, author's name, and abstract of the documents to be included in Elasticsearch index.



```
import boto3 as bt3
import botocore
import PyPDF2 as py
from docx import Document
from pptx import Presentation
from io import BytesIO
from elasticsearch import Elasticsearch, RequestsHttpConnection
from requests_aws4auth import AWS4Auth
import pymysql
import sys
```

```
import json
import boto3 as bt3
import urllib
import json
import tarfile
```

```
import boto3 as bt3
import botocore
from elasticsearch import Elasticsearch,RequestsHttpConnection
from requests_aws4auth import AWS4Auth
from io import BytesIO
import json
import tarfile
```

```
import boto3 as bt3
import botocore
from elasticsearch import Elasticsearch,RequestsHttpConnection
from requests_aws4auth import AWS4Auth
from io import BytesIO
import json
import tarfile
```

```
import json
import boto3 as bt3
import urllib
import json
import tarfile
```

Figure 4 Lambda functions and their python libraries

About AWS

Different components have been configured in AWS platform and some of them are not listed in the Figure 3 diagram because they have complementary functions and not all are involved in the flow of data and transformation. Following items describe all the components used and provide short descriptions of their contribution and how they were implemented and deployed:

AWS IAM

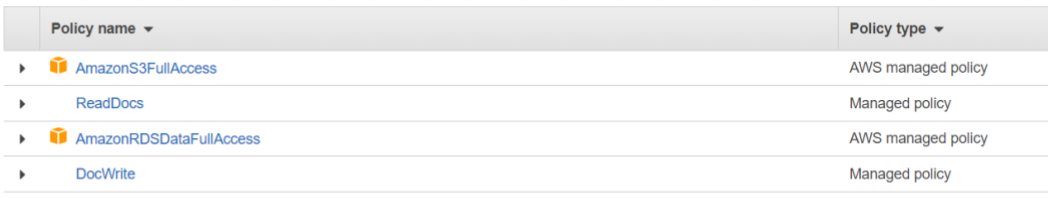
Provides management and control about the users, their roles and the interaction between different security policies that allow AWS services execute other services in their behalf.

GreyLit's configuration includes several roles that were created to access the database and to access AWS console, but also policies used for Lambda functions to interact with other services.

User: to access the general services from our development and local environments is ***greylit-comprehend.***

Roles: we have created 2 roles to be attached to the lambda functions in order to execute the algorithms and also to call the Elasticsearch and Comprehend services.

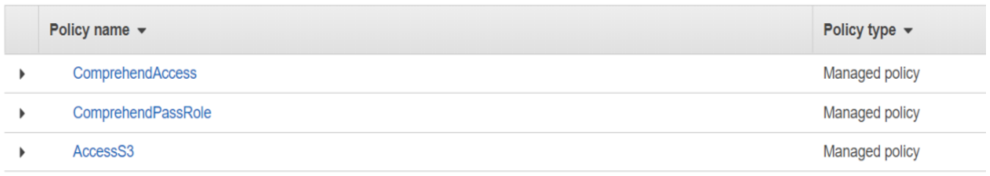
1. *Role DocStep1*: this role has access to s3 services and resources, it is mainly used when the algorithm is accessing the files and writing the txt output files. The Figure 5 shows the Policies attached to this role, ReadDocs and DocWrite provides the specific grants to access GreyLit's buckets.



Policy name	Policy type
AmazonS3FullAccess	AWS managed policy
ReadDocs	Managed policy
AmazonRDSDataFullAccess	AWS managed policy
DocWrite	Managed policy

Figure 5 Policies attached to role DocStep1

2. *Role DocStep2*: this role lets Comprehend service to have access to s3 GreyLit's buckets. Given that Comprehend will run asynchronic processes they have to have access to write new files into s3 buckets independently of the lambda algorithm that was previously executed (Figure 6).



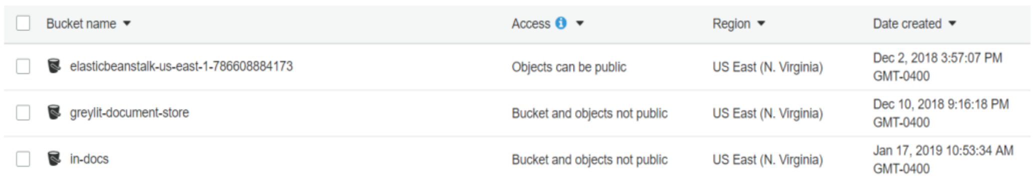
	Policy name	Policy type
▸	ComprehendAccess	Managed policy
▸	ComprehendPassRole	Managed policy
▸	AccessS3	Managed policy

Figure 6 Policies attached to role DocStep2

Amazon s3

AWS provides serverless storage services. A bucket is the concept used to define a location into the service where the user can create new folders and store any kinds of documents. GreyLit creates 3 main buckets (Figure 7) for business running:

- **elasticbeanstalk-us-east-1-786608884173:** this bucket has all the files related to the current website (GreyLit Portal) and all the configuration files related to it.
- **GreyLit-document-store:** this bucket works as a file system unit, where all the documents are stored when the users upload them. Here starts the process when a trigger detects if a new file was uploaded. The structure used by GreyLit is that, for every file, a new folder with their corresponding portal's id is created.
- **In-docs:** this bucket was created to be used to move and store the data through all the process of transformation and extraction of the text from the original documents.



	Bucket name	Access	Region	Date created
☐	elasticbeanstalk-us-east-1-786608884173	Objects can be public	US East (N. Virginia)	Dec 2, 2018 3:57:07 PM GMT-0400
☐	greylit-document-store	Bucket and objects not public	US East (N. Virginia)	Dec 10, 2018 9:16:18 PM GMT-0400
☐	in-docs	Bucket and objects not public	US East (N. Virginia)	Jan 17, 2019 10:53:34 AM GMT-0400

Figure 7 GreyLit's buckets

Amazon RDS

RDS provides a database services that can be configured and used with any type of database engine. GreyLit is using MySQL database engine as a part of their portal. The back-end process only takes into consideration the table documents, to extract the Authors' names and the Abstract of the document based in its id.

Amazon Lambda

Lambda is a serverless service that creates pieces of functional programs and logic that executes several tasks and could be controlled with their triggers that respond to mostly all the different events of the rest of AWS services. Lambda functions control how the data related to the text of the files is moving from one step to the next one and have the orchestration function to deliver the final results into Elasticsearch. The following five lambda functions were created in this project:

- **Step1_0DocIn (Figure 8)**: this lambda function has a trigger that is listening to the greylit-document-store bucket. When new files are uploaded, the trigger will extract the text from ppt, docx, or pptx files. Then it verifies if the index template exists in Elasticsearch. If does not exist, the template is created. Next it checks if the index exists and if does not exist, it is created as well. It then goes to the database to extract the Author's name and the abstract to be included into a new entrance in Elasticsearch. Sections related to topics and key-phrases will be updated later.

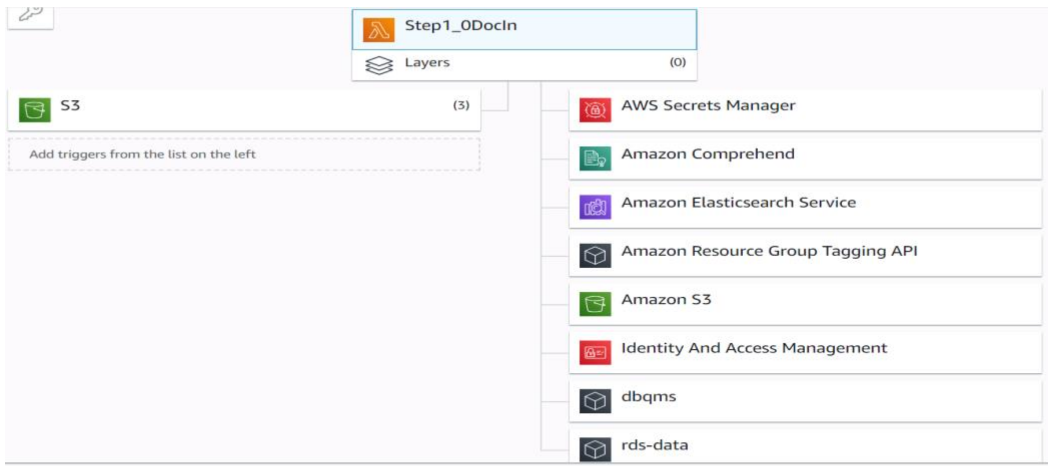

Step1_0DocIn
Layers (0)
S3 (3)
Add triggers from the list on the left
AWS Secrets Manager
Amazon Comprehend
Amazon Elasticsearch Service
Amazon Resource Group Tagging API
Amazon S3
Identity And Access Management
dbqms
rds-data

Figure 8 Step1_0DocIn Lambda function definition

- **InDocuments (Figure 9)**: this lambda function triggers when a new file is created in the bucket in-docs folder txt. The algorithm is taking the txt file created by the first lambda function and then calls the Comprehend Service to extract the key phrases, entities and topics.

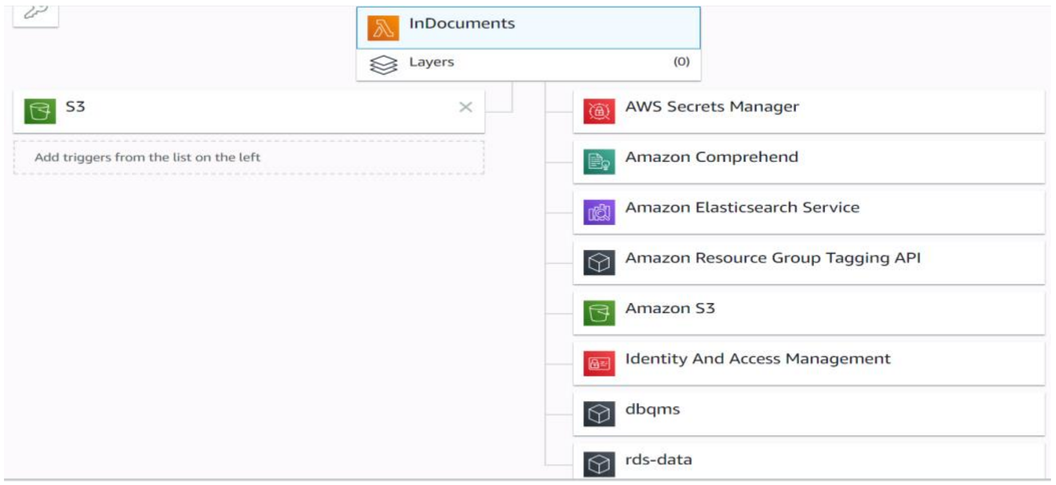

InDocuments
Layers (0)
S3
Add triggers from the list on the left
AWS Secrets Manager
Amazon Comprehend
Amazon Elasticsearch Service
Amazon Resource Group Tagging API
Amazon S3
Identity And Access Management
dbqms
rds-data

Figure 9 InDocuments lambda function definition

- **Step3_1ProcEnt (Figure 10):** when the asynchronic job to identify the entities finished in Comprehend, the compressed file is stored in s3 bucket (in-docs/entities), and the trigger process the file. It will look fist for the id number to identify the corresponding document, then use this id to retrieve the row in Elastic search. Once the row is retrieved, the algorithm updates the entities section with the results previously processed.

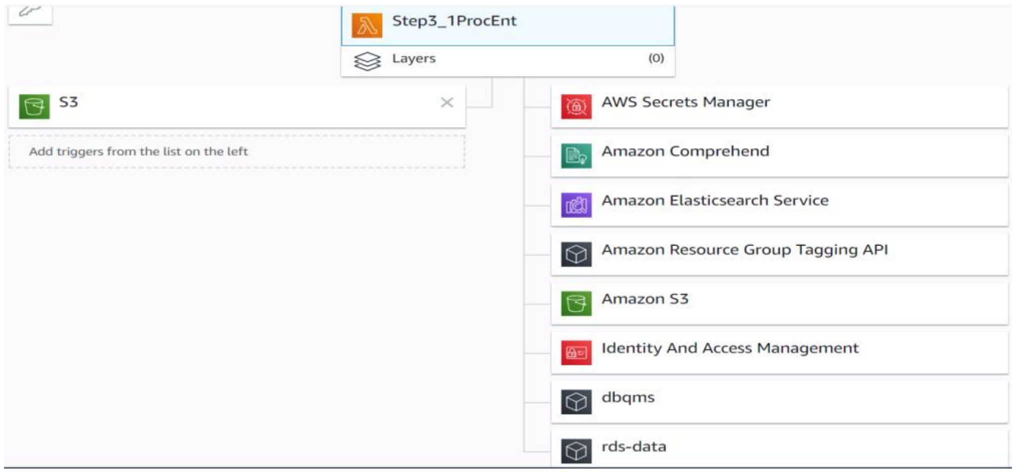

Step3_1ProcEnt
Layers
(0)
S3
Add triggers from the list on the left
AWS Secrets Manager
Amazon Comprehend
Amazon Elasticsearch Service
Amazon Resource Group Tagging API
Amazon S3
Identity And Access Management
dbqms
rds-data

Figure 10 Step3_1ProcEnt lambda function definition

Step3_2ProcKeyPhr (Figure 11): this lambda process is triggered when Comprehend's job finishes to do the analytics related to the key phrases. Then the algorithm opens the compressed file, extracts the data, and sends it to Elasticsearch. Similarly, to the previous lambda function, it recovers the ID from the txt file name, then looks for the row that already exists in Elasticsearch, retrieves the data, and updates the section of KeyPhrases with the corresponding data.

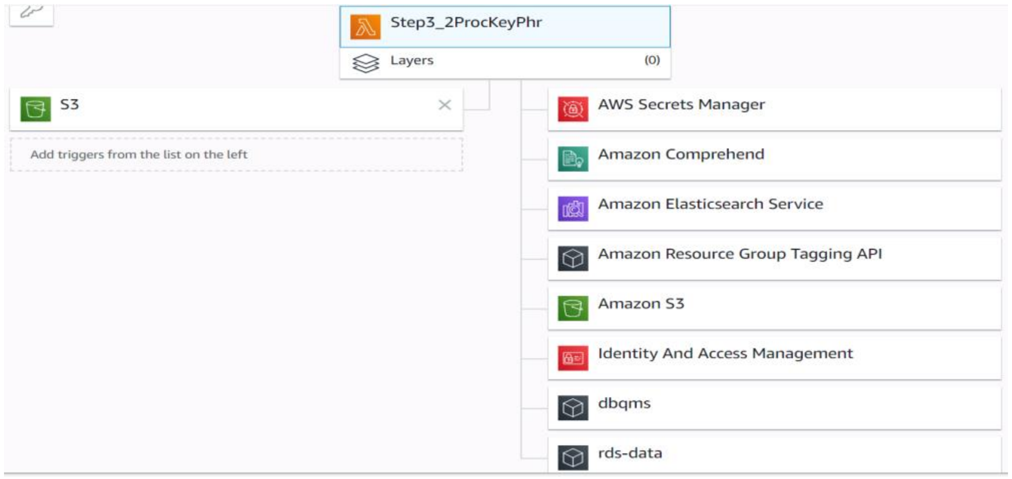

Step3_2ProcKeyPhr
Layers
(0)
S3
Add triggers from the list on the left
AWS Secrets Manager
Amazon Comprehend
Amazon Elasticsearch Service
Amazon Resource Group Tagging API
Amazon S3
Identity And Access Management
dbqms
rds-data

Figure 11 Step3_2ProcKeyPhr lambda function definition

- **Step3_3ProcTopics (Figure 12):** this function takes the data coming from Comprehend related to the topics. The difference in this lambda is that the input data coming in the compressed file are 2 csv files, the first one contains a list of the topics with the score and the second one contains the topic id with the document number.

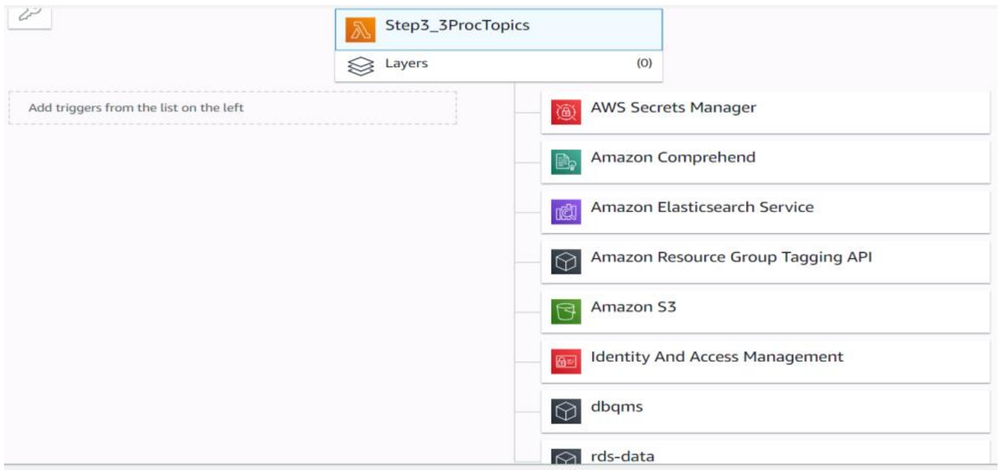
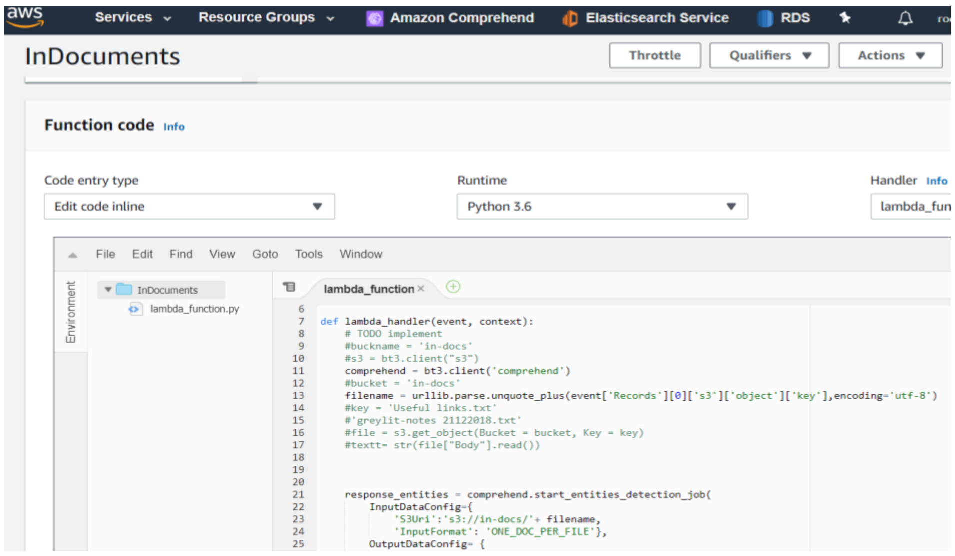

Step3_3ProcTopics
Layers
(0)
Add triggers from the list on the left
AWS Secrets Manager
Amazon Comprehend
Amazon Elasticsearch Service
Amazon Resource Group Tagging API
Amazon S3
Identity And Access Management
dbqms
rds-data

Figure 12 Step3_3ProcTopics lambda definition

AWS provides a tool to create your own code directly into a GUI (Figure 13) that runs in the cloud. This GUI is really useful especially when testing and debugging are needed to make sure your scripts are running as expected.



Code entry type: Edit code inline
Runtime: Python 3.6
Handler: lambda_fun

```

def lambda_handler(event, context):
    # TODO implement
    #buckname = 'in-docs'
    #s3 = bt3.client("s3")
    comprehend = bt3.client('comprehend')
    #bucket = 'in-docs'
    filename = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'],encoding='utf-8')
    #key = 'Useful links.txt'
    #'greylit-notes 21122018.txt'
    #file = s3.get_object(Bucket = bucket, Key = key)
    #textt= str(file["Body"].read())



    response_entities = comprehend.start_entities_detection_job(
        InputDataConfig={
            'S3Uri':'s3://in-docs/'+ filename,
            'InputFormat': 'ONE_DOC_PER_FILE'},
        OutputDataConfig= {
```

Figure 13 Lambda GUI to edit and test python code

The GUI has some limitations and cannot be used in every type and size of scripts given that AWS only provides a default small number of libraries installed in their servers. Only two of our lambda functions were installed and tested in this environment, the rest of functions were uploaded using a secondary method provided by AWS. The secondary method consists in to create a compressed zip file containing the library dependencies and the python script, then upload it directly to AWS to be executed. Once this package is attached to the lambda function it is deployed in the environment and is ready to be used.

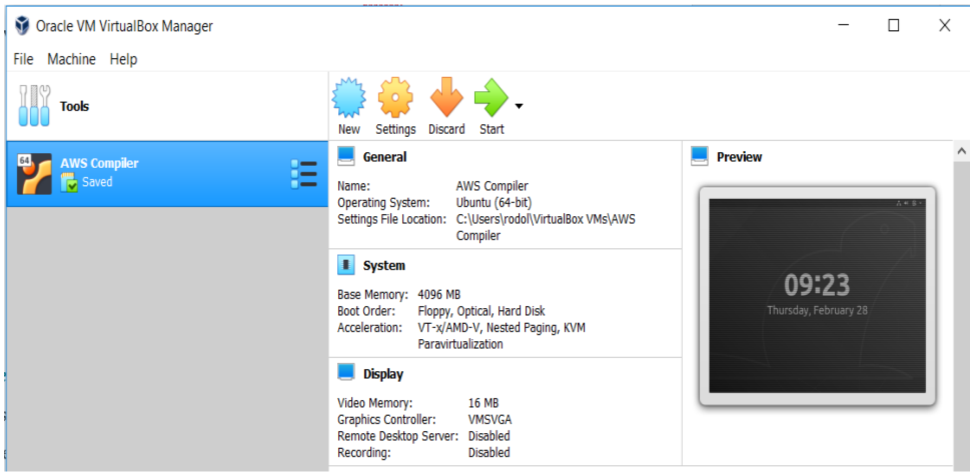
The process to create a python package for lambda functions follows several steps that have to be done in certain order and keeping attention to the independency of the virtual environments related to the libraries that the script will use.

Another important detail about all this process is that the complete creation of the python environment, compilation and creation of each package have to be done in a Linux environment. AWS servers are implemented and managed by Linux platforms and any package compiled on Windows or IOS environments will fail at running time. The following give the details of how to create and upload the packages into AWS lambda functions.

1. Create the environment on Linux. As our team has Windows based computers, we had to create a new environment to do all the configuration. We had two

options and both of them are good enough to fulfill the requirements, one is to install Docket and create the virtual environment and the second one is to create a virtual machine with Linux or similar operative system.

2. We choose the option to install a virtual machine using Oracle's Virtual Box, then installed Ubuntu 64-bit, version 18.04.2 (Figure 14). The deployment could be done using Docker if preferred.



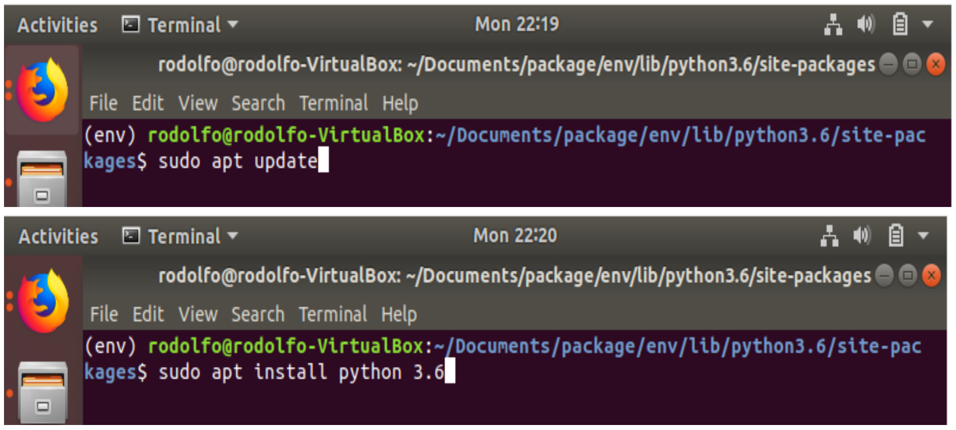
```
Name:                    AWS Compiler
Operating System:        Ubuntu (64-bit)
Settings File Location:  C:\Users\rodol\VirtualBox VMs\AWS
                         Compiler

Base Memory:   4096 MB
Boot Order:    Floppy, Optical, Hard Disk
Acceleration:  VT-x/AMD-V, Nested Paging, KVM
               Paravirtualization

Video Memory:            16 MB
Graphics Controller:     VMSVGA
Remote Desktop Server:   Disabled
Recording:               Disabled
```

Figure 14 Ubuntu Virtual Machine

3. Once the virtual machine was successfully created, we proceed to install python 3.6. (Figure 15).

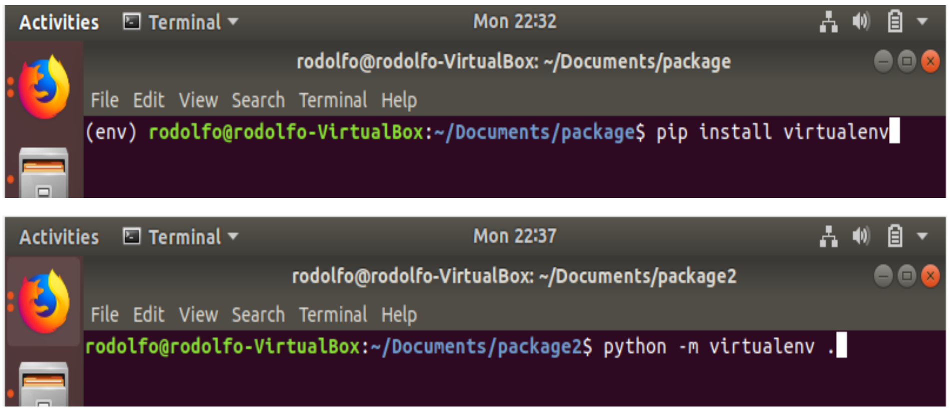

Activities
Terminal
Mon 22:19
rodolfo@rodolfo-VirtualBox: ~/Documents/package/env/lib/python3.6/site-packages
File Edit View Search Terminal Help
Activities
Terminal
Mon 22:20
rodolfo@rodolfo-VirtualBox: ~/Documents/package/env/lib/python3.6/site-packages
File Edit View Search Terminal Help

```
(env) rodolfo@rodolfo-VirtualBox:~/Documents/package/env/lib/python3.6/site-pac
kages$ sudo apt update
```

```
(env) rodolfo@rodolfo-VirtualBox:~/Documents/package/env/lib/python3.6/site-pac
kages$ sudo apt install python 3.6
```

Figure 15 Update applications and install Python

4. Once python was installed, we installed the library virtualenv to create the virtual environment that basically is a new full environment of python that stores their own dependencies and structures (Figure 16).


Activities
Terminal
Mon 22:32
rodolfo@rodolfo-VirtualBox: ~/Documents/package
File Edit View Search Terminal Help

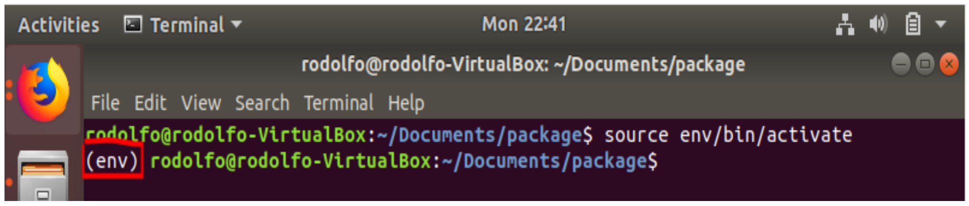
```
(env) rodolfo@rodolfo-VirtualBox:~/Documents/package$ pip install virtualenv
```

Activities
Terminal
Mon 22:37
rodolfo@rodolfo-VirtualBox: ~/Documents/package2
File Edit View Search Terminal Help

```
rodolfo@rodolfo-VirtualBox:~/Documents/package2$ python -m virtualenv .
```

Figure 16 Creating virtual environment

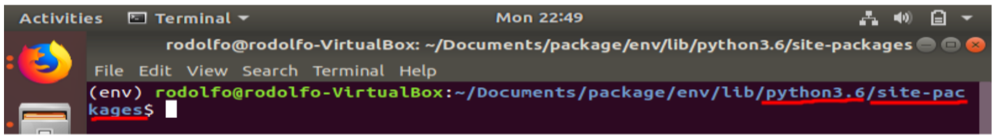
5. Once the environment is created, we have to activate it, in order to work on it and to install the libraries. Once the environment is activated, its name will appear at the front of the user in the terminal (Figure 17).


Activities
Terminal
Mon 22:41
rodolfo@rodolfo-VirtualBox: ~/Documents/package
File Edit View Search Terminal Help

```
rodolfo@rodolfo-VirtualBox:~/Documents/package$ source env/bin/activate
(env) rodolfo@rodolfo-VirtualBox:~/Documents/package$
```

Figure 17 Activating virtual environment

6. Now the environment is active, we had to look for the folder named site-packages, full path is showed in the image (Figure 18).


Activities
Terminal
Mon 22:49
rodolfo@rodolfo-VirtualBox: ~/Documents/package/env/lib/python3.6/site-packages
File Edit View Search Terminal Help

```
(env) rodolfo@rodolfo-VirtualBox:~/Documents/package/env/lib/python3.6/site-pac
kages$
```

Figure 18 Looking for site-packages folder

7. Under this path we installed the following libraries.
   a. pip install pdfminer.six
   b. pip install jsonlib-python3
   c. pip install python-pptx
   d. pip install python-docx
   e. pip install PyPDF2
   f. pip install pandas
   g. pip install PyMySQL
   h. pip install elasticsearch
   i. pip install requests-aws4auth

8. Next, we had to copy the script in this folder and compress the all of them into a single zip file (Figure 19).

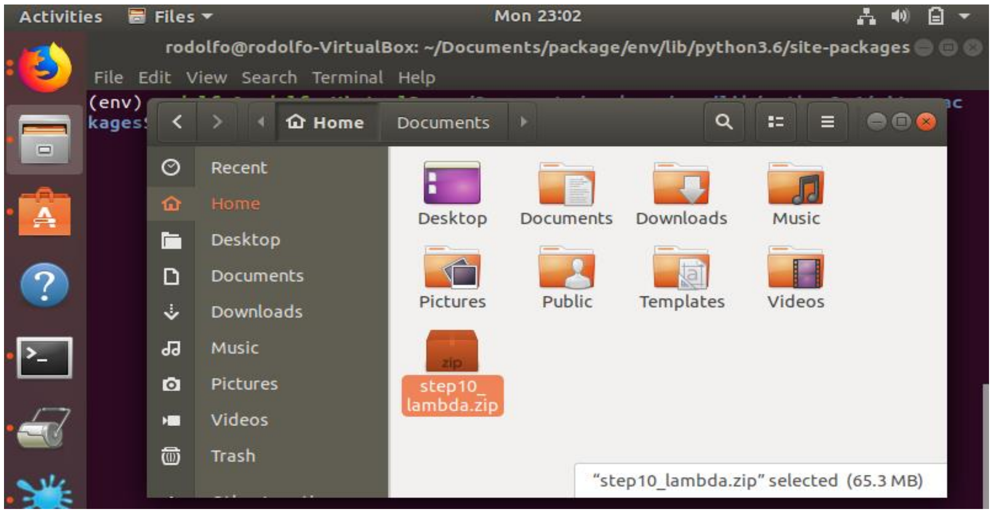
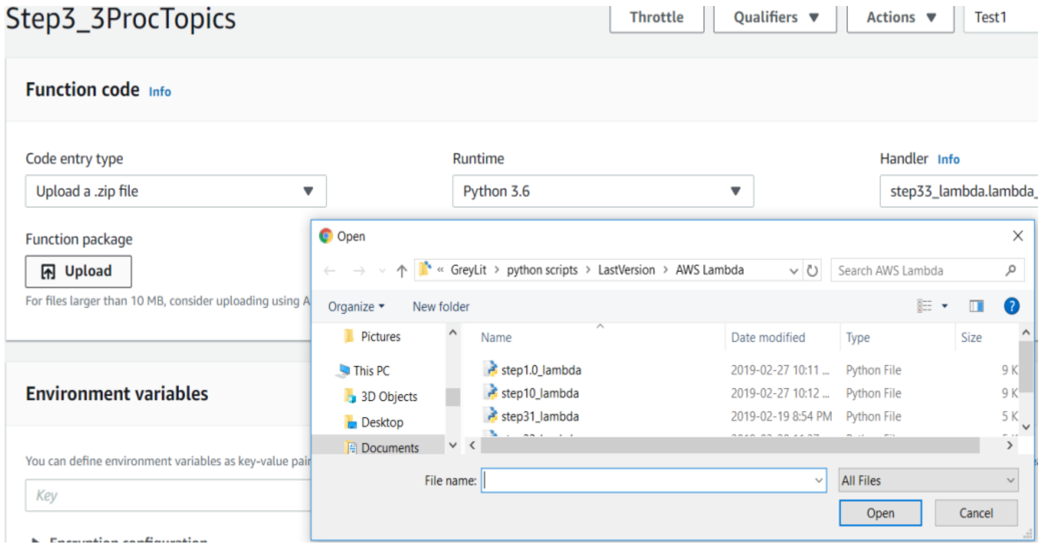


Figure 19 Lambda zip file

9. Finally, we had to upload the package to the lambda function that we want to update. If we upload the package, we will be unable to edit the python code from the GUI (Figure 20).



Code entry type: Upload a .zip file

Runtime: Python 3.6

Handler: step33_lambda.lambda_

Function package: Upload

For files larger than 10 MB, consider uploading using A

You can define environment variables as key-value pai

GreyLit › python scripts › LastVersion › AWS Lambda

Search AWS Lambda

Pictures, This PC, 3D Objects, Desktop, Documents

Name	Date modified	Type	Size
step1.0_lambda	2019-02-27 10:11 ...	Python File	9 K
step10_lambda	2019-02-27 10:12 ...	Python File	9 K
step31_lambda	2019-02-19 8:54 PM	Python File	5 K

File name:

All Files

Figure 20 Uploading zip package to Lambda

Amazon Comprehend

Comprehend takes care of the data, creates all the analytic process, finds the relevant key phrases, the entities, and the topics that are relevant in the text of the documents. The service is able to process one document at a time or a list of documents at a time, depending on the "InputFormat" flag parameter, used when the process is called. It is important to mention that it could run in two modes:

- Synchronic processing: real time response of the service, it is a RESTFUL application that has a limitation with the size of the text characters that can process at a time. To use this mode the text length has to be equals or less than 500 characters.
- Asynchronic processing: is a batch processing that takes the text and sends it to a queue, then when the analytic process finishes, the resulting compressed file should be stored in a s3 bucket instance or in an AWS EC2 instance.

AWS Elasticsearch

Elasticsearch is a service that is based on indexes that store the data in a non-relational structure. We can create several indexes depending on the type of data and also on how this data could be used.

We have created an index's template to let the lambda functions insert and update the data in a common structure defined since the beginning of the process. This common structure provides the flexibility to add the information related to the key phrases and entities by the rest of the Lambda functions at different times. Every index follows the same structure base structure to store the data:

- Index: that is comparable with the concept of database.
- Type: that is comparable with the concept of a table.
- Document: that is comparable with the concept of a row.

CONCLUSION

In general terms, the project was directly related with machine learning techniques and algorithms, taking advantage of the cloud services. We certainly can affirm that the techniques and models reviewed and studied in all the courses of the program help us to understand the current business problems and model a good solution. We had created an innovative solution, that will help to improve the current client's platform performance, but also help the client to increase their business across the region.

In the process of implementing the model we studied and analyzed the different cloud providers, their platforms, and the different approaches they offer to solve the same problems. We recognize the importance to know about these technologies and the wide set of function, services, and combinations of the cloud solutions.

We recognized the potential of our solution, that could be used in the future to solve other problems related to the same nature of the business, for example create new search indexes, or expand the functionality to be used on searching of videos and pictures related to studies of medicine and health areas. The framework can potentially be applied to other similar types of analysis as well, expanding the functionality with additional components provided by AWS or create the customized tools to be used for Greylit's platform. In overall we are sure the solutions are scalable and has a good foundation to be the base of a robust and long-term solution for the company.

TIMELINE CHRONICLE

Project	Phase	Month	Description	Time
Greylit	Analysis	Aug-18	First meetings with Scott Rogers and Cora Cole to define the scope and the expectations of the project. Created project Charter, estimating tasks and timelines. Begin the research on the cloud providers related to natural language processing.	22 hours
Greylit	Analysis	Sep-18	Created analysis document with all the research made about the different cloud providers. Created the business diagram with the changes proposed, meeting with the client to approve the change and present the research results of the comparison in the different cloud providers.	91 hours
Greylit	Analysis	Oct-18	Analyzed and compare additional request made by the client only related to Azure and AWS. Presentation of the second analysis based only in the natural language processing tools provided AWS and Azure. Looking for all the technical details about the AWS components to build the solution	29.5 hours
Greylit	Development	Nov-18	Project was stand because the setup of the environment with AWS. Have follow up meetings and started exploring the library boto3 in python to build the lambda functions	12 hours
Greylit	Development	Dec-18	Created Elasticsearch instance, checked the roles and policies needed to access the environment outside of the VPC. Developing the code to connect, authenticate and process text into Elasticsearch service. Creating lambda functions for steps 2 and 3 basically to send the data into Comprehend. Started working with PDFminer and PyPDF2 libraries to extract text from pdf files.	67 hours
Greylit	Development	Jan-19	Created new bucket to work with the files that are shared in the process of extraction and analysis. Create folders and flow of the files between them. Created the roles and policies for s3 and lambda. Finished step 2 and 3. Started checking the process to create lambda packages. Connecting to the database and extracting data from the documents	119 hours
Greylit	Testing	Feb-19	Creating test cases, uploading and testing lambda packages in AWS platform. Testing text extraction from different file types. Testing flow of files coming from and going to comprehend and Elasticsearch. Querying data from Elasticsearch and Greylit's database	48 hours
			Total	**388.5 hours**

Appendix

SECTION 1 PROJECT CHARTERS

22.08.2018

Project Charter: **GreyLit Search and Indexing.**

Background

GreyLit is an organization that offers several services to help people to find and store hundreds of documents related to health topics. The organization has decided to add new search and indexing functionalities to enhance their services and create a better user experience.

Goals

This project is aimed at developing new functionalities around the current services, specifically to create new user experience related to searching and indexing the documents. The searching solution will:

1. Indexing documents by a specific topic with the highest relevance overall
2. Searching documents by topic
3. Searching documents by one or more keywords
4. Look and present matching word results by relevance, in other words how many times the word (words) appears in the document, and which of them are more relevant in descendant order.

Scope

The scope of the project includes the following:

- UI to interact with the searching engine (functionality in Azure or AWS), to select options and submit the search parameters described in Goals section.
- Implement search engine, in one of the current functionalities available in Azure or AWS, this deliverable will better be defined in next stages of the project
- Search functionality as follows:
  - Search documents by one or more words: the search will be made in all the text of the entire document, not only on the title
  - Relevant percentage indicator: for each word used in the search will be presented the percentage of relevance in the specific document
  - Search by topic: in overall, the documents could be classified by the main topic developed in their entire content, could be one or more words.

Key Stakeholders

Client	GreyLit
Sponsor	Cora Cole, Yasushi Akiyama
Project manager	TBD

Project team members	Scott Rogers, Rodolfo Garcia
End Users	TBD

Project Milestones

Start Date: 22/08/2018
Analysis 1: 17/09/2018
Development 2: 02/11/2018
Testing 3: 04/02/2019
Deployment 4: 25/02/2019
End Date: 01/03/2019

Project Budget

Estimated Hours: 300
Cost for each hour: $TBD
Estimated Total Budget: $TBD

Constraints, Assumptions, Risks and Dependencies

Constraints	The solution must be integrated with the current platform of Geylit. There is no preference for AWS, Azure or any other cloud services provider. All personal information about the portal users must remain in Canadian territory. That means the physical location of the server in any cloud service must be in Canada.
Assumptions	Sponsor, key users and technical resources will be available to response questions and requests in any stage of the project Additional technical resources could be added to validate specific technical requirements in different areas (development and server configuration) The solution will use current databases with the current data available New UI will use the current sign in and new user functionalities All security policies and validation schemas (data) will be provided by GreyLit for new UI Searching functionalities will support only English language for this project
Risks and Dependencies	Fixed Budget – no additional functionalities or deviations in timelines could be allowed because the budget restrictions. Compatibility from previous solution platform with new proposed solution is unknown at the beginning of the project. Complexity and technical requirements to build natural language functionalities are not evaluated at the beginning of the project. Scalability is very important aspect to be taking into consideration for future functionalities and multiple language support.

Approval Signatures

Project Client	Project Sponsor	Project Manager

SECTION 2 ELASTICSEARCH TEMPLATES AND INDEXES

Index template

```
#Index Template
indexDoc = {
        "index_patterns": "doc*",
        "settings": {
        "number_of_shards": 1
         },
        "mappings" :
          {
           "_doc":
            {
             "properties" :
                {
                 "ID" : {"type" : "long"},
                 "Title": {"type" : "text"},
                 "IPAdd": {"type" : "text"},
                 "Author": {"type" : "text"},
                 "Abstract": {"type": "text"},
                 "FileLocation": {"type" : "text"},
                 "createdDate" : {"type" : "text"},
                 "Entities": {"type": "object"},
                 "KeyPhrases": {"type": "object"},
                 "Topics": {"type": "object"},
                }
             }
           }
        }
```

Index structure

```
IndexDoc = {
        "ID"                : FileId,
        "Title"             : Title,
        "IPAdd"             : IP,
        "Author"            : Author,
        "Abstract"          : Content,
        "FileLocation"      : Location,
        "createdDate"       : Date,
        "Entities"          : Entities,
        "KeyPhrases"        : Phrases,
        "Topics"            : Topics
       }
```