

Part A. Read AWS official documentation on SageMaker and Comprehend. Learn how these services are used in event-driven applications and explore various use cases.

AWS SageMaker [4]

AWS SageMaker is a Machine Learning service provided by Amazon. We can build and train ML models and deploy on the environment using SageMaker hosting services. It provides an inbuilt Jupyter notebook and some common ML algorithms which work effectively on a large dataset.

SageMaker also provides Apache Spark library that integrates Spark with SageMaker. We can use Apache Spark for preprocessing the data and then SageMaker for training the data using ML algorithms. There are predefined functions for the ML algorithms, one of which is “KMeansSageMakerEstimator” for K means algorithm.

We can use AWS SageMaker built-in ground truth types for automatic data labeling which can help in increasing accuracy and create varied train data.

Last but not the least, we can create our custom algorithms on SageMaker. SageMaker’s built-in algorithms use docker for each algorithm which makes building and deploying faster. Hence to create our custom algorithm, we can create our container which can contain scripts, libraries, dependencies, etc. Hence SageMaker provides flexibility to use any script or algorithm, which can be shipped faster due to containerization.

We can apply Machine Learning to many places to make predictions. For instance, in a tourism application, we can use AWS SageMaker to predict the places people would be interested to visit in the upcoming season by their profile. With past years' data which would contain information about which type of people usually visit what type of places in the season, we can predict the places a person would be interested in. This can be achieved with decent accuracy by the simplest ML algorithm i.e. K-means algorithm.

AWS Comprehend [5]

AWS Comprehend is a service that uses Natural Language Processing to fetch insights from the text or language. It gets insights by sentiments, entities, language, grammar, etc. Comprehend can find entities from text such as named entities, places, locations, verbs, adjectives, etc.

We can also count some phrases for a context. For example, in an article about a Cricket match, it can fetch the score, match name, venue, etc. AWS Comprehend can work with 100 languages and can give the sentiment of the paragraph or a conversation in any of the supported languages. POS tagging is a very important part of NLP and Comprehend can be used to find POS of each word in the document.

There are many other things which Comprehend does, some of them are Custom classification, Custom entities, Topic modeling, etc. For Custom classification, we can provide proper labels that describe best for each document, and finally, we can train our classifiers on those labels. AWS Comprehend and AWS SageMaker both can be used together for many ML tasks and are a great combination to achieve a better solution. For topic modeling, AWS Comprehend can analyze the corpus of text and organize it based on similar topics and entities found.

AWS Comprehend can process the document in 3 ways based on the situation. When we must analyze large documents and many documents, we can use asynchronous batch processing, where we store the documents in an S3 bucket and can start one or more jobs at the same time. When we want synchronous document processing, AWS Comprehend can receive up to 25 documents at the same time to process, and Comprehend provides a list of responses.

For a tourism application, we can analyze the reviews provided by the users of the application and get insights from thousands of people. We can find sentiments of the reviews and analyze for which place or tour, the reviews are good or bad and can improvise accordingly.

Part B. Build an event-driven serverless application using AWS SageMaker. In this part of the assignment, you need to use an S3 bucket, and Lambda Functions.

- a. Create your 1st S3 bucket SourceDataB00xxxxxx and upload the files (from 001 to 299) given in the Train folder. You need to write a script or use the SDK to upload the files on the bucket.

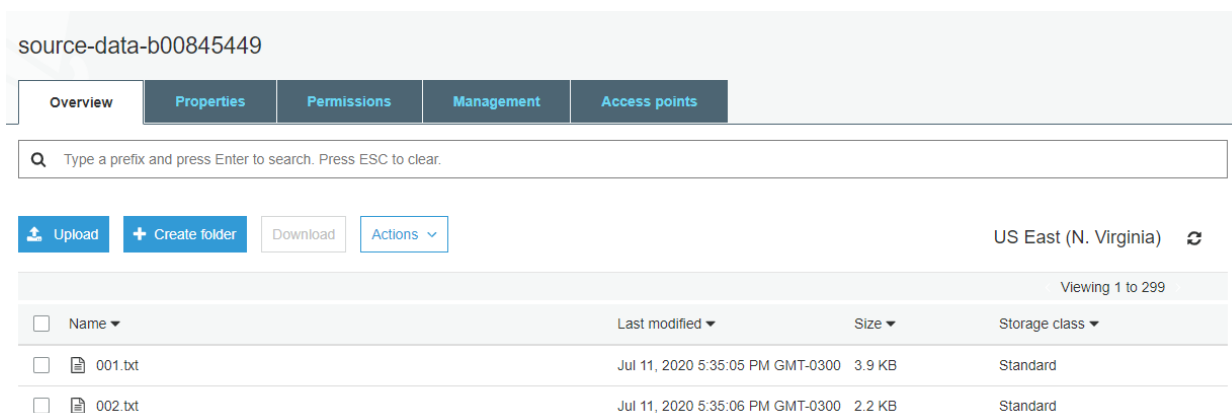


Figure a source-data-b00845449

All training files are uploaded to the bucket named “source-data-b00845449”.

```
class s3Api:
    """ Get name of the buckets """
    def listBuckets(self):
        s3 = boto3.client('s3')
        return s3.list_buckets()

    """ Upload a file """
    def fileUpload(self, bucket_name, source_file_name, file_name):
        s3Resource = boto3.resource('s3')
        s3Resource.Object(bucket_name, source_file_name).upload_file(Filename=file_name)
        print('file uploaded')

    def getFile(self, bucket_name, file_name):
        s3 = boto3.client('s3')
        try:
            return s3.get_object(Bucket=bucket_name, Key=file_name)
        except Exception as e:
            print_(e)
            return False

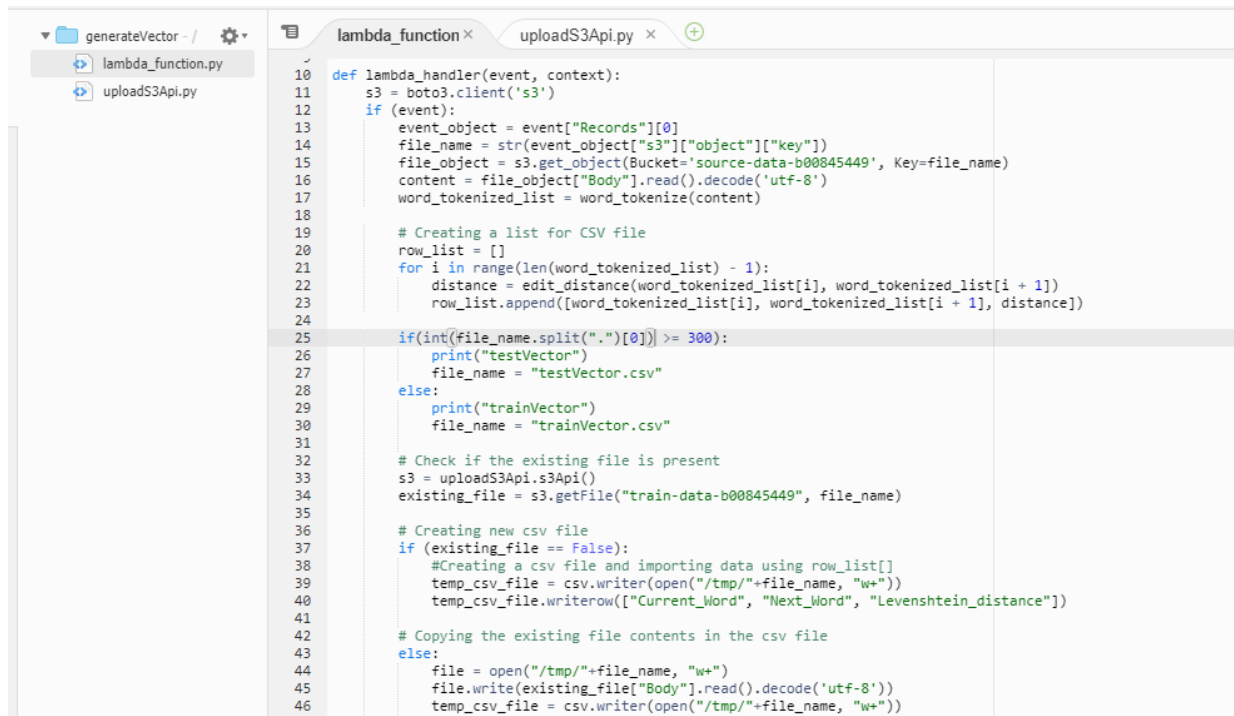
s3 = s3Api()
for i in glob.glob("Dataset\Dataset\Train\*.txt"):
    s3.fileUpload("source-data-b00845449", i.split('\\')[-1], i)
print('Training files Upload Successful')

# for i in glob.glob("Dataset\Dataset\Test\*.txt"):
#     s3.fileUpload("source-data-b00845449", i.split('\\')[-1], i)
# print('Testing files Upload Successful')
```

Figure b bucket upload program

Above is the script which uploads all the training files to the bucket.

- b. Once a file is uploaded, a Lambda function - “generateVector” should extract words from all the files (remove the stop words). Then compute Levenshtein distance between the Current, and Next words.



```

10 def lambda_handler(event, context):
11     s3 = boto3.client('s3')
12     if (event):
13         event_object = event["Records"][0]
14         file_name = str(event_object["s3"]["object"]["key"])
15         file_object = s3.get_object(Bucket='source-data-b00845449', Key=file_name)
16         content = file_object["Body"].read().decode('utf-8')
17         word_tokenized_list = word_tokenize(content)
18
19         # Creating a list for CSV file
20         row_list = []
21         for i in range(len(word_tokenized_list) - 1):
22             distance = edit_distance(word_tokenized_list[i], word_tokenized_list[i + 1])
23             row_list.append([word_tokenized_list[i], word_tokenized_list[i + 1], distance])
24
25         if(int(file_name.split(".")[0]) >= 300):
26             print("testVector")
27             file_name = "testVector.csv"
28         else:
29             print("trainVector")
30             file_name = "trainVector.csv"
31
32         # Check if the existing file is present
33         s3 = uploadS3Api.s3Api()
34         existing_file = s3.getFile("train-data-b00845449", file_name)
35
36         # Creating new csv file
37         if (existing_file == False):
38             #Creating a csv file and importing data using row_list[]
39             temp_csv_file = csv.writer(open("/tmp/"+file_name, "w+"))
40             temp_csv_file.writerow(["Current_Word", "Next_Word", "Levenshtein_distance"])
41
42         # Copying the existing file contents in the csv file
43         else:
44             file = open("/tmp/"+file_name, "w+")
45             file.write(existing_file["Body"].read().decode('utf-8'))
46             temp_csv_file = csv.writer(open("/tmp/"+file_name, "w+"))

```

Figure c generateVector program 1 [2]

```

# Copying the existing file contents in the csv file
else:
    file = open("/tmp/"+file_name, "w+")
    file.write(existing_file["Body"].read().decode('utf-8'))
    temp_csv_file = csv.writer(open("/tmp/"+file_name, "a"))

# Adding new values to CSV
for row in row_list:
    temp_csv_file.writerow([row[0], row[1], row[2]])

# Sending csv file to S3
s3.fileUpload("train-data-b00845449", file_name, "/tmp/"+file_name)

```

Figure d generateVector program 2 [2]

After the files are uploaded, the lambda function fetches the files, and find levenshtein distance between the adjacent words of the file. The csv file is generated and is uploaded to another bucket “train-data-b00845449”. Each time the lambda function checks the filename because the files 000-299 are training files and hence will be appended to trainVector.csv. If the filename is between 300-401, then the levenshtein distance of words will be appended to testVector.csv [2].

The program checks if the file “trainVector.csv” or “testVector.csv” is present in the bucket or not; if present, then it will append the data otherwise it will create a new file with the headings and send it.

```

import boto3
import glob

class s3Api:
    """ Get name of the buckets """
    def listBuckets(self):
        s3 = boto3.client('s3')
        return s3.list_buckets()

    """ Upload a file """
    def fileUpload(self, bucket_name, source_file_name, file_name):
        s3Resource = boto3.resource('s3')
        s3Resource.Object(bucket_name, source_file_name).upload_file(Filename=file_name)
        print('file uploaded')

    def getFile(self, bucket_name, file_name):
        s3 = boto3.client('s3')
        try:
            return s3.get_object(Bucket=bucket_name, Key=file_name)
        except Exception as e:
            print(e)
            return False

```

Figure e generateVector program 2

Above is the script used for the uploading and fetching the files from the AWS S3. This file is being called in the main lambda function.

There are older events to load. Load more.		
2020-07-11T17:35:11.071-03:00	START	RequestId: 4152f1e2-83f0-4d7b-85ed-f21b6c59f38e Version: \$LATEST
2020-07-11T17:35:12.029-03:00	trainVector	
2020-07-11T17:35:12.498-03:00	file uploaded	
2020-07-11T17:35:12.529-03:00	END	RequestId: 4152f1e2-83f0-4d7b-85ed-f21b6c59f38e
2020-07-11T17:35:12.529-03:00	REPORT	RequestId: 4152f1e2-83f0-4d7b-85ed-f21b6c59f38e Duration: 1457.45 ms Billed Duration: 1500 ms Memory Si
2020-07-11T17:35:12.699-03:00	START	RequestId: dfeedccc-8e4b-481a-840e-3a27415d0808 Version: \$LATEST
2020-07-11T17:35:12.890-03:00	trainVector	
2020-07-11T17:35:13.696-03:00	file uploaded	
2020-07-11T17:35:13.710-03:00	END	RequestId: dfeedccc-8e4b-481a-840e-3a27415d0808
2020-07-11T17:35:13.710-03:00	REPORT	RequestId: dfeedccc-8e4b-481a-840e-3a27415d0808 Duration: 1007.76 ms Billed Duration: 1100 ms Memory Si
2020-07-11T17:35:14.191-03:00	START	RequestId: c0061106-b3f8-468c-9c54-6d25e2980a74 Version: \$LATEST
2020-07-11T17:35:14.429-03:00	trainVector	
2020-07-11T17:35:14.736-03:00	file uploaded	

Figure f cloudwatch logs for generateVector

Above is the snapshot of cloudwatch logs which shows the log of files uploaded and also the file (“trainVector or testVector”) where it gets appended.

- c. This file (“trainVector.csv”) is saved in a new bucket TrainDataB00xxxxxxx.

train-data-b00845449

Overview

Properties

Permissions

Management

Access points

Q

Type a prefix and press Enter to search. Press ESC to clear.

Upload

+ Create folder

Download

Actions

US East (N. Virginia)

Viewing 1 to 1

<div><input type="checkbox"/></div> <div>Name</div>	Last modified	Size	Storage class
<div><input type="checkbox"/></div> <div><div><div></div></div>trainVector.csv</div>	Jul 11, 2020 5:36:34 PM GMT-0300	408.4 KB	Standard

Figure g train-data-b00845449

As the above screenshot shows that the trainVector.csv file is getting generated from the lambda function “generateVector” and the lavenshtein distance of all the words of the training data is then appended to the same file.

UK	net	3
net	users	4
users	leading	7
leading	TV	7
TV	download	9
download	British	9
British	TV	7
TV	viewers	7
viewers	lead	6
lead	the	4
the	trend	3
trend	of	5
of	illegally	9
illegally	download	10
download	US	11
US	shows	5
shows	from	4
from	the	4
the	net	3
net	,	3
,	according	9
according	to	8

Figure h sample of csv file

The above figure shows the csv file sample. The columns are “FirstWord”, “NextWord”, “Lavenshtein distance” respectively.

Note: As the AWS educate account does not provide permission to complete a further task which requires AWS SageMaker, hence no further work could be done.

Testing for generateVector lambda function [3]

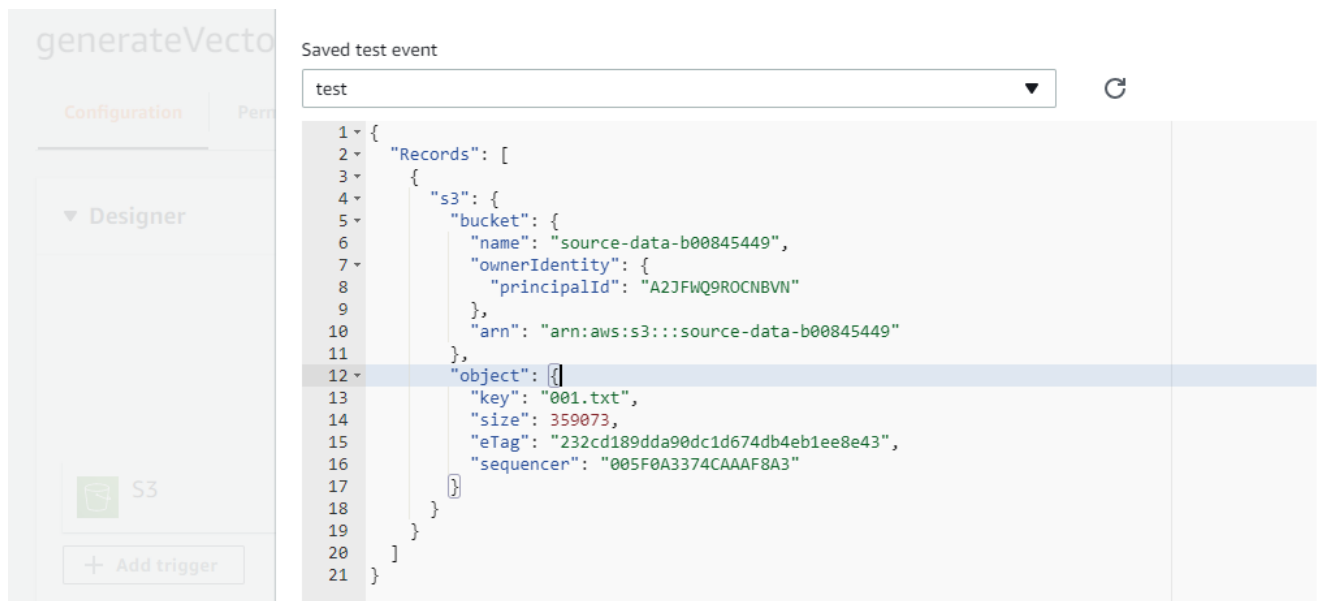


Figure i testing 1 for generateVector [3]

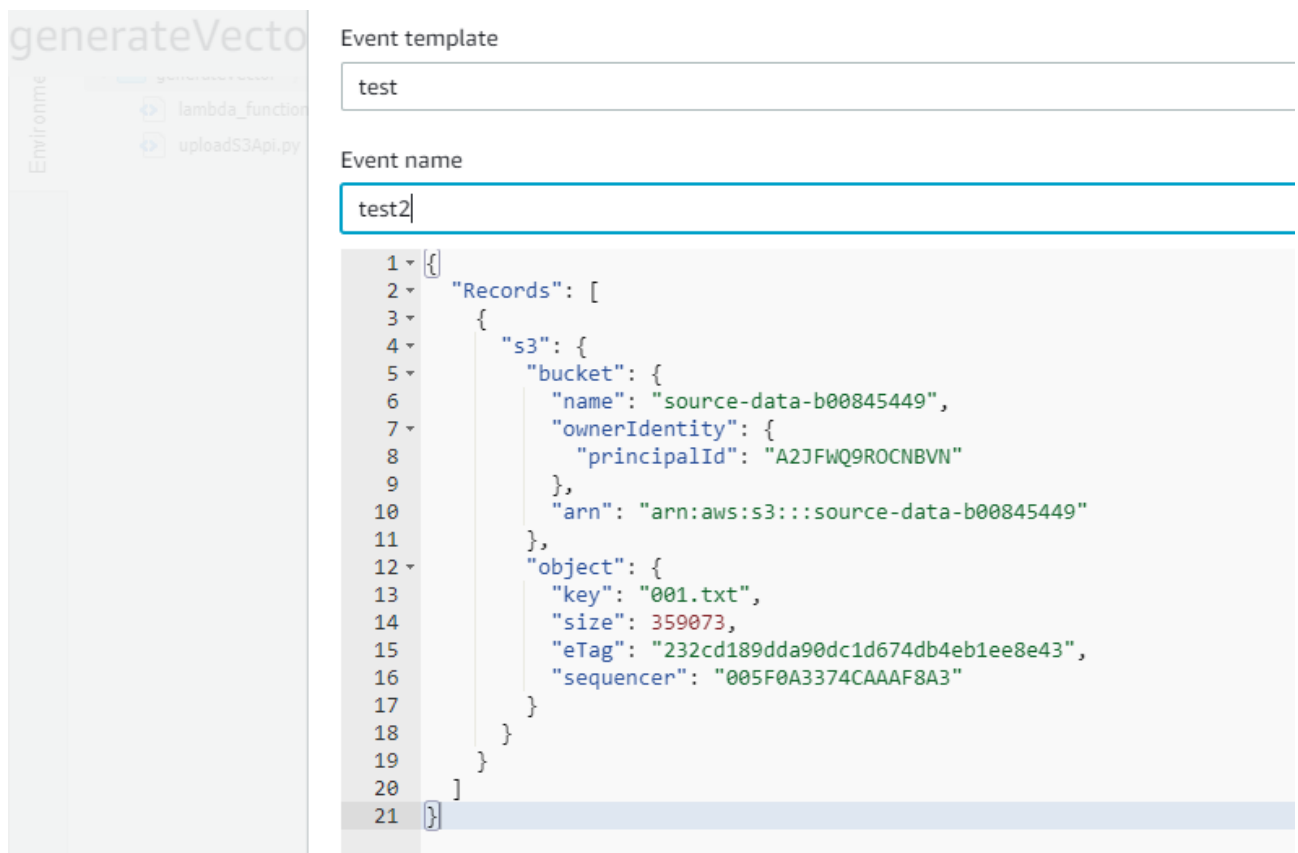


Figure j testing 2 for generateVector [3]

Part C. Build an event-driven serverless application using AWS Comprehend.

- Create your 1st S3 bucket `TwitterDataB00xxxxxx` and upload the files given in the tweets folder. You need to write a script or use the SDK to upload the files on the bucket.

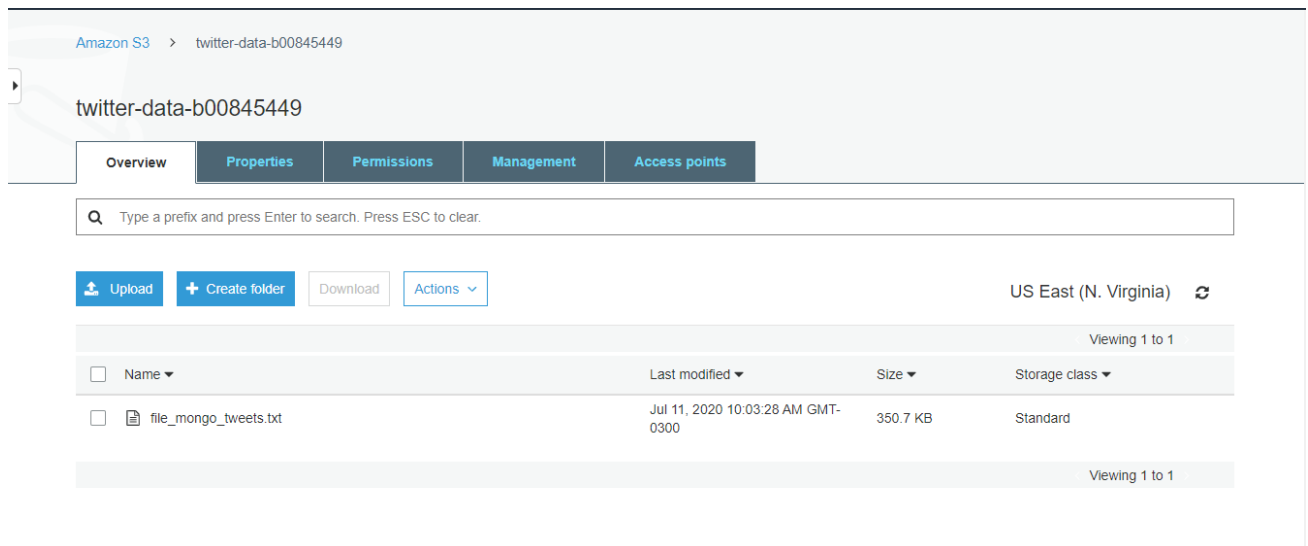


Figure k twitter-data-b00845449

The above figure shows the tweets file uploaded to the bucket named “twitter-data-b00845449”.

```
import boto3

class s3Api:
    """ Get name of the buckets """
    def listBuckets(self):
        s3 = boto3.client('s3')
        return s3.list_buckets()

    """ Upload a file """
    def fileUpload(self, bucket_name, source_file_name, file_name):
        s3Resource = boto3.resource('s3')
        s3Resource.Object(bucket_name, source_file_name).upload_file(Filename=file_name)
        print('file uploaded')

s3 = s3Api()
s3.fileUpload("twitter-data-b00845449", "file_mongo_tweets.txt", "file_mongo_tweets.txt")
print('Upload Successful')
```

Figure l uploading tweets to the bucket

Above is the snapshot of the script which is used to upload the file to the bucket.

- b. To perform any pre or post-processing of the files, you can write Lambda functions.

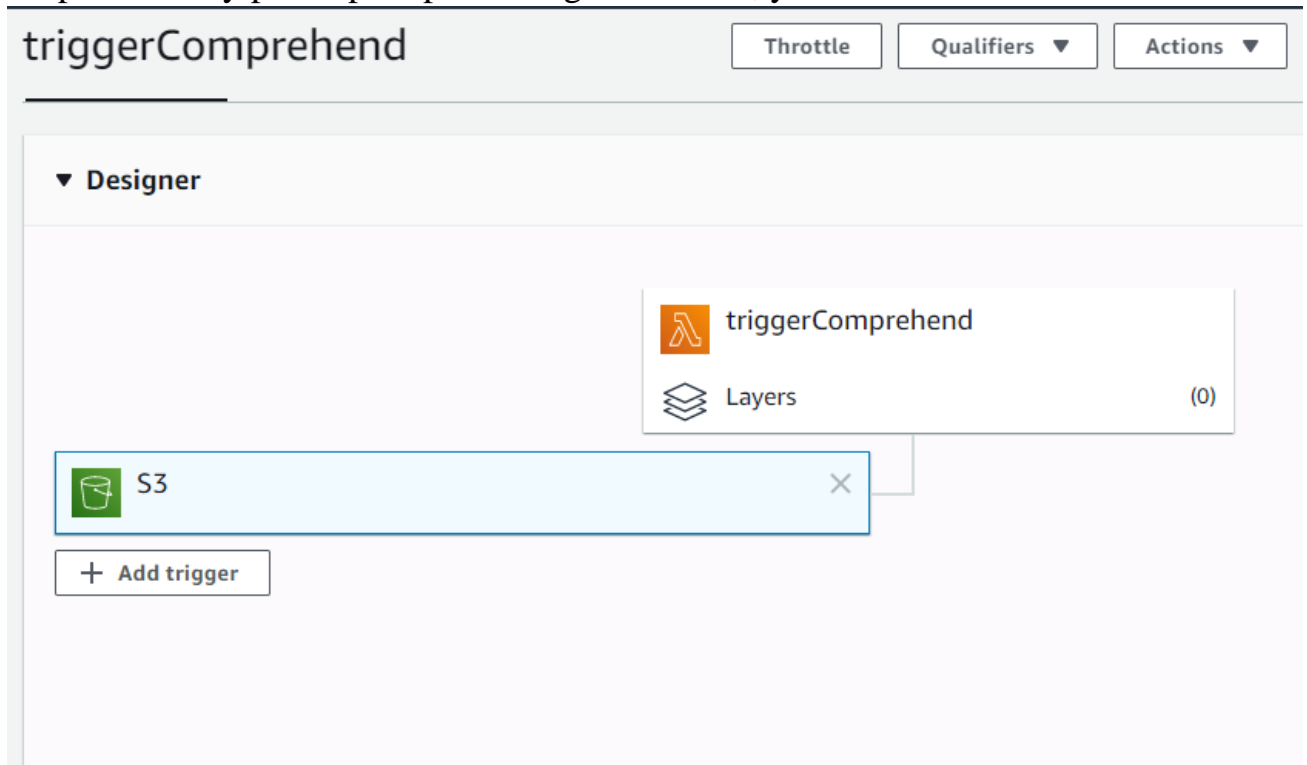


Figure m triggerComprehend lambda trigger

The lambda function named “triggerComprehend” is created to do preprocessing tasks. The lambda function is triggered on PUT operation of S3 bucket named “twitter-data-b00845449”.

Create policy

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor

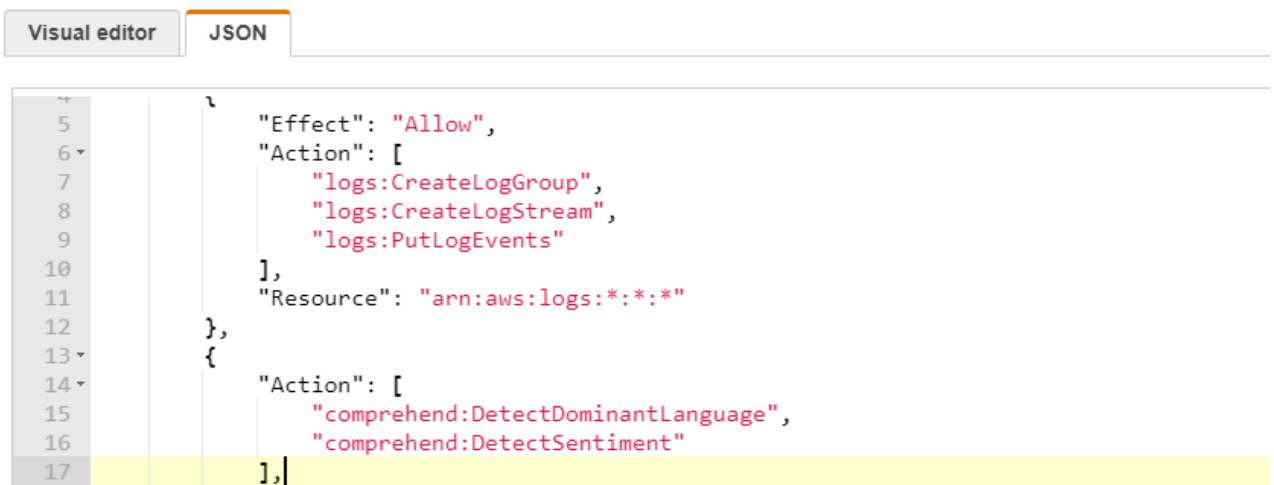


Figure n create policy

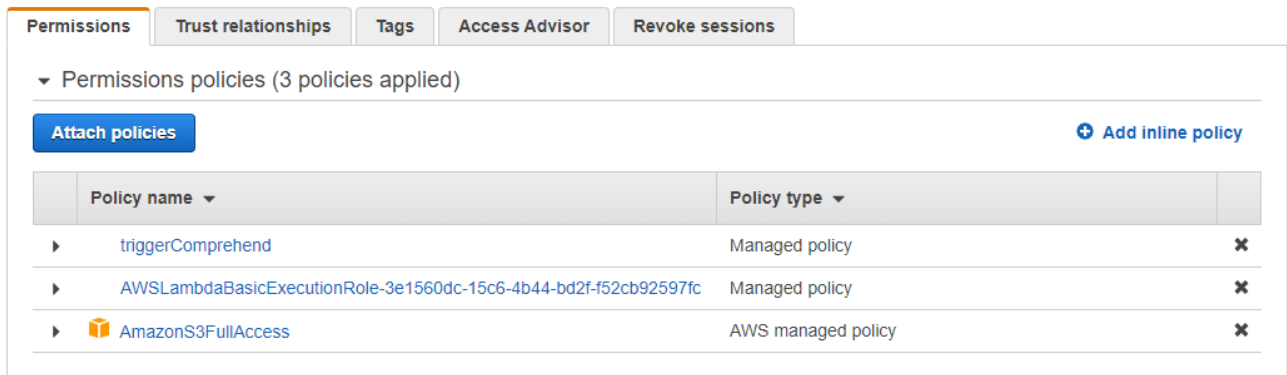


Figure o permissions and role

The policy for the Lambda function is created and attached to the role. A policy to access Comprehend is also attached.

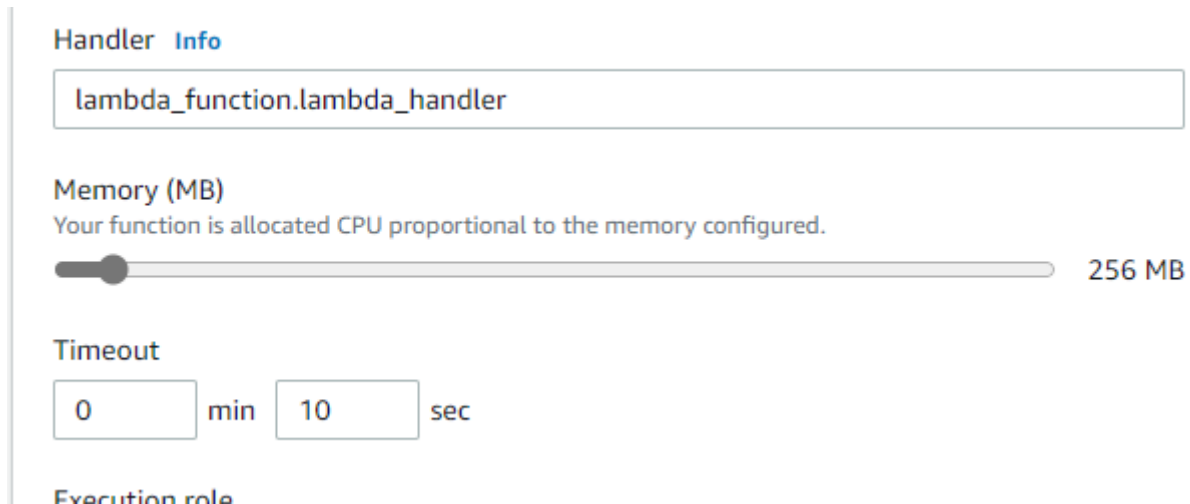


Figure p triggerComprehend settings

As the processing will require more time and memory, the memory is increased to 256 MB and time is increased to 10 seconds.

```
1 import boto3
2
3 def lambda_handler(event, context):
4     s3 = boto3.client('s3')
5     if (event):
6         event_object = event["Records"][0]
7         file_name = str(event_object["s3"]["object"]["key"])
8         file_object = s3.get_object(Bucket='twitter-data-b00845449', Key=file_name)
9         content = file_object["Body"].read()
10
11         """As all the tweets paragraphs are separated by 2 lines,
12         I split the paragraphs by '/n/n' and find each tweet sentiments"""
13         content = content.replace(b'\n\n', b'\n\n\n')
14         content = content.split(b'\n\n\n')
15
16         client2 = boto3.client('comprehend')
17         for i in content:
18             sentiment = client2.detect_sentiment(Text=i.decode('utf-8'), LanguageCode='en')['Sentiment']
19             print([i, sentiment])
```

Figure q triggerComprehend function program [1]

Above is the script of the lambda function which fetches the content from each bucket object and detects sentiment. I analyzed that each tweet is separated by a single line, hence I created a list of tweets by splitting using `'\\n\\n'`. AWS Comprehend is used to find the sentiment of each tweet [1].

- c. Once all the files are uploaded on the bucket, AWS Comprehend is used to perform sentiment analysis of tweets.

		There are older events to load. Load more.
▼	2020-07-11T11:28:13.196-03:00	[b'\nIt\xe2\x80\x99s Red Party Day! Why celebrate Canada Day anywhere else than at Toronto\xe2\x80\x99s BIGGEST pool party. ', 'POSITIVE']
▼	2020-07-11T11:28:13.262-03:00	[b'Free before 3p\xe2\x80\xa6\nWhere in Canada would you like to travel to the most? #LongWeekendVibes \nSoyez un bon voisin. Respectez les autres lorsque vous c\xe3\xa9l\xe9brez la f\xeate du Canada!', 'POSITIVE']
▼	2020-07-11T11:28:13.322-03:00	[b'\nProgram director with Renfrew\xe2\x80\x99s Parks and Recreations, Jo-anne Caldwell, says the town will host a spectacular pyro\xe2\x80\xa6\nQUINTE WEST CANADA DAY ', 'NEUTRAL']
▼	2020-07-11T11:28:13.384-03:00	[b'Get ready for a full day of activities including a vendor village, free cake, outdoor bi\xe2\x80\xa6\nBe a good neighbour. Respect others while celebrating Canada Day!', 'POSITIVE']
▼	2020-07-11T11:28:13.486-03:00	[b' #ottcity \nRT @reneekendall198: #top #day #music #need #HelpEach1 #hi #summer #Donations #health #Family #vegan #travel #Donation #sale \nUSA\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c location : Regional District of North Okanagan, British Columbia, Canada Local Date/Time : 2019-06-30 07:00:02.454609-07:00\nRT @EllenJBotelhol: Retweet the hell out of this, please!!! \nOn June 30, 1812, a proclamation gave American citizens fourteen days to leave Upper Canada. A great place to read\xe2\x80\xa6\nRT @OGMurphy1: Since the Brexit vote, the EU has agreed trade deals with;', 'NEUTRAL']
▼	2020-07-11T11:28:13.584-03:00	[b'1. Canada\n2. Japan\n3. Brazil\n4. Argentina\n5. Paraguay\n6. Uruguay\xe2\x80\xa6\n@reneekendall198: #top #day #music #need #HelpEach1 #hi #summer #Donations #health #family #vegan #travel #Donation #sale \nUSA\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c\xe2\x80\x9c murdered Syrians &#amp; is walking the streets of Canada. The only th\xe2\x80\xa6\nOpen to US and Canada -- Enter to win 1 of 5 huge prize packs for mom and baby or a \$50 gift card in\xe2\x80\xa6\n@politicususa: "A self-declared \xe2\x80\x9cxccaravan\xe2\x80\x9d of Americans bused across the Canada-U.S. border on Saturday, seeking affordable prices for i\xe2\x80\xa6\nOur #poetry and #fiction close TONIGHT! Submit now for your chance to win publication in our fall issue \xe2\x80\x9cx plus the\xe2\x80\x96\n@theworldindex: Best Countries for Citizenship, in 2019:', 'NEUTRAL']

Figure 1 triggerComprehend cloudwatch logs

After the lambda function detects sentiments of each tweet, we can see the result on cloudwatch logs. Each log shows the tweet and the sentiment (Positive, Negative, Neutral).

Testing for triggerComprehend lambda function [3]

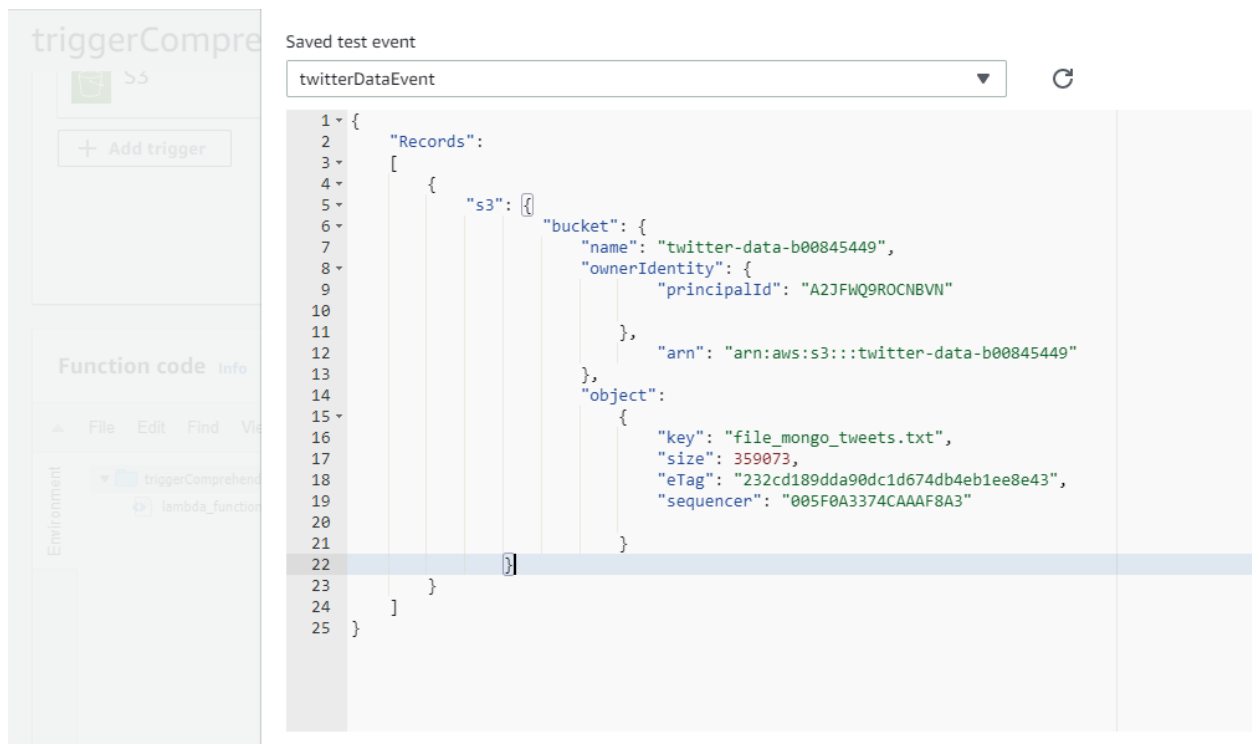


Figure 5 testing for TriggerComprehend [3]

References

- [1] "Using AWS Lambda and Amazon Comprehend for sentiment analysis | Amazon Web Services", *Amazon Web Services*, 2020. [Online]. Available: <https://aws.amazon.com/blogs/compute/using-aws-lambda-and-amazon-comprehend-for-sentiment-analysis>. [Accessed: 11- Jul- 2020]
- [2] "Metrics", *Nltk.org*, 2020. [Online]. Available: <http://www.nltk.org/howto/metrics.html>. [Accessed: 11- Jul- 2020]
- [3] "How to unit test an AWS Lambda", *Medium*, 2020. [Online]. Available: <https://medium.com/@pekelnny/how-to-unit-test-an-aws-lambda-524069d4fe06>. [Accessed: 11- Jul- 2020]
- [4] *Docs.aws.amazon.com*, 2020. [Online]. Available: <https://docs.aws.amazon.com/sagemaker/>. [Accessed: 11- Jul- 2020]
- [5] *Docs.aws.amazon.com*, 2020. [Online]. Available: <https://docs.aws.amazon.com/comprehend/>. [Accessed: 11- Jul- 2020]