EVENT-DRIVEN SERVERLESS APPLICATION USING AWS COMPREHEND ALONG WITH AWS S3 AND AWS LAMBDA

CSCI 5410 – Assignment 4 – Part C

Dhrumil Amish Shah (B00857606)
dh416386@dal.ca

Build an event-driven serverless application using AWS Comprehend. In this part of the assignment, you need to use the S3 bucket, Lambda Functions, and AWS Comprehend. [B00xxxxxx = your B00 number] used in bucket naming.

Figure 1 displays the architecture of the serverless application built using Node.js and Express.js backend frameworks along with S3 (Simple Storage Service), Lambda, and Comprehend AWS services.

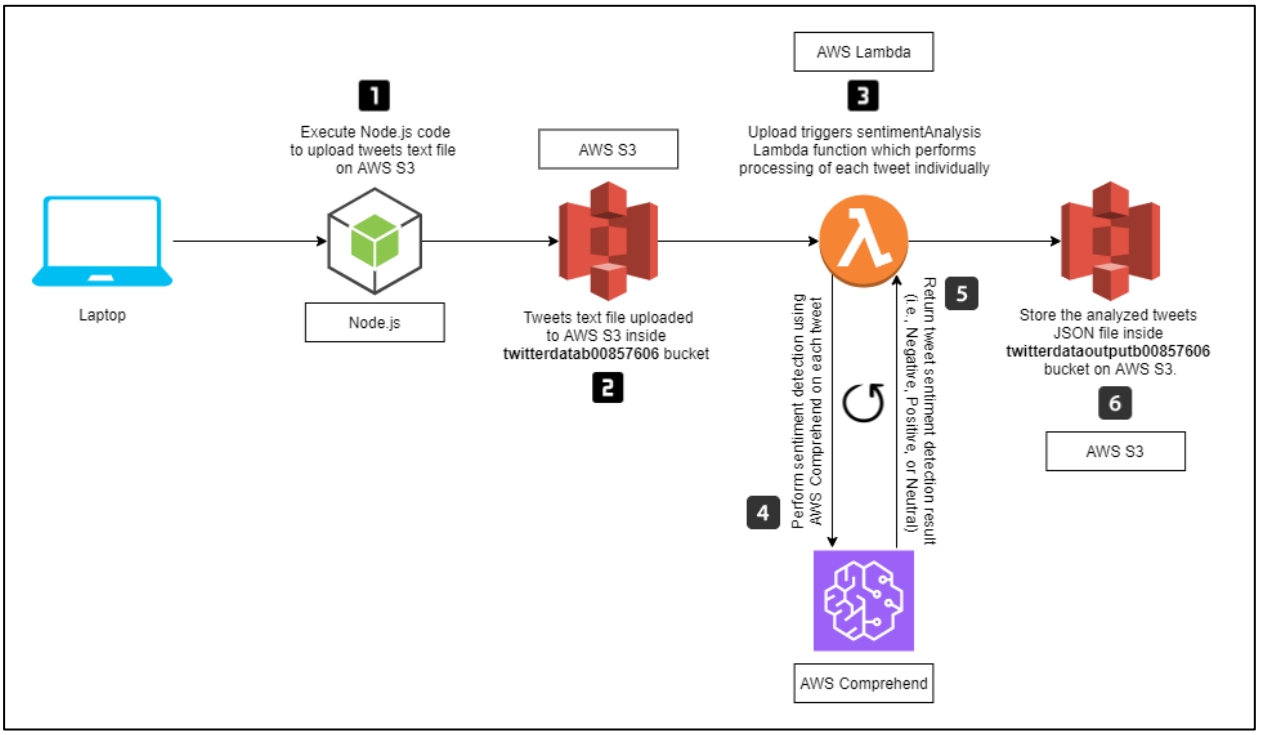

Figure 1 - Serverless application architecture for tweets sentiment analysis using AWS Comprehend [1]

Steps performed for creating the serverless application for tweets sentiment analysis using Node.js and Express.js backend frameworks along with AWS S3, AWS Lambda, and AWS Comprehend are as below:

Step 1: Run the code written in Node.js and Express.js to upload the tweets text file to the **twitterdatab00857606** S3 bucket.

Step 2: Tweets text file uploaded to the **twitterdatab00857606** S3 bucket.

Step 3: File uploaded to the **twitterdatab00857606** S3 bucket triggers the **sentimentAnalysis** Lambda function. This function gets the tweet file uploaded to the **twitterdatab00857606** S3 bucket, separates each tweet using REGEX and process it individually.

Step 4: For each tweet, perform sentiment detection using Comprehend service API.

Step 5: Result returned from Step 4 is stored in a JSON object. Each JSON object consists of the original tweet, sentiment (i.e., POSITIVE, NEGATIVE or NEUTRAL) and sentiment score.

Step 6: Store the final JSON array with sentiment JSON objects inside a file with the ".json" extension. Upload the ".json" file to the **twitterdataoutputb00857606** S3 bucket.

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

Figure 2 displays the screenshot of the S3 bucket creation page filled with details to create an S3 bucket with bucket name **twitterdatab00857606**. This bucket contains two tags namely **organization** with value **CSCI5410-A4-PartC** and **developer** with value **Dhrumil-Amish-Shah**.

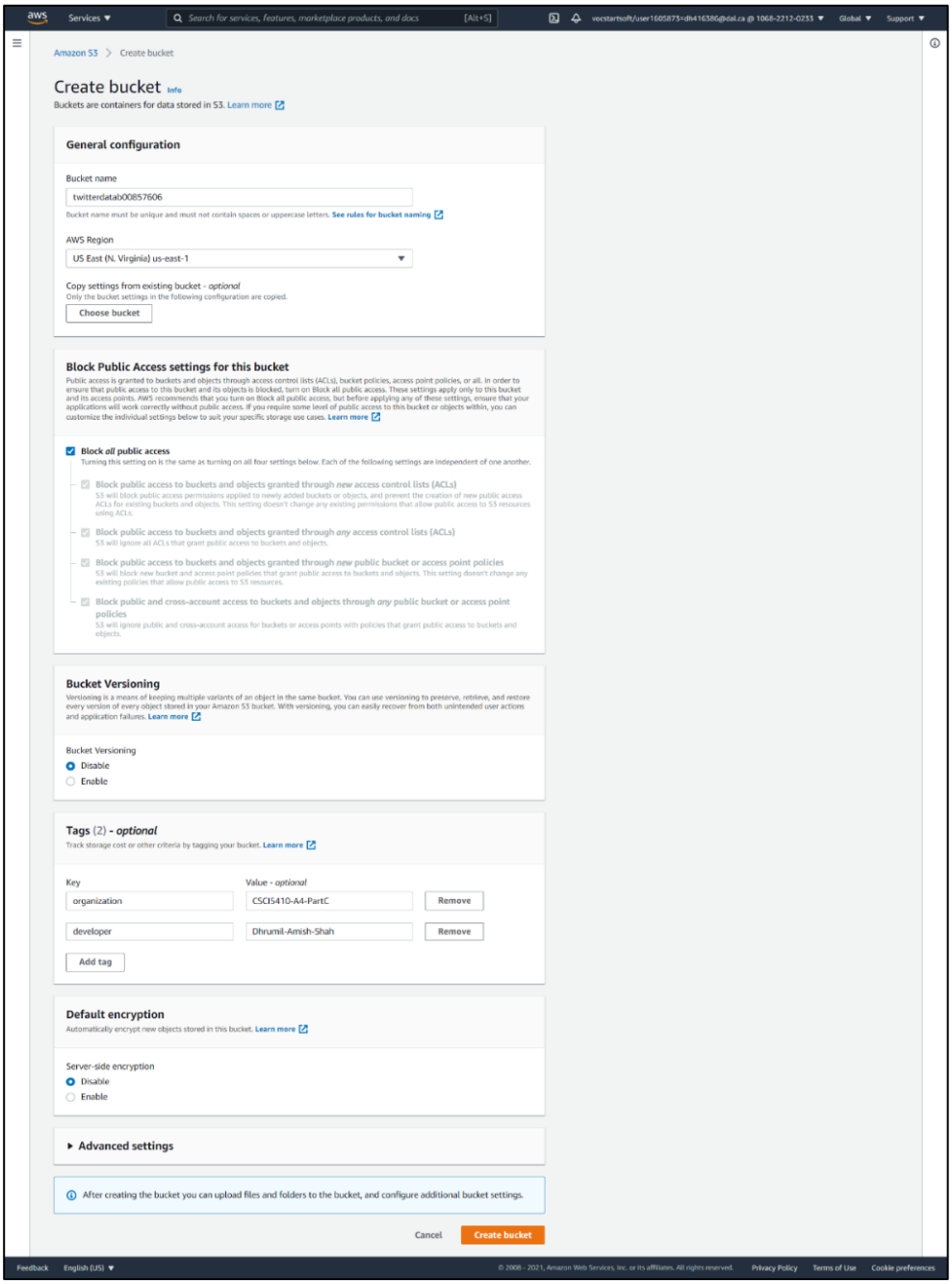

Figure 2 - S3 bucket creation page to create twitterdatab00857606 bucket [2]

Figure 3 displays the screenshot of created S3 bucket **twitterdatab00857606** in the US East (N. Virginia) us-east-1 region.

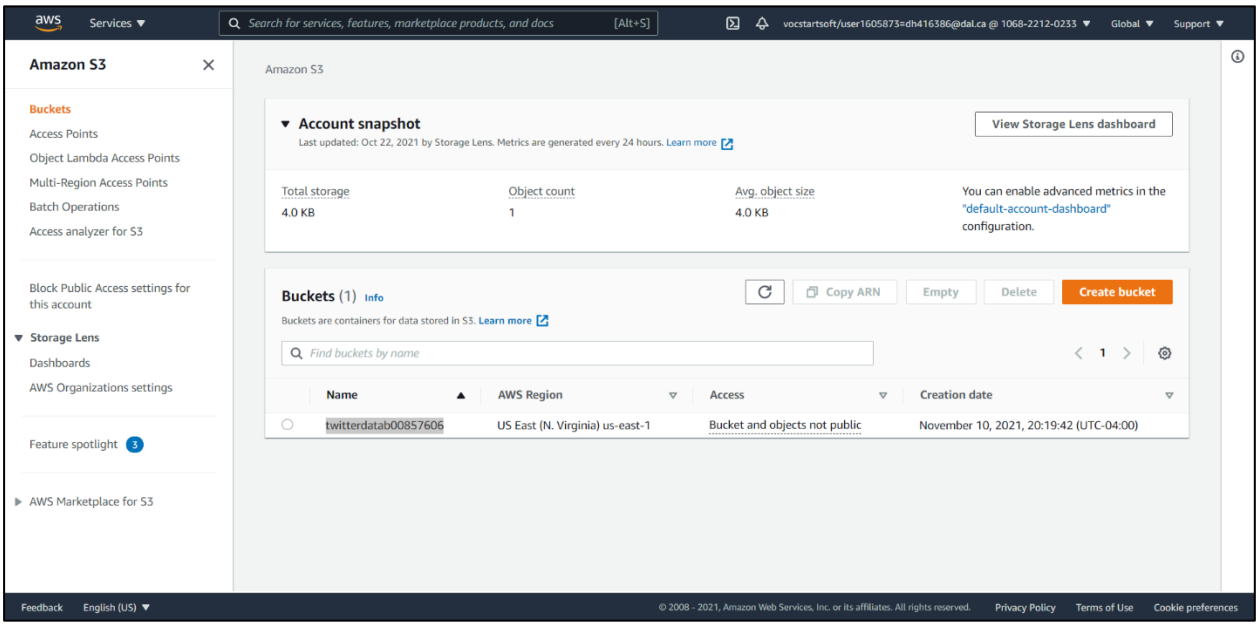

Figure 3 - Created S3 bucket twitterdatab00857606 in us-east-1 region [2]

Figure 4 displays the screenshot of the initial state of the **twitterdatab00857606** bucket. Initially, there are no files in the bucket.

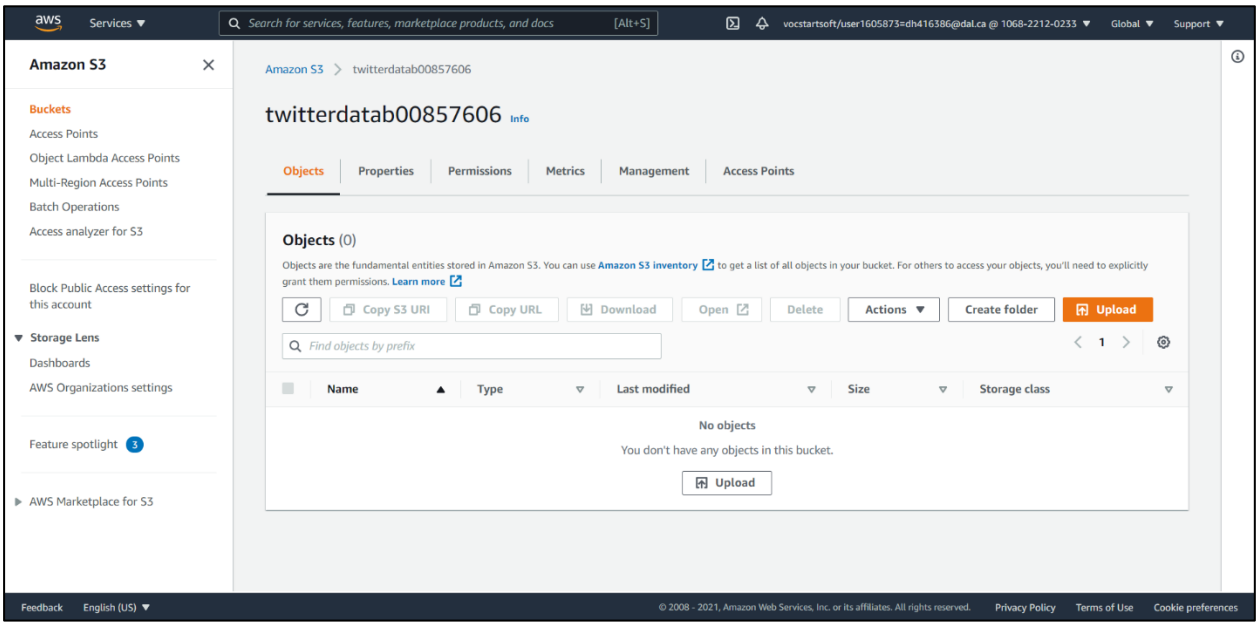

Figure 4 - Empty twitterdatab00857606 bucket [2]

Figure 5 displays the screenshot of the execution of the application. Successful start of the application displays **Server is up at PORT: 3000** message. The application can be accessed at **http://localhost:3000** address. Also, it exposes one endpoint (i.e., http://localhost:3000/upload).

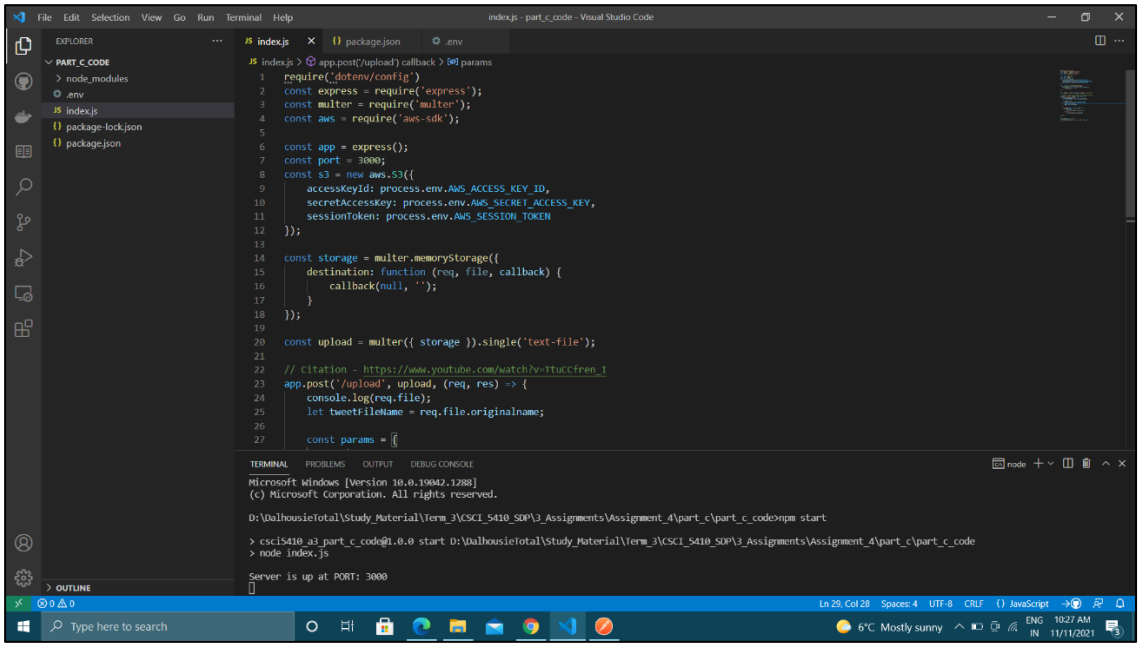

Figure 5 - Execution of Node.js and Express.js application running at PORT – 3000

Figure 6 displays the screenshot of the successful execution of the **http://localhost:3000/upload** API endpoint in the Postman application. Successful execution of this endpoint returns a JSON object with entries namely ETag, Location, key, Key, and Bucket.

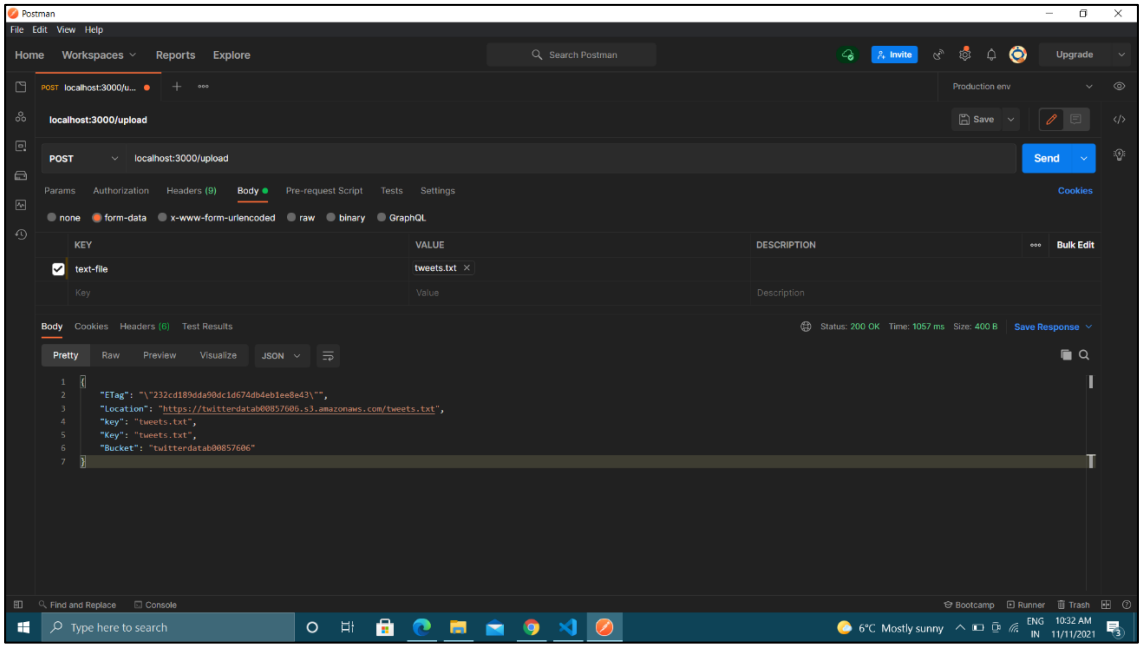

Figure 6 - Successful execution of endpoint http://localhost:3000/upload [3]

Figure 7 displays the screenshot of log messages printed on the console when **http://localhost:3000/upload** API is executed. They display the content of **req.file** object which includes fieldname, originalname, encoding, mimetype, and buffer. The buffer contains the actual file that is to be uploaded.

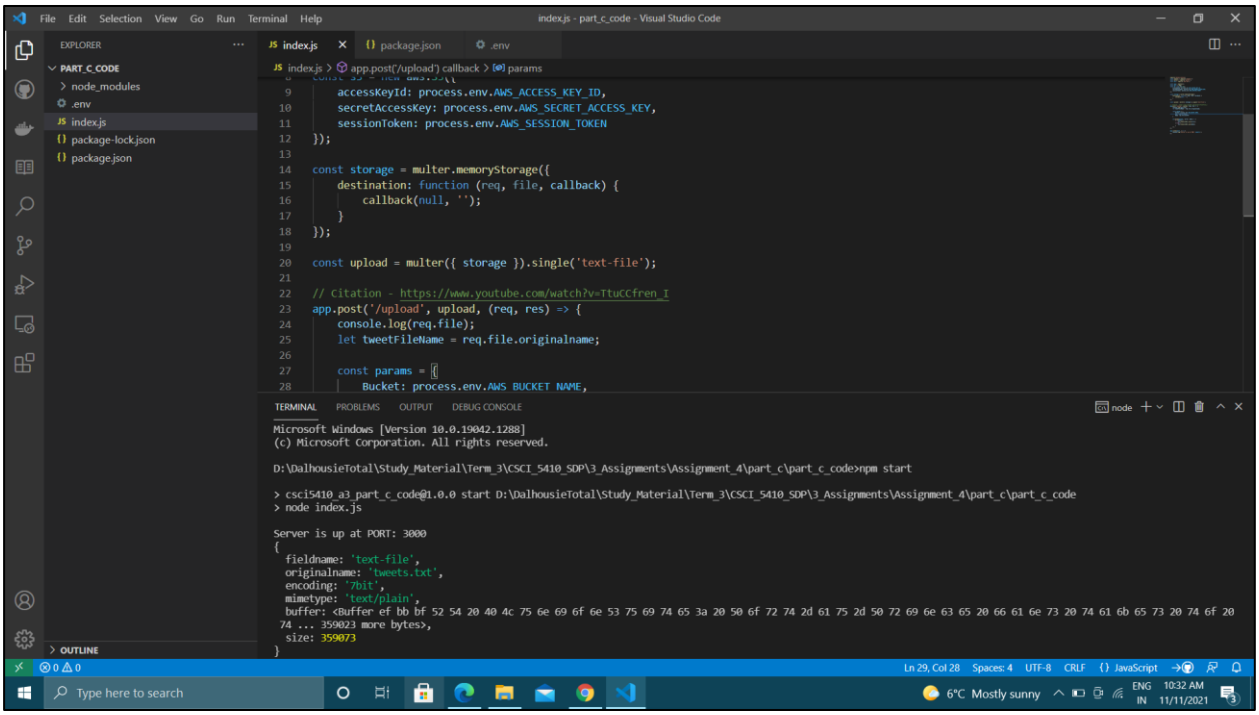

Figure 7 - Log messages printed on the console

Figure 8 displays the screenshot of the tweets.txt file uploaded to the twitterdatab00857606 bucket. All tweets are inside this file.

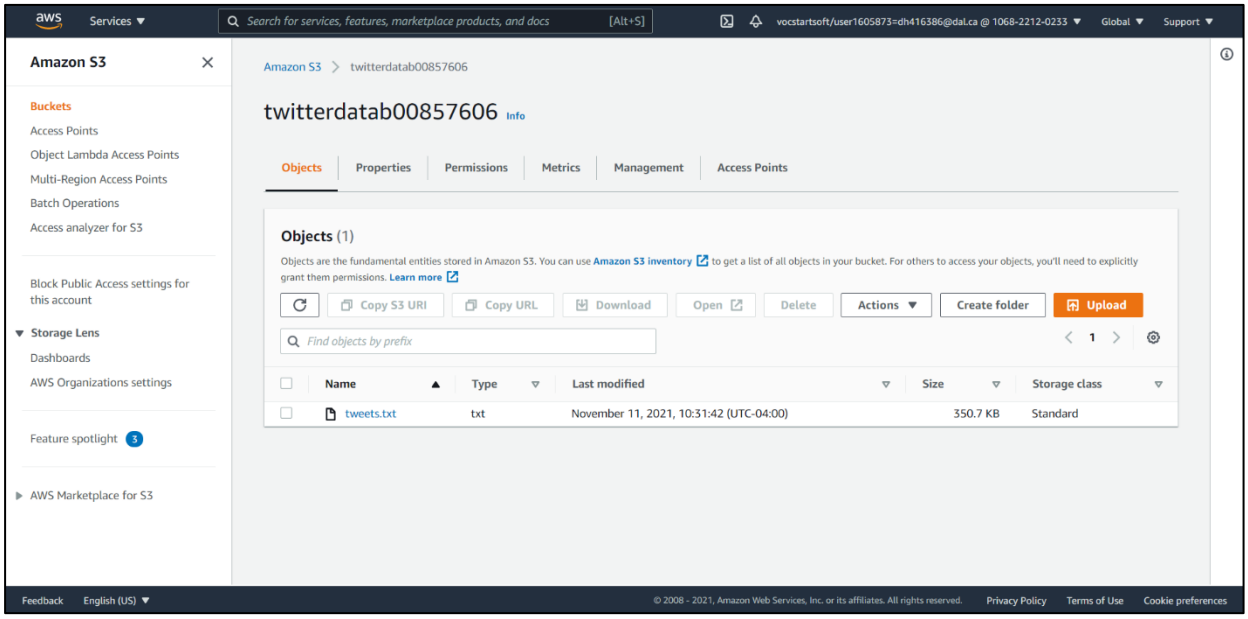

Figure 8 - Bucket twitterdatab00857606 with tweets.txt file [2]

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

Figure 9 displays the screenshot of the Create role – Review page. The primary responsibility of the role **AWSLamdaS3CloudWatchComprehendFullAccess** is to allow AWS Lambda full access to AWS S3, AWS CloudWatch, and AWS Comprehend services.

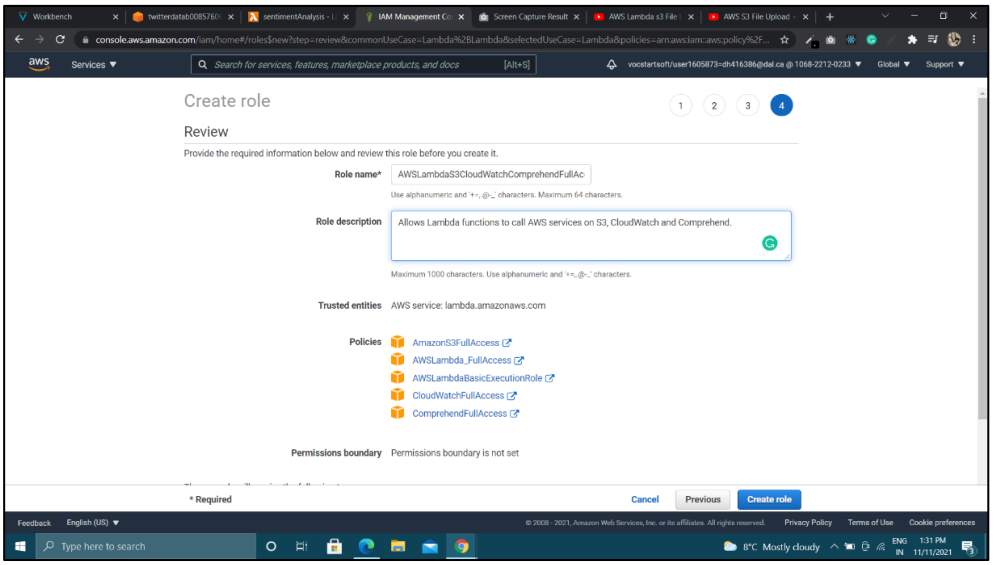

Figure 9 - AWSLamdaS3CloudWatchComprehendFullAccess Create role – Review page [4]

Figure 10 displays the screenshot of the Summary page of the newly created role AWSLamdaS3CloudWatchComprehendFullAccess.

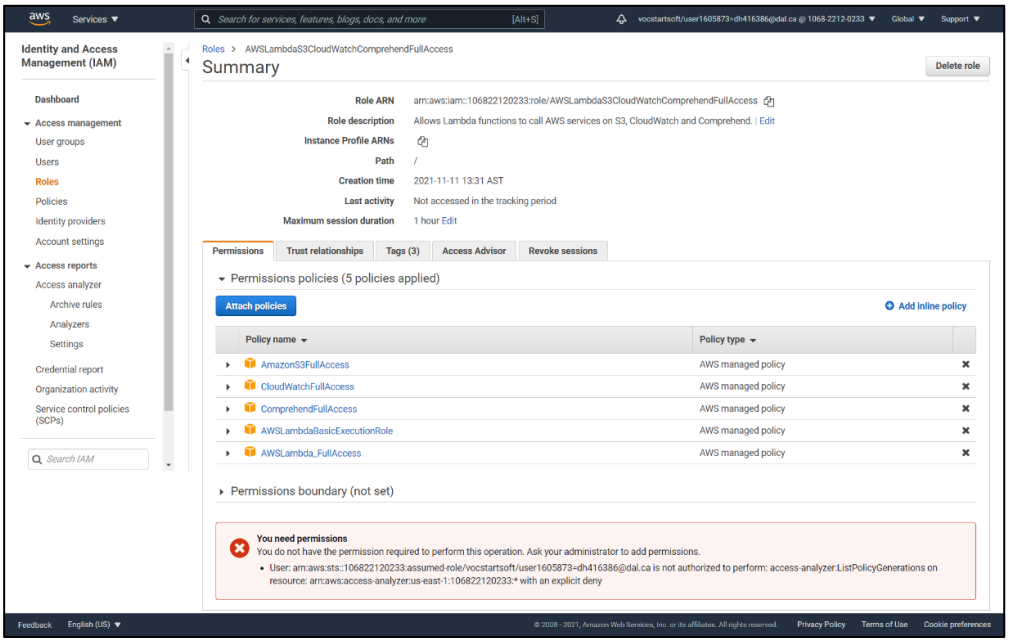

Figure 10 - AWSLamdaS3CloudWatchComprehendFullAccess role summary page [4]

Figure 11 displays the screenshot of the Lambda function creation page filled with details to create a Lambda function with the function name sentimentAnalysis. The Lambda function consists of AWSLamdaS3CloudWatchComprehendFullAccess existing role to have full access to AWS S3, AWS CloudWatch and AWS Comprehend services. Also, the runtime environment select is Node.js.

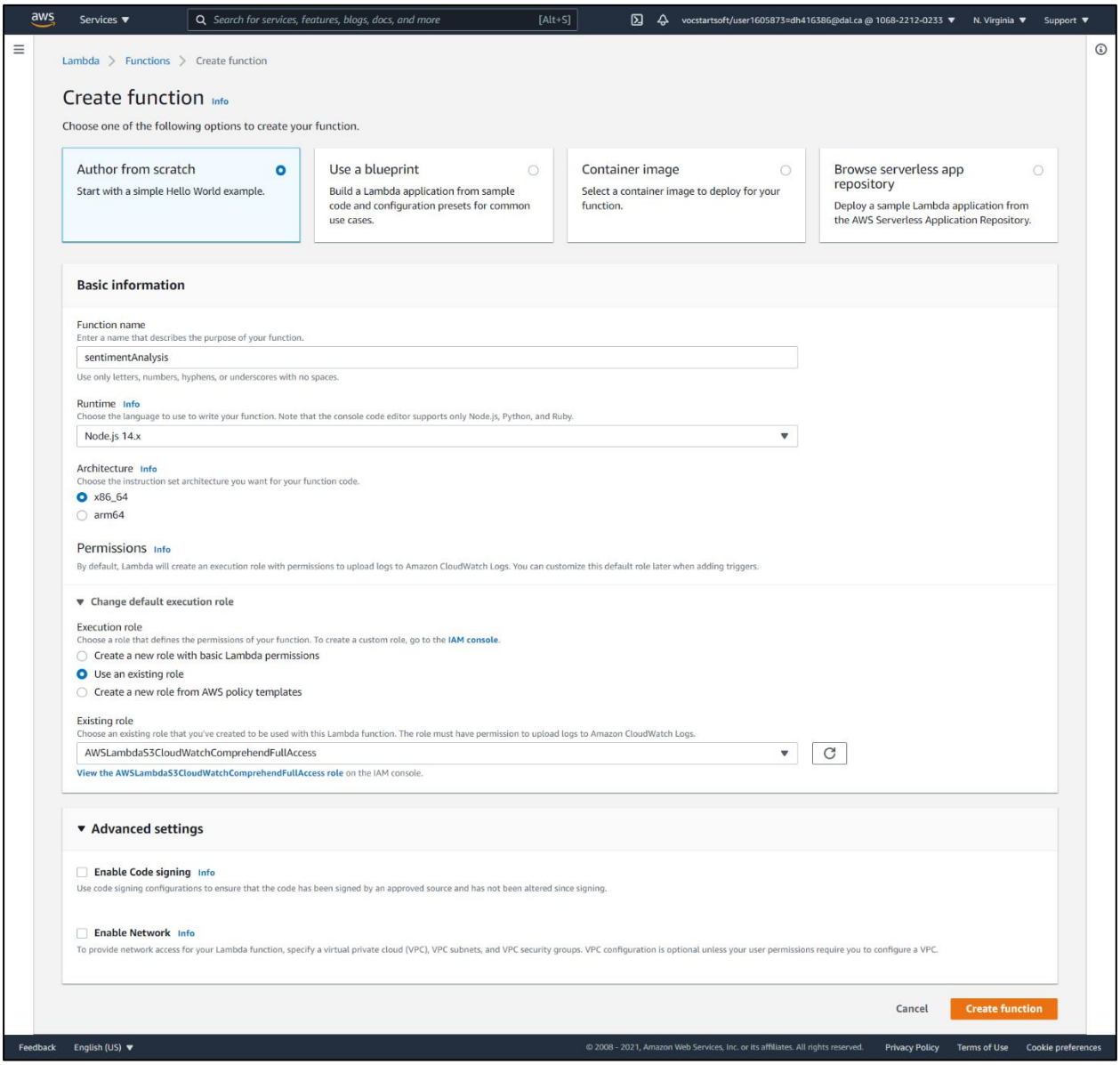

Figure 11 - sentimentAnalysis Lambda function creation page [5]

Figure 12 displays the screenshot of the **Add trigger** creation page. Trigger configuration consists of bucket **twitterdatab00857606** linked to **sentimentAnalysis** Lambda function. This event is triggered when an object is uploaded to bucket **twitterdatab00857606.**

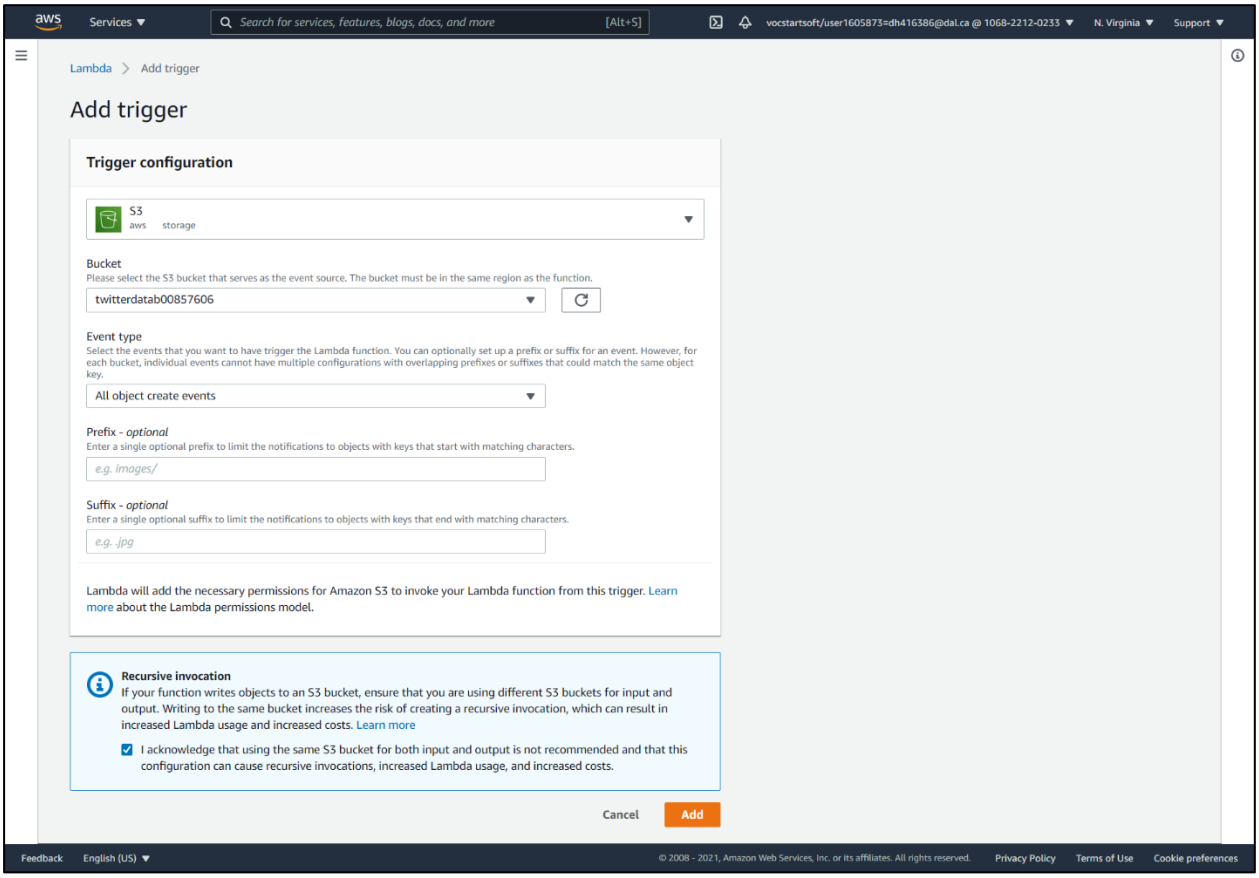

Figure 12 - Add trigger creation page for Lambda function linked to bucket twitterdatab00857606 [5]

Figure 13 displays the screenshot of the sentimentAnalysis function page with the S3 trigger configured to listen to object uploads on bucket **twitterdatab00857606.** An object upload to bucket **twitterdatab00857606** triggers the semanticAnalysis Lambda function.

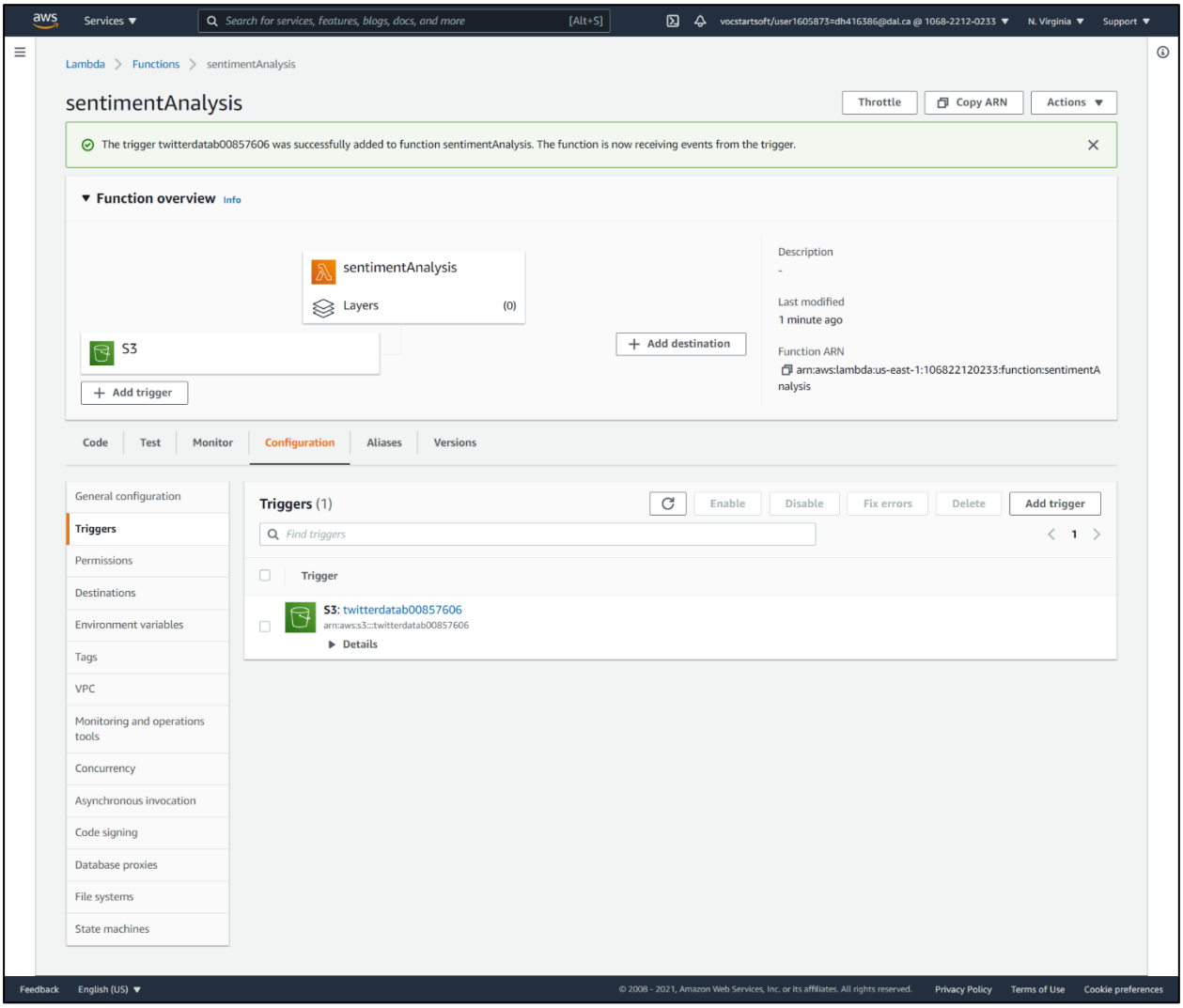

Figure 13 - sentimentAnalysis function page with S3 trigger configured on bucket twitterdatab00857606 [5]

Figure 14 displays the sentimentAnalysis function page with the Lambda function written in the Node.js framework.

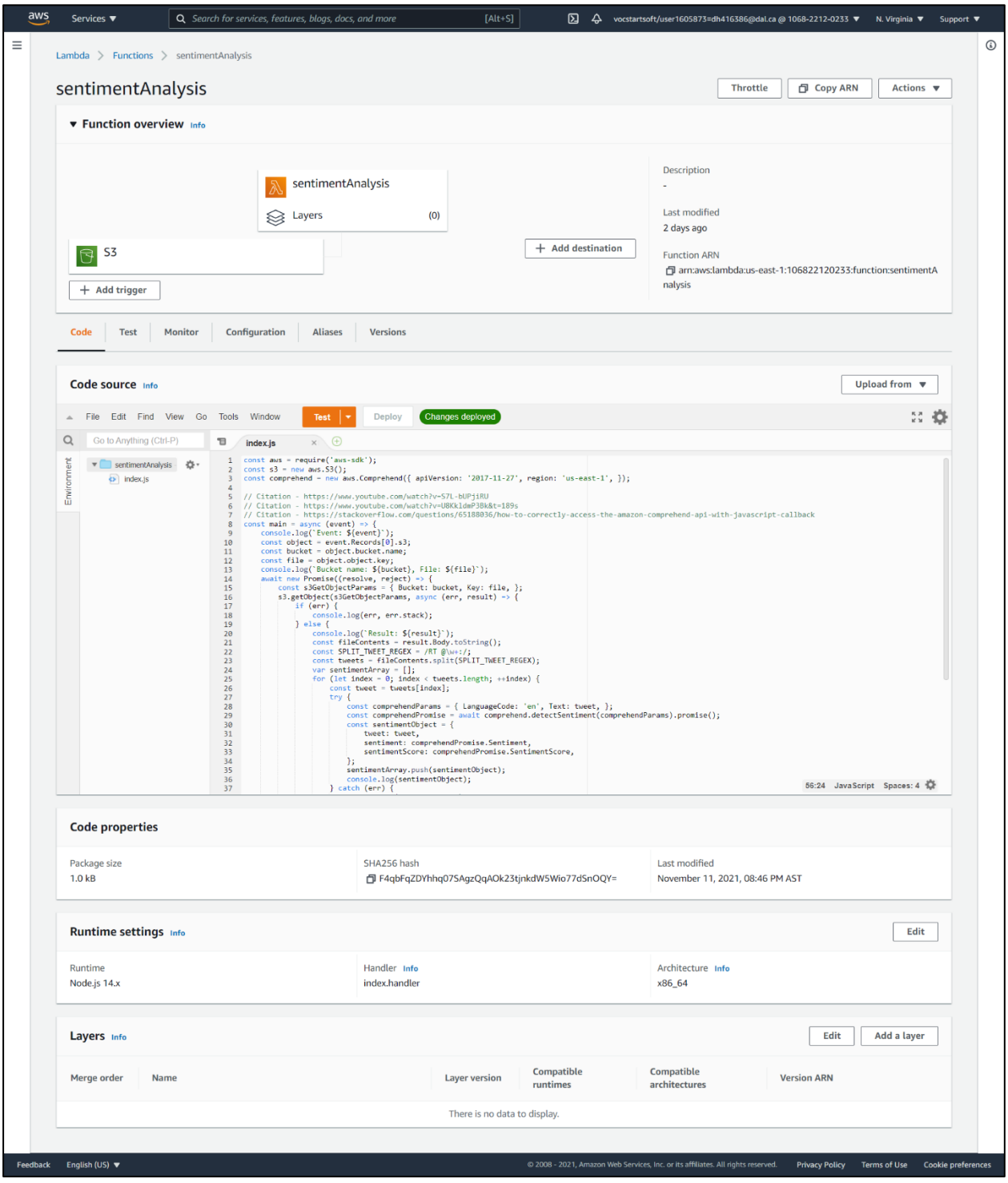

Figure 14 - sentimentAnalysis function with Lambda function written in Node.js framework [5]

c. Once the file containing all the tweets is uploaded on the bucket, AWS Comprehend is used to perform sentiment analysis of tweets.

Figures 15 and 16 display the screenshot of the log group of sentimentAnalysis Lambda function recorded when executed. This log group contains all the log messages printed on the console by the sentimentAnalysis Lambda function.

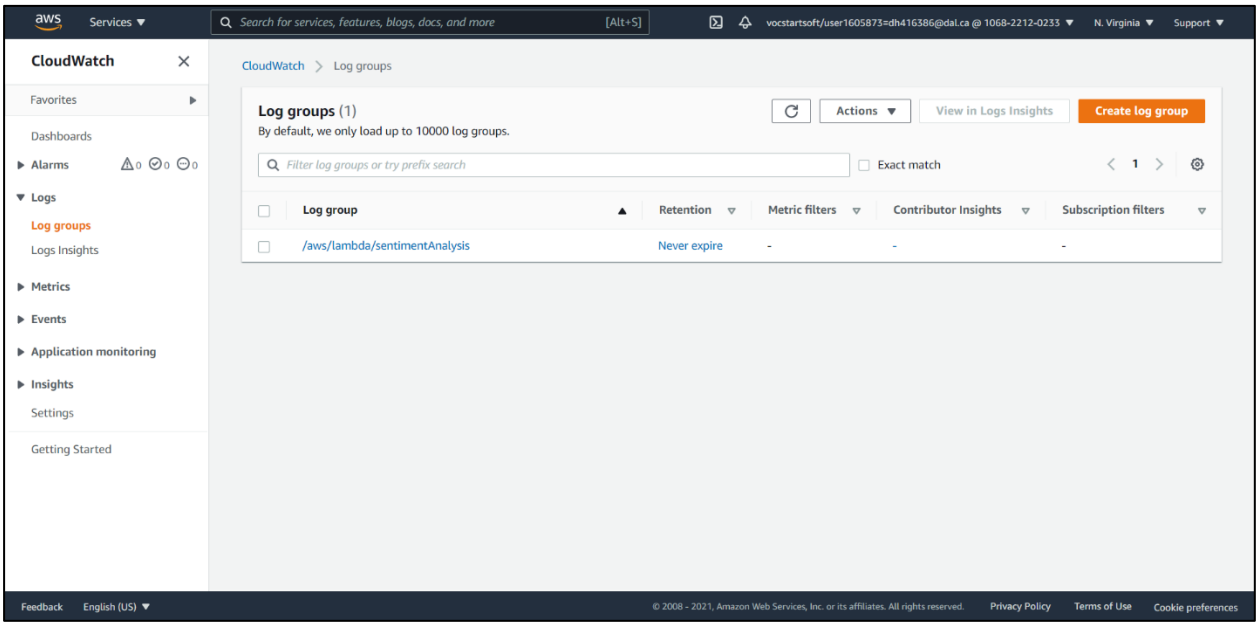

Figure 15 - sentimentAnalysis Lambda function log group [6]

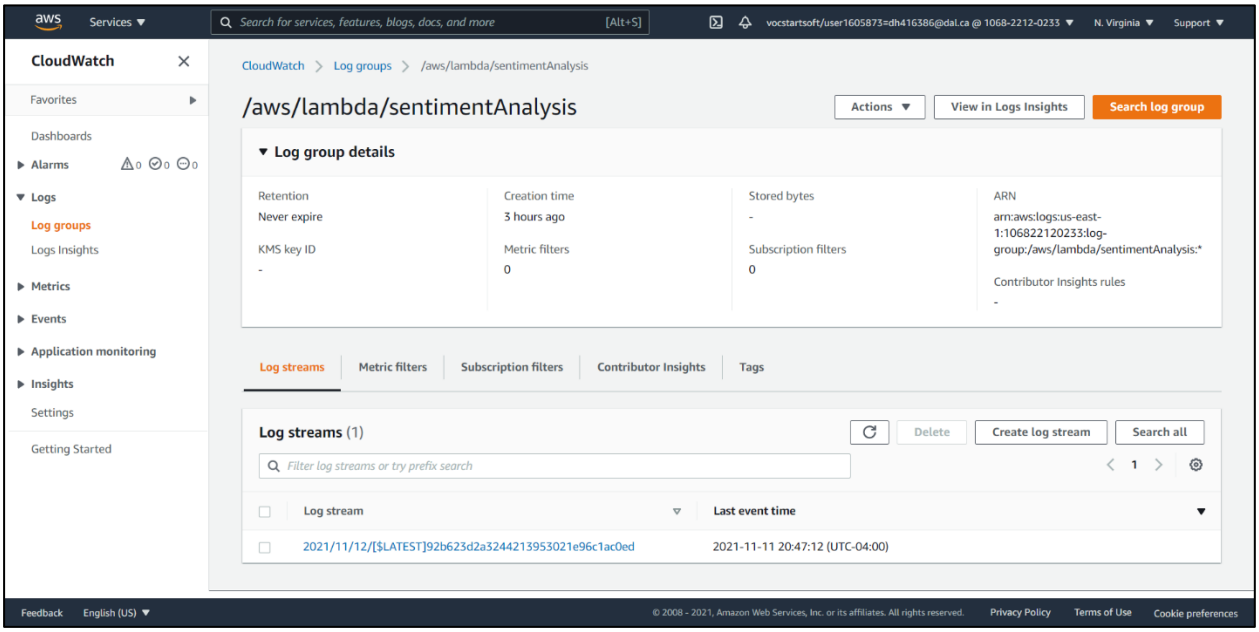

Figure 16 - sentimentAnalysis Lambda function log group [6]

Figure 17 displays the screenshot of logs captured by the AWS CloudWatch service when the sentimentAnalysis Lambda function was executed. The first two log messages are expended to display the JSON object that contains the original tweet, sentiment (i.e., POSITIVE, NEGATIVE, and NEUTRAL) and sentiment score.

Figure 17 - Log messages captured by AWS CloudWatch on the execution of sentimentAnalysis Lambda function [6]

Figure 18 displays the screenshot of the sentimentAnalysis Lambda function's configuration page. The timeout was increased from 3 seconds to 10 minutes.

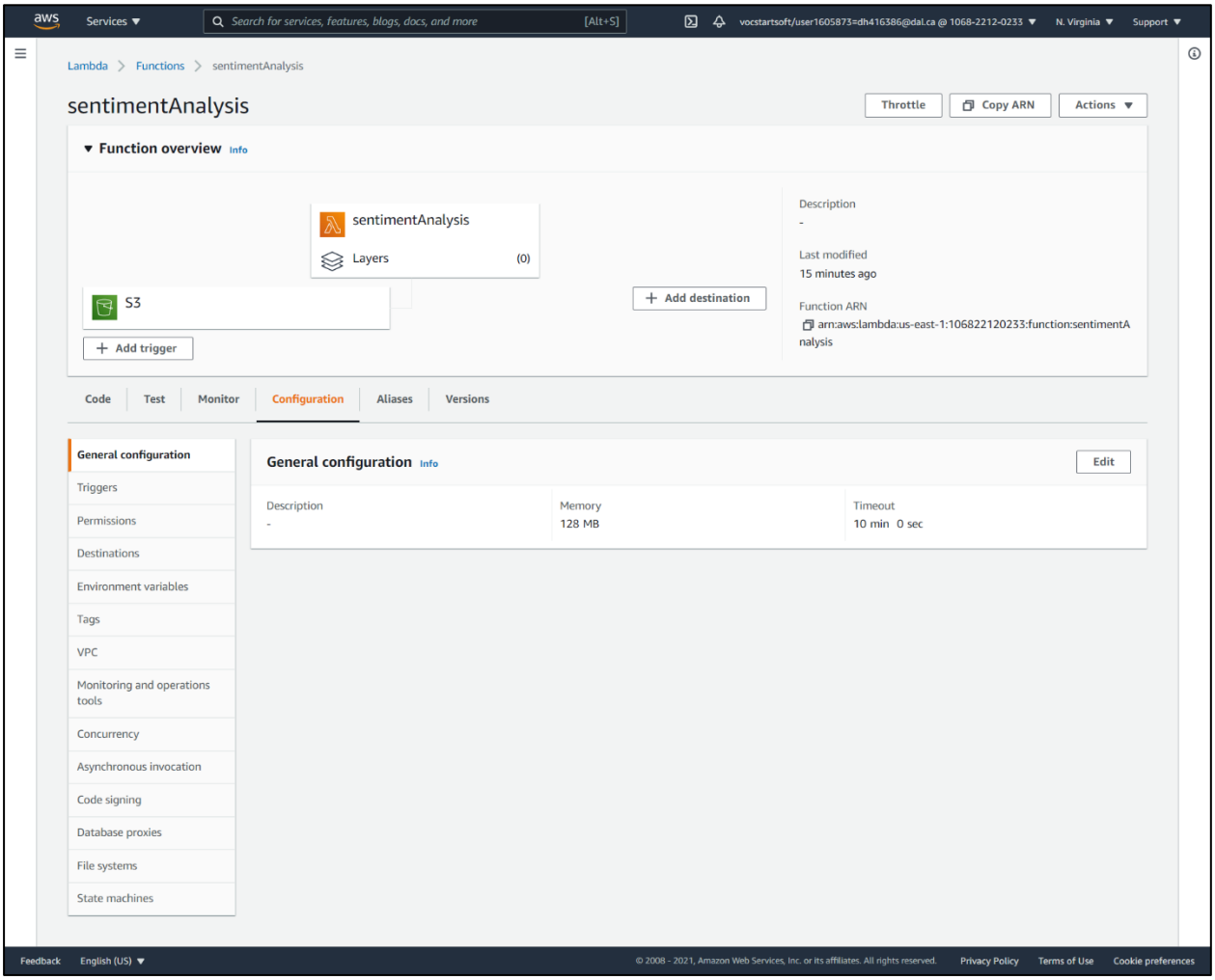

Figure 18 – sentimentAnalysis Lambda function Timeout was increased from 3 seconds to 10 minutes

Figure 19 displays the screenshot of the second S3 bucket creation page filled with details to create an S3 bucket with bucket name **twitterdataoutputb00857606**. This bucket contains three tags with other configuration parameters set to default. The JSON generated by sentimentAnalysis Lambda function is stored in this bucket with the file extension ".json".

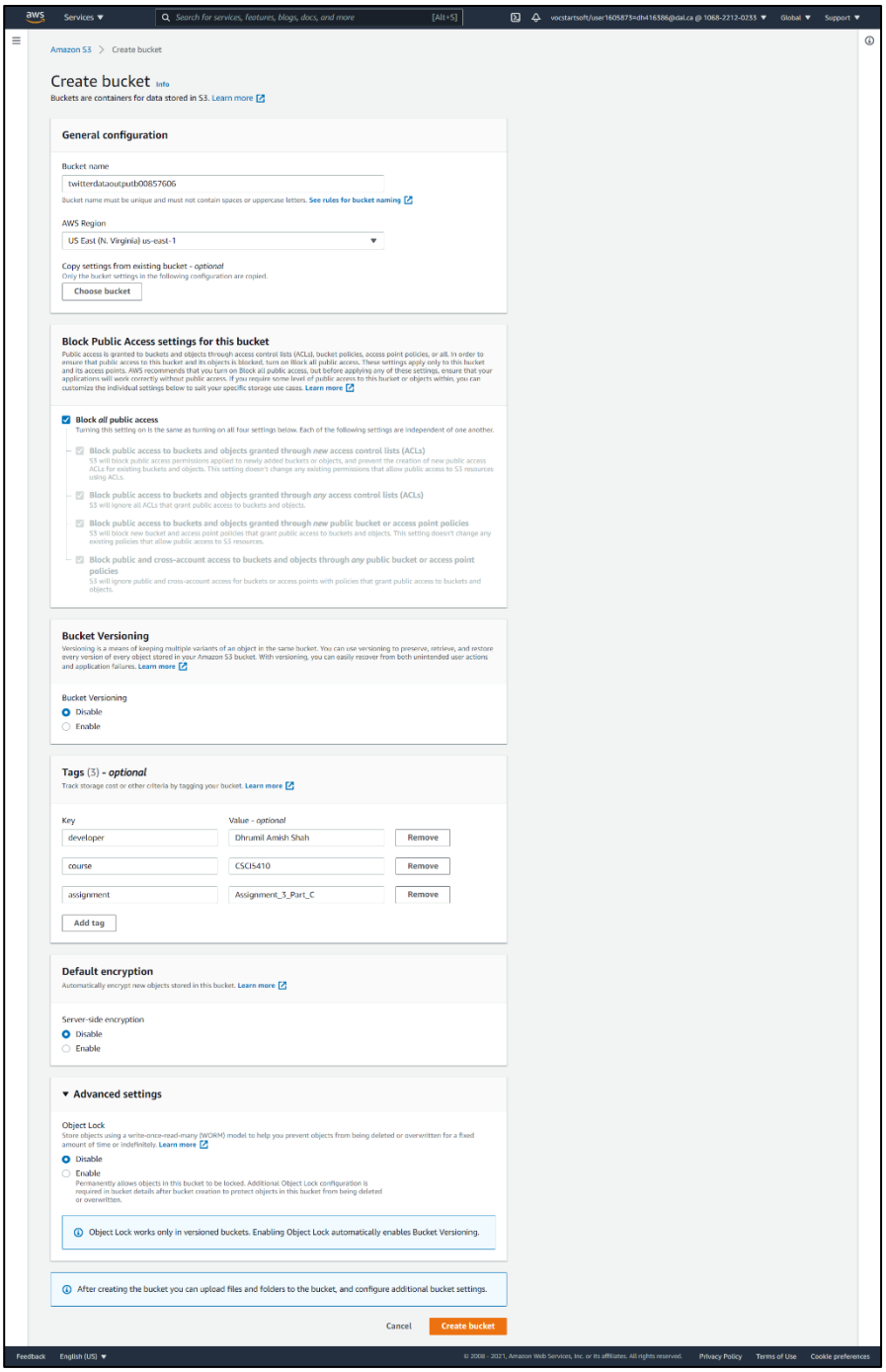

Figure 19 - Second S3 bucket twitterdataoutputb00857606 which holds the output JSON file [2]

Figure 20 displays the second bucket created which holds the final JSON file. The sentimentAnalysis Lambda function uploads the file with the ".json" extension to the second bucket twitterdataoutputb00857606.

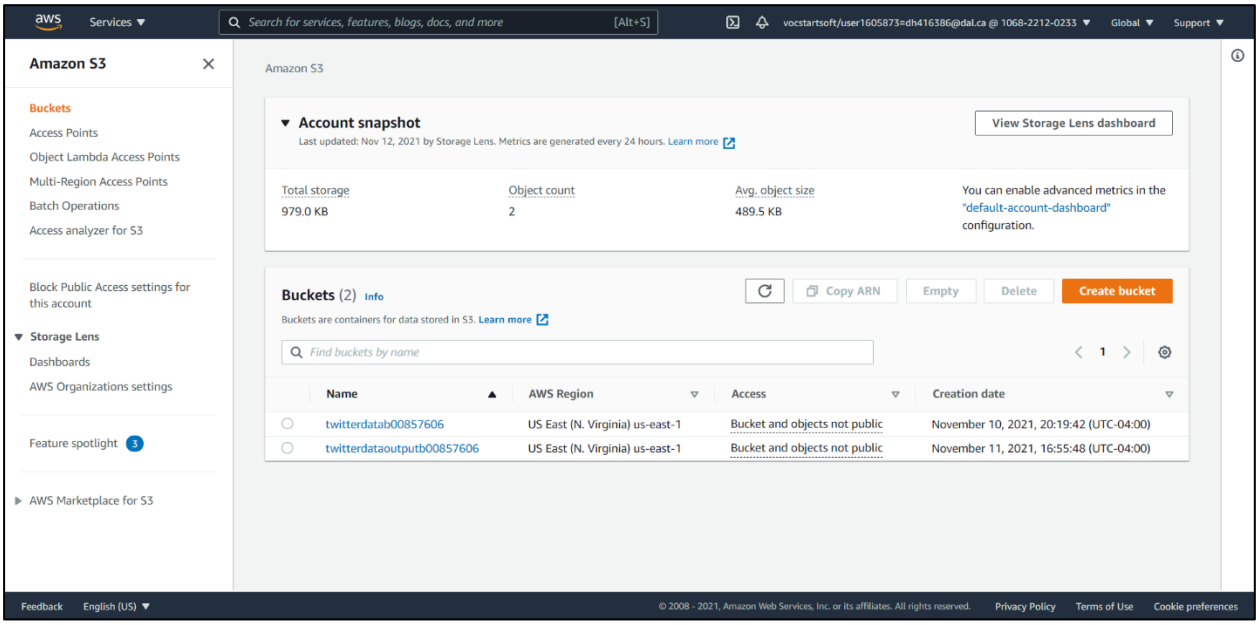

Figure 20 - Second S3 bucket twitterdataoutputb00857606 created [2]

Figure 21 displays the tweetsAnalyzed.json file uploaded to twitterdataoutputb00857606 S3 bucket by sentimentAnalysis Lambda function.

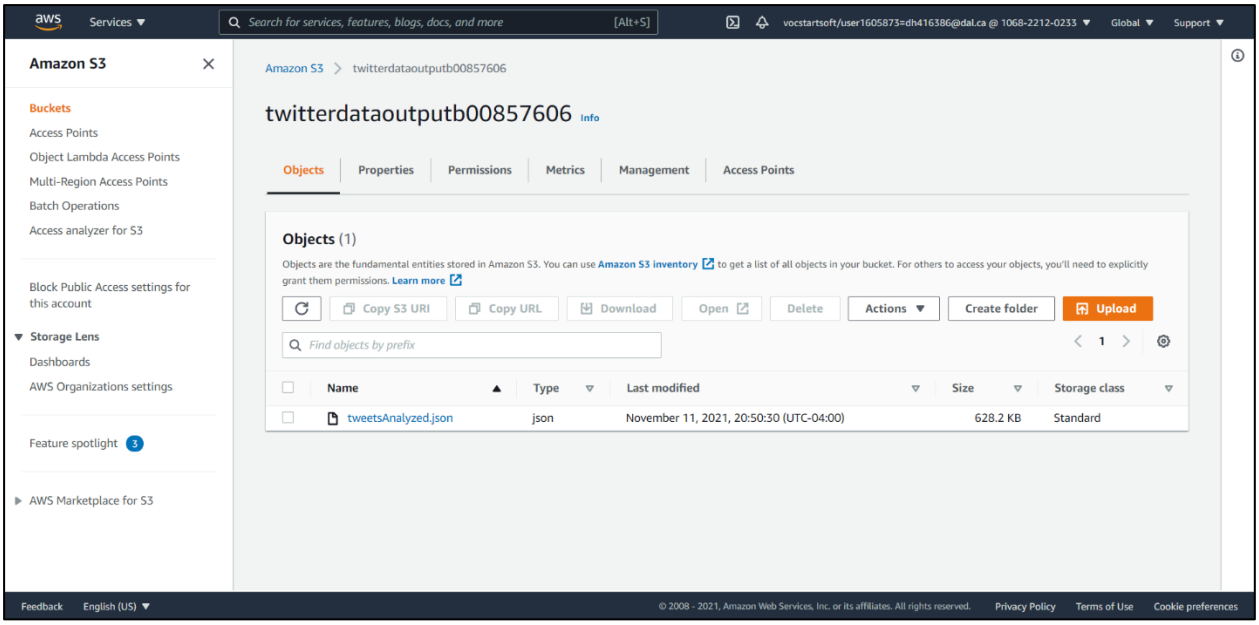

Figure 21 - tweetsAnalyzed.json file uploaded to twitterdataoutputb00857606 S3 bucket [2]

d. Your output should be captured in a CSV or JSON format.

Figures 22 and 23 display the JSON file **tweetsAnalyzed.json** that contains a JSON array with multiple JSON objects with each object having tweet, sentiment, and sentimentScore properties.

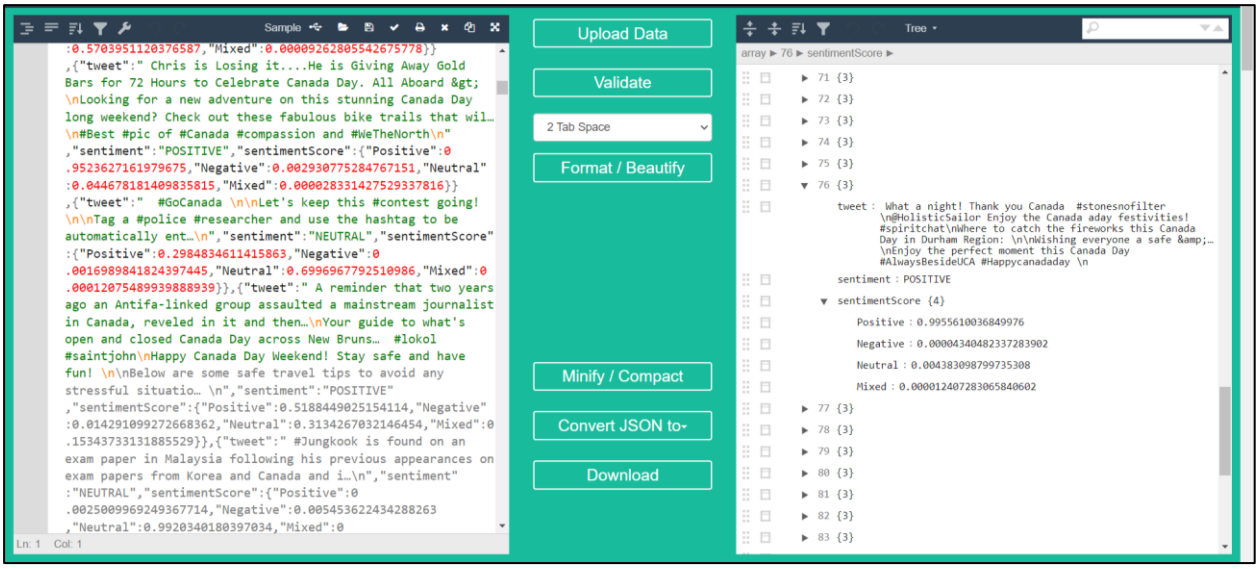

Figure 22 - Final output file - tweetsAnalyzed.json JSON file

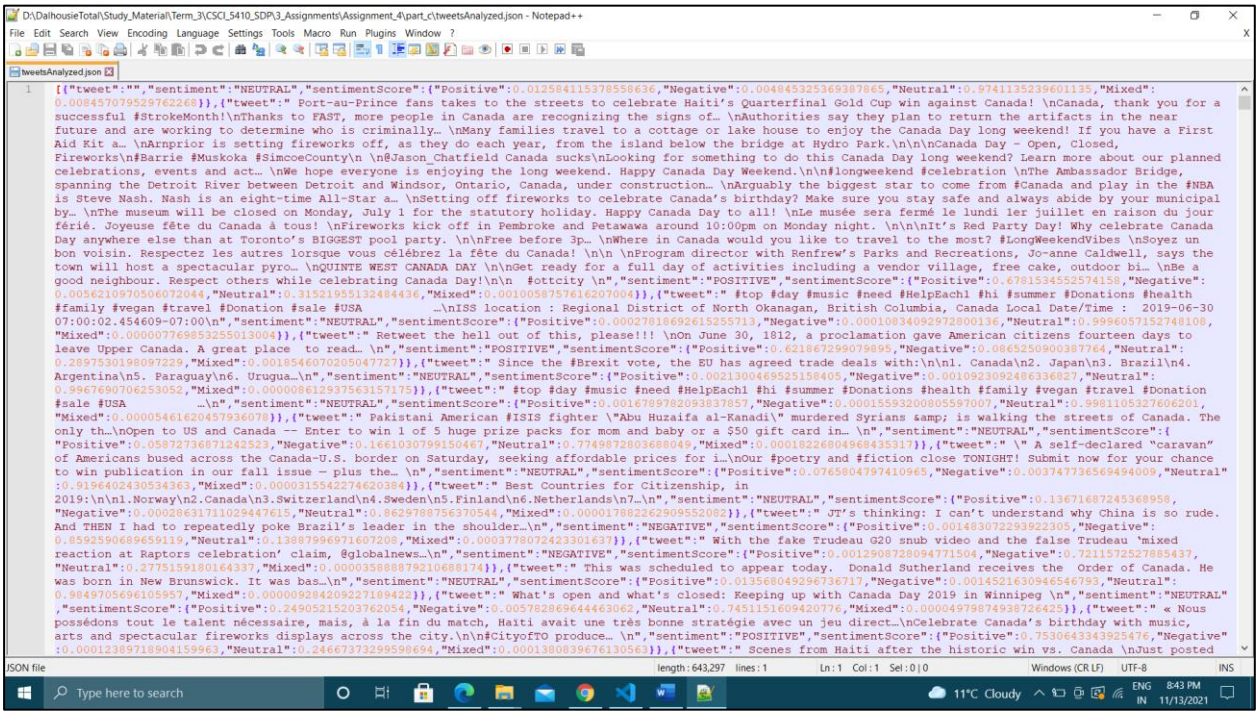

Figure 23 - Final output file - tweetsAnalyzed.json JSON file

Program files

index.js file – This file contains the logic responsible for uploading the text file with all the tweets to the AWS S3 bucket – **twitterdatab00857606**. Along with Node.js and Express.js frameworks, three packages were used namely **aws-sdk**, **dotenv**, and **multer**. The aws-sdk package consists of helper classes and methods to integrate various AWS services. (For example – AWS S3). The dotenv package hides the important credentials in a different file (i.e., file with name .env) so that credentials that are local to the development are not compromised. Finally, the multer package is responsible for the file storage and upload. The code in the file index.js is as below:

```
require('dotenv/config')
const express = require('express');
const multer = require('multer');
const aws = require('aws-sdk');
const app = express();
const port = 3000;

const s3 = new aws.S3({
    accessKeyId: process.env.AWS_ACCESS_KEY_ID,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
    sessionToken: process.env.AWS_SESSION_TOKEN
});

const storage = multer.memoryStorage({
    destination: function (req, file, callback) {
        callback(null, '');
    }
});
const upload = multer({ storage }).single('text-file');

app.post('/upload', upload, (req, res) => {
    console.log(req.file);
    let tweetFileName = req.file.originalname;
    const params = {
        Bucket: process.env.AWS_BUCKET_NAME,
        Key: tweetFileName,
        Body: req.file.buffer
    }
    s3.upload(params, (error, data) => {
        if (error) {
            res.status(500).json(error);
        } else {
            res.status(200).json(data);
        }
```

```
    });
});

app.listen(port, () => {
    console.log(`Server is up at PORT: ${port}`);
});
```

sentimentAnalysis.js file – This file contains the **sentimentAnalysis** Lambda function logic. It first reads the text file with all the tweets uploaded to the **twitterdatab00857606** S3 bucket, Then, it processes all the tweets individually, performs sentiment analysis using AWS Comprehend [7] on all the tweets, and finally create a JSON array with multiple JSON objects with each object having the original tweet, sentiment, and sentimentScore properties. Below is the code in the sentimentAnalysis.js file.

```
const aws = require('aws-sdk');
const s3 = new aws.S3();
const comprehend = new aws.Comprehend({ apiVersion: '2017-11-27', region: 'us-east-1', });

const main = async (event) => {
    console.log(`Event: ${event}`);
    const object = event.Records[0].s3;
    const bucket = object.bucket.name;
    const file = object.object.key;
    console.log(`Bucket name: ${bucket}, File: ${file}`);
    await new Promise((resolve, reject) => {
        const s3GetObjectParams = { Bucket: bucket, Key: file, };
        s3.getObject(s3GetObjectParams, async (err, result) => {
            if (err) {
                console.log(err, err.stack);
            } else {
                console.log(`Result: ${result}`);
                const fileContents = result.Body.toString();
                const SPLIT_TWEET_REGEX = /RT @\w+:/;
                const tweets = fileContents.split(SPLIT_TWEET_REGEX);
                var sentimentArray = [];
                for (let index = 0; index < tweets.length; ++index) {
                    const tweet = tweets[index];
                    try {
                        const comprehendParams = { LanguageCode: 'en', Text: tweet, };
                        const comprehendPromise = await
comprehend.detectSentiment(comprehendParams).promise();
                        const sentimentObject = {
                            tweet: tweet,
                            sentiment: comprehendPromise.Sentiment,
                            sentimentScore: comprehendPromise.SentimentScore,
```

```
                    };
                    sentimentArray.push(sentimentObject);
                    console.log(sentimentObject);
                } catch (err) {
                    console.log(err, err.stack);
                }
            };
            console.log(`Output:`);
            const bucketName = "twitterdataoutputb00857606";
            const fileName = "tweetsAnalyzed.json";
            const s3UploadFileParams = { Bucket: bucketName, Key: fileName, Body:
JSON.stringify(sentimentArray) };
            try {
                await s3.upload(s3UploadFileParams).promise();
                console.log(`Output in file ${fileName} and bucket ${bucketName}.`);
            } catch (err) {
                console.log(err, err.stack);
            }
        };
    });
});

exports.handler = main;
```

.env file – This file contains the credentials required to access the AWS services. Along with that, it contains the bucket name to where the initial text file with all the tweets is to be uploaded. Below is the code from the .env file:

```
AWS_ACCESS_KEY_ID = "AWS_ACCESS_KEY_ID"
AWS_SECRET_ACCESS_KEY = "AWS_SECRET_ACCESS_KEY"
AWS_SESSION_TOKEN = "AWS_SESSION_TOKEN"
AWS_BUCKET_NAME = "twitterdatab00857606"
```

package.json file – This file contains all the necessary dependencies required to run this program. Along with that, it contains other properties like project name, project description, project version, the main starting point of the program, scripts, keywords, author, and licence.

```
{
  "name": "csci5410_a3_part_c_code",
  "version": "1.0.0",
  "description": "CSCI5410 A3 Part C Code - Event driven serverless application using AWS
Comprehend",
  "main": "index.js",
  "scripts": {
```

```
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "aws",
    "s3",
    "lambda",
    "comprehend",
    "csci5410",
    "serverless-data-processing",
    "sdp",
    "amazon",
    "dalhousie-university",
    "dal"
  ],
  "author": "Dhrumil Amish Shah",
  "license": "ISC",
  "dependencies": {
    "aws-sdk": "^2.1026.0",
    "dotenv": "^10.0.0",
    "express": "^4.17.1",
    "multer": "^1.4.3"
  }
}
```

GitLab Source Code

The source code for **Part C** is available at GitLab URL given below:
https://git.cs.dal.ca/dashah/csci-5410-f2021-b00857606-dhrumil-amish-shah/-/tree/main/assignment_4_code/part_c.

References

[1] draw.io, "Flowchart Maker & Online Diagram Software," draw.io, [Online]. Available: https://app.diagrams.net/. [Accessed 14 November 2021].

[2] AWS, "Amazon S3," Amazon, [Online]. Available: https://aws.amazon.com/s3/. [Accessed 14 November 2021].

[3] Postman, Inc., "Build APIs together," Postman, Inc., [Online]. Available: https://www.postman.com/. [Accessed 14 November 2021].

[4] AWS, "AWS Identity and Access Management (IAM)," Amazon, [Online]. Available: https://aws.amazon.com/iam/. [Accessed 14 November 2021].

[5] AWS, "AWS Lambda," Amazon, [Online]. Available: https://aws.amazon.com/lambda/. [Accessed 14 November 2021].

[6] AWS, "Amazon CloudWatch," Amazon, [Online]. Available: https://aws.amazon.com/cloudwatch/. [Accessed 14 November 2021].

[7] Amazon, "Amazon Comprehend," AWS, [Online]. Available: https://aws.amazon.com/comprehend/. [Accessed 14 November 2021].