BUILD AN EVENT–DRIVEN SERVERLESS APPLICATION USING AWS LAMBDA

CSCI 5410 – Serverless Data Processing
Assignment 3 – Part A

Dhrumil Amish Shah (B00857606)
dh416386@dal.ca

Project Git Repository

https://git.cs.dal.ca/dashah/csci-5410-f2021-b00857606-dhrumil-amish-shah/-/tree/main/assignment_3_code

Event-Driven Serverless Application Using AWS Lambda and Java

a. Create your 1st S3 bucket FirstB00xxxxxx and 2nd S3 bucket SecondB00xxxxxx using AWS SDK (any programming language)

Figure 1 displays the screenshot of an empty Amazon S3 bucket. (i.e., No S3 buckets initially)

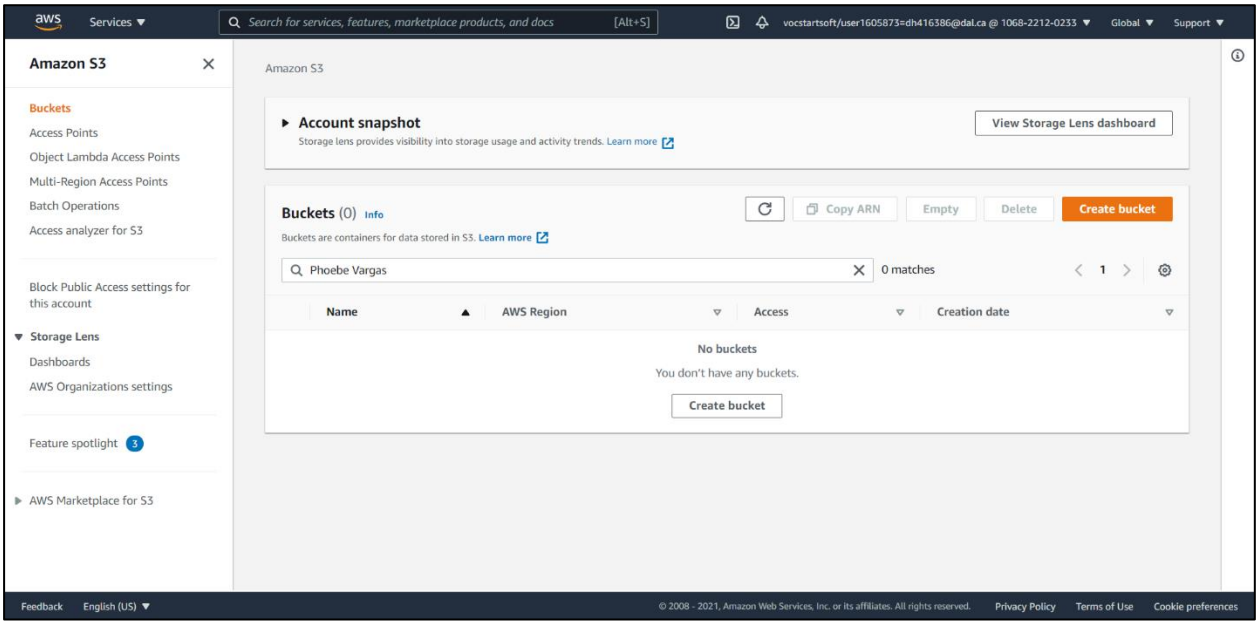

Figure 1 - Empty Amazon S3 (i.e., No buckets) [1]

Figure 2 displays the creation of two S3 buckets namely "first-b00857606" and "second-b00857606" using JAVA programming language and AWS JAVA SDK in the US East (N. Virginia) us-east-1 region.

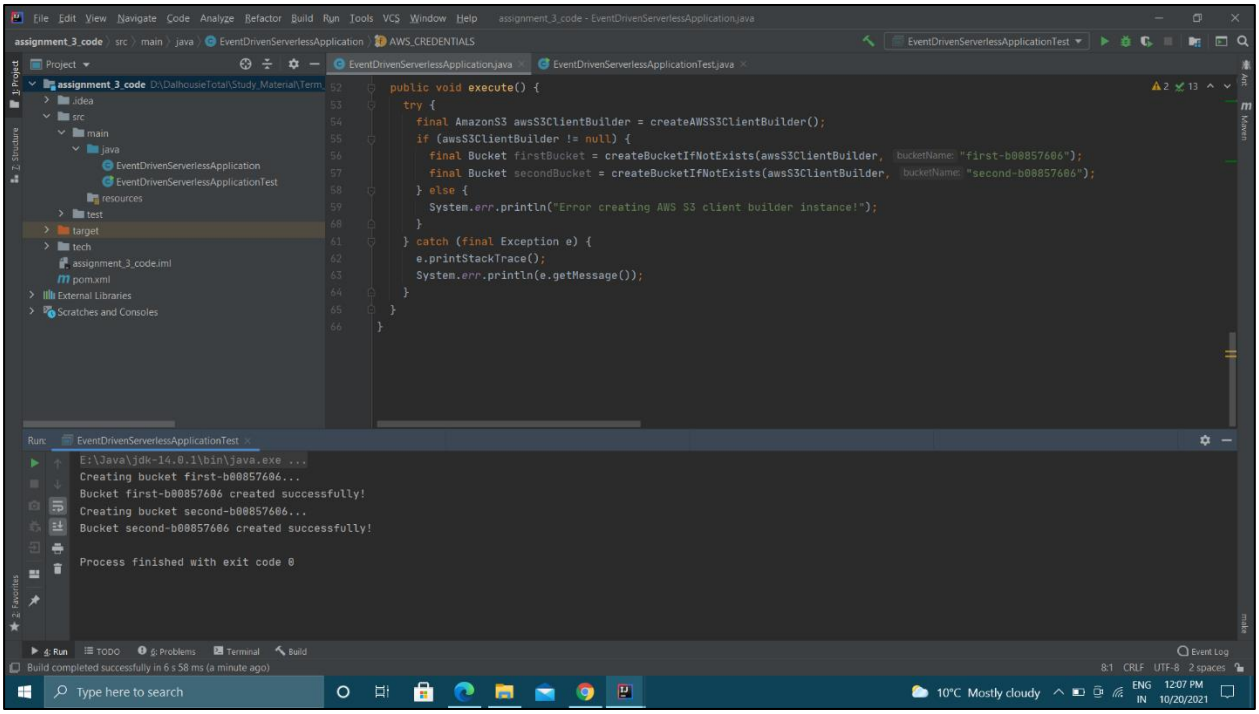

Figure 2 - Creation of buckets "first-b00857606" and "second-b00857606" using JAVA programming language and AWS JAVA SDK

Figure 3 displays two S3 buckets namely “first-b00857606” and “second-b00857606” created in the us-east-1 region. (AWS S3 Buckets console image)

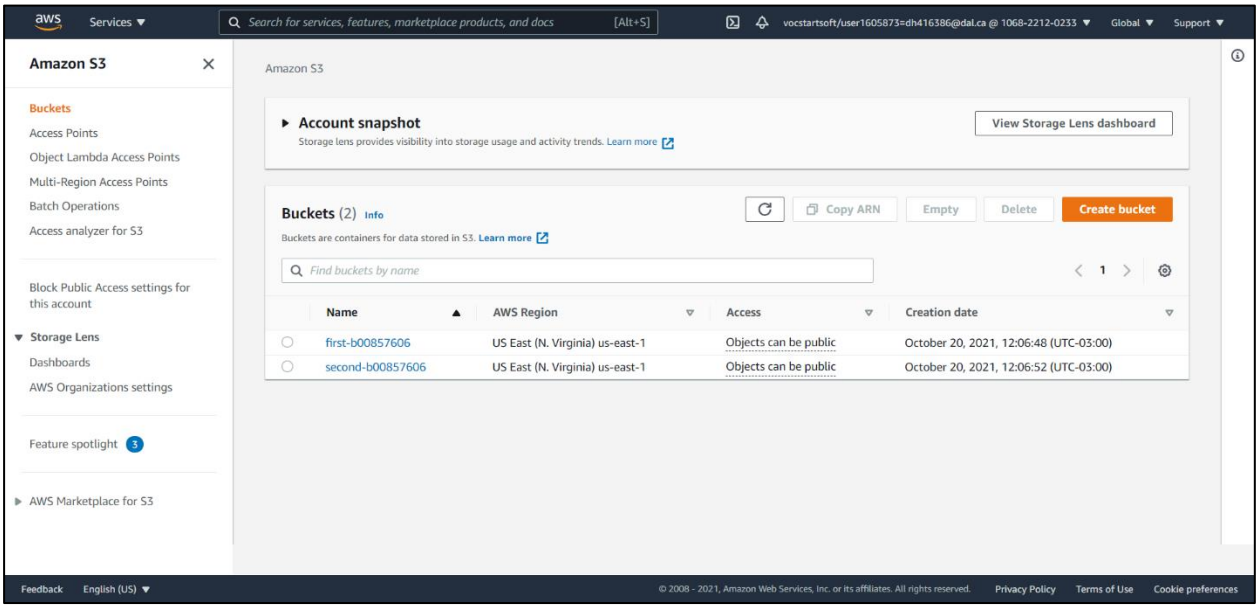

Figure 3 - Two S3 buckets namely "first-b00857606" and "second-b00857606" in Amazon S3 console [1]

b. Upload the files given in the Tech folder one at a time with a delay of 200 milliseconds on the 1st bucket. You need to write a script using SDK to upload the files one at a time to the S3 bucket.

Figure 4 displays the creation of a new role "AWSLambdaS3FullAccess" which consist of three policies namely AmazonS3FullAccess, AWSLambdaBasicExecutionRole, and AWSLambda_FullAccess. This role is attached to the first lambda function (i.e., the "extractFeatures" lambda function).

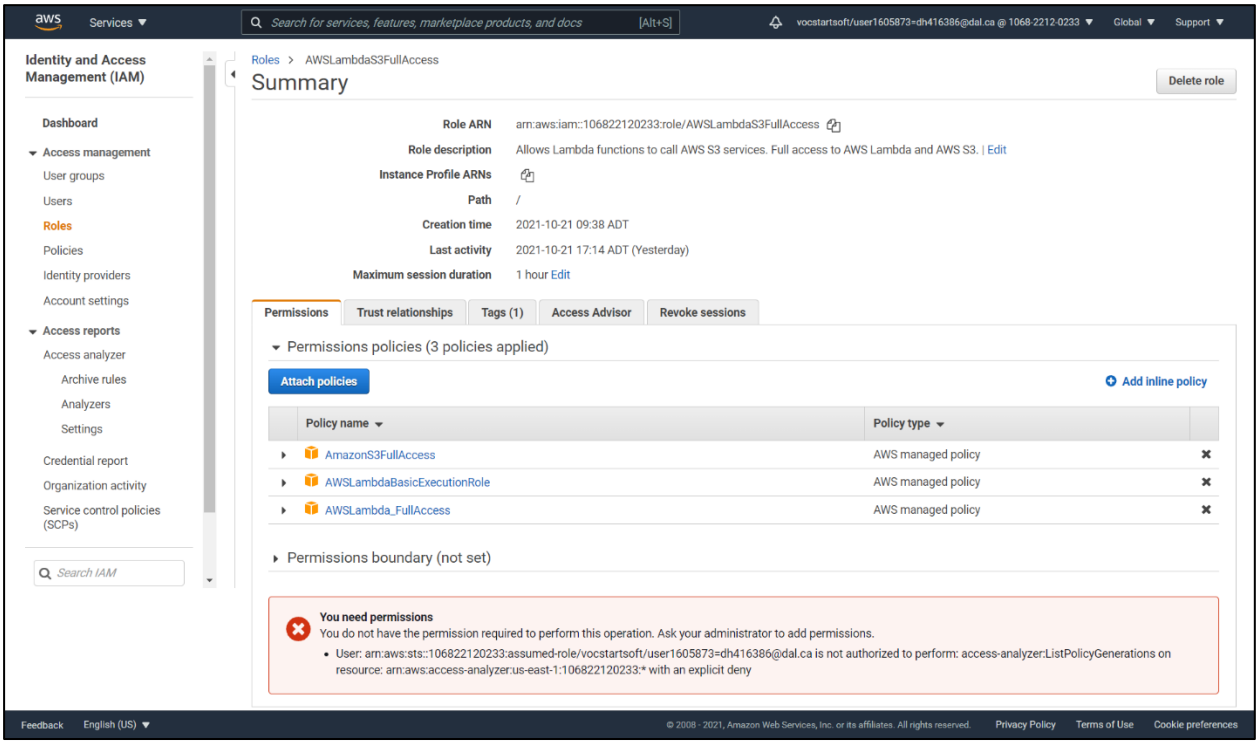

Figure 4 - Creation of new Role "AWSLambdaS3FullAccess" [2]

Figure 5 displays the creation of a new role "AWSLambdaS3DynamoDBFullAccess" that consist of four policies namely AmazonS3FullAccess, AmazonDynamoDBFullAccess, AWSLambdaBasicExecutionRole, AWSLambda_FullAccess. This role is attached to the second lambda function (i.e., the "accessDB" lambda function).

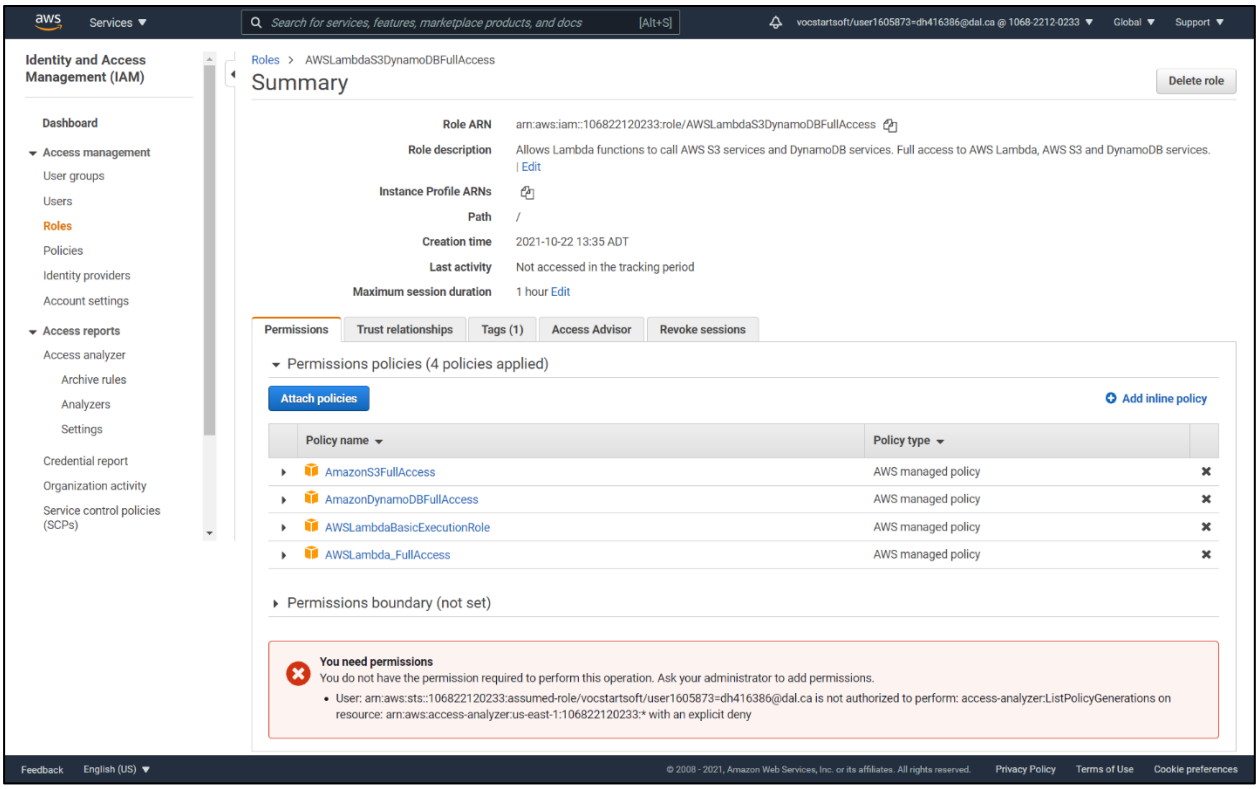

Figure 5 - Creation of new Role "AWSLambdaS3DynamoDBFullAccess" [2]

Figure 6 displays the creation of a lambda function "extractFeatures" with JAVA runtime Java 8 on Amazon Linux 1 with execution role – AWSLambdaS3FullAccess.

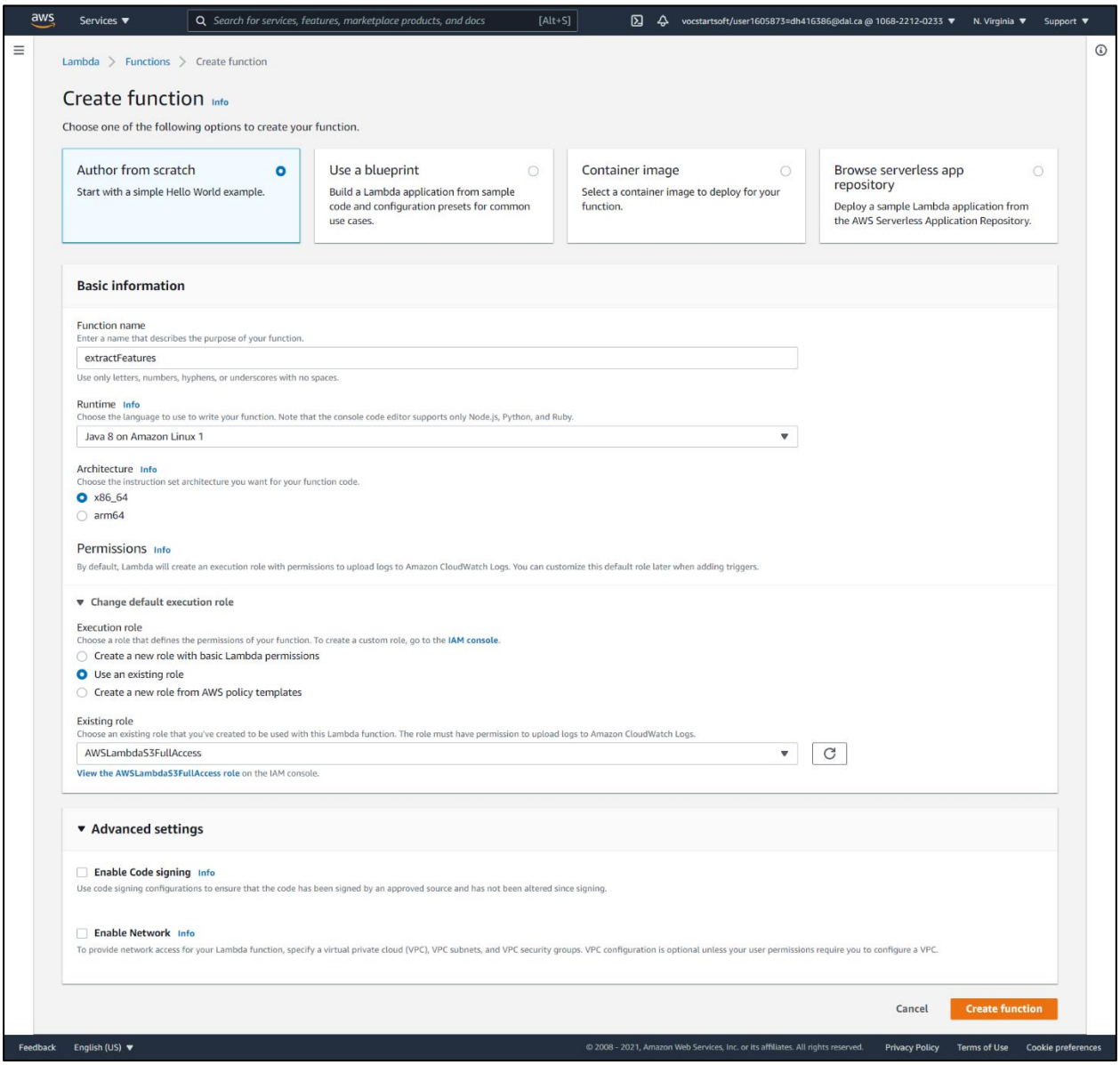

Figure 6 - Creation of "extractFeatures" lambda function [3]

Figure 7 displays the creation of a trigger on lambda “extractFeatures” attached to bucket “first-b00857606”. This trigger is triggered when an object is uploaded to bucket “first-b00857606”.

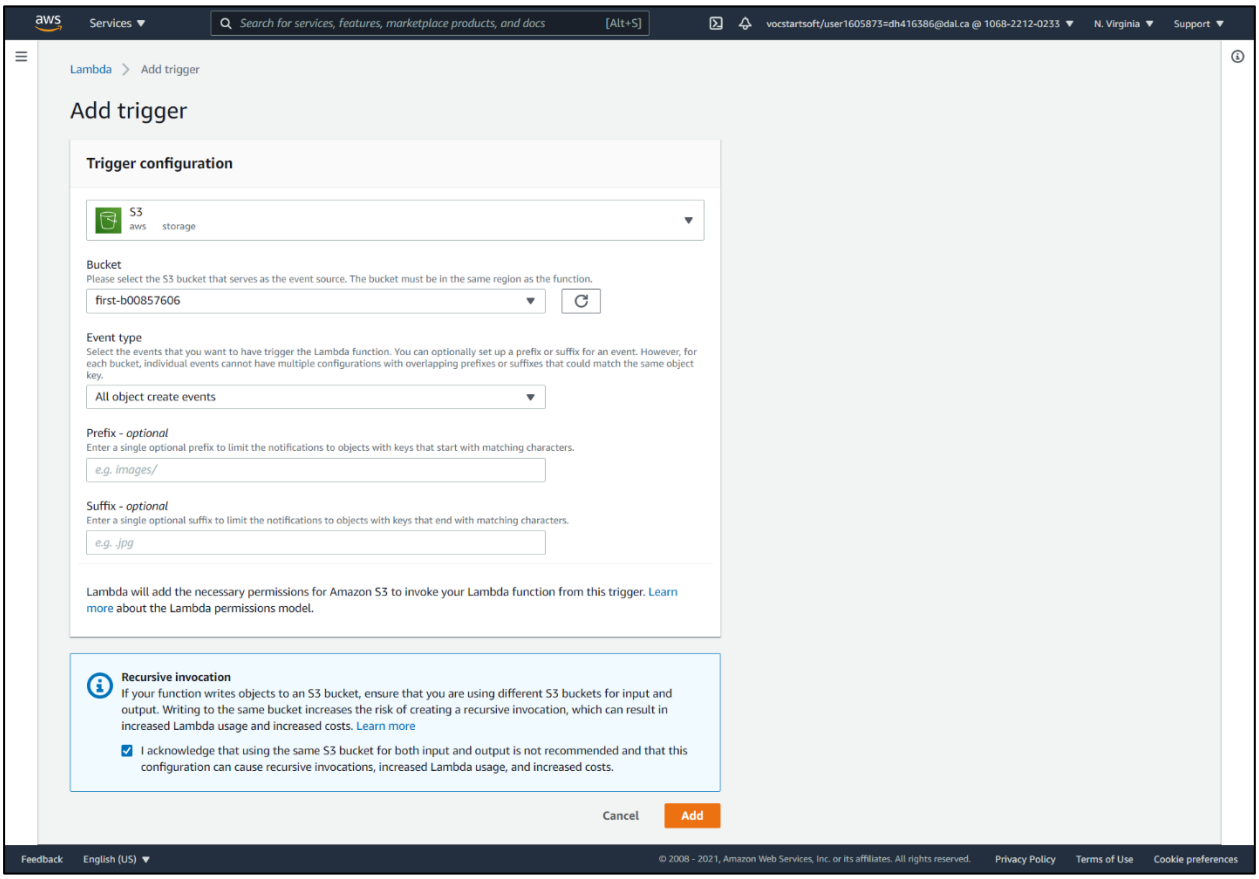

Figure 7 - Creation of trigger on lambda “extractFeatures” attached to bucket “first-b00857606” [3]

Figure 8 displays the creation of a lambda function “accessDB” with JAVA runtime Java 8 on Amazon Linux 1 with execution role – AWSLambdaS3DynamoDBFullAccess.

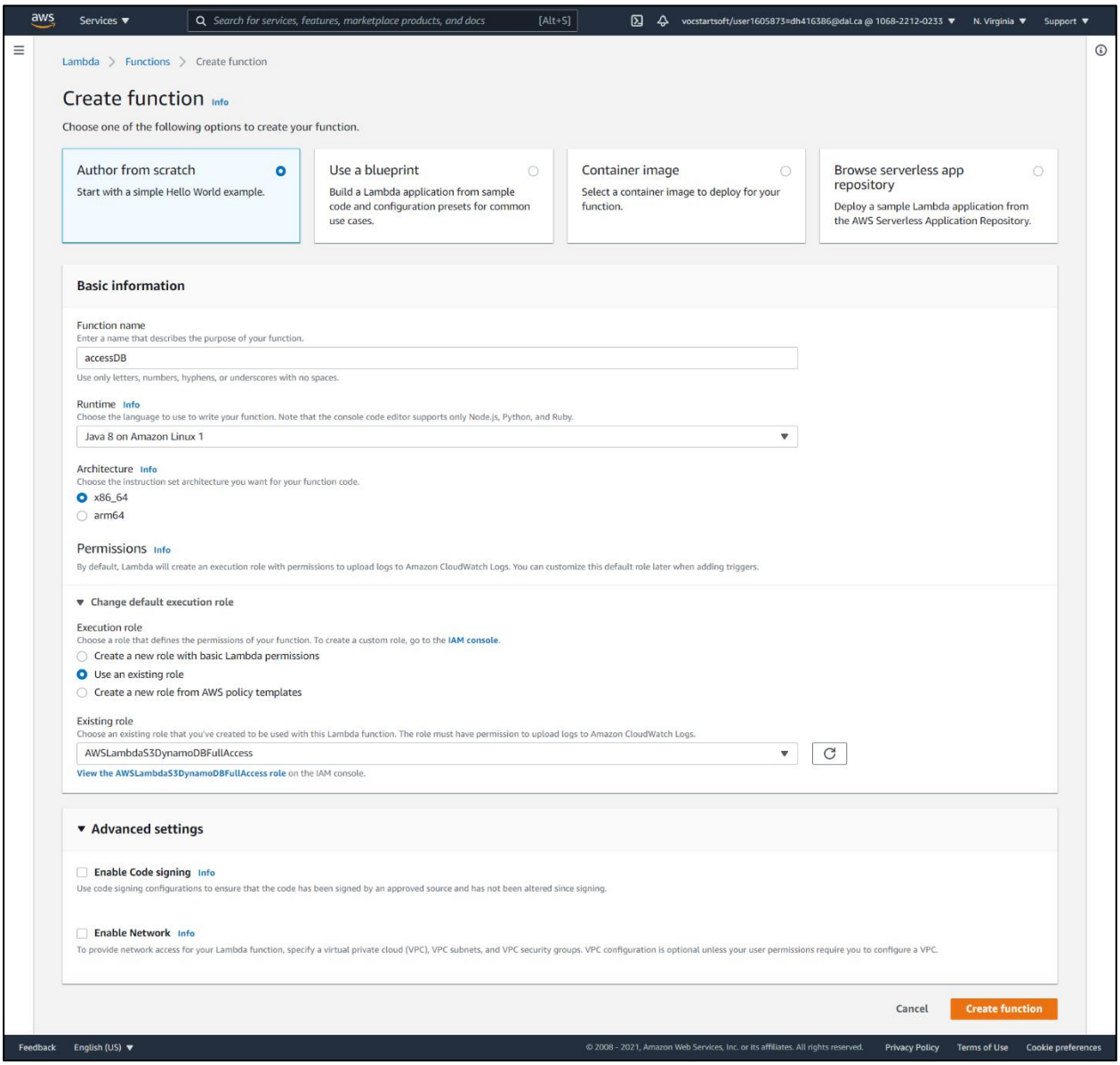

Figure 8 - Creation of "accessDB" lambda function [3]

Figure 9 displays the creation of a trigger on lambda “accessDB” attached to bucket “second-b00857606”. This trigger is triggered when an object is uploaded to bucket “second-b00857606”.

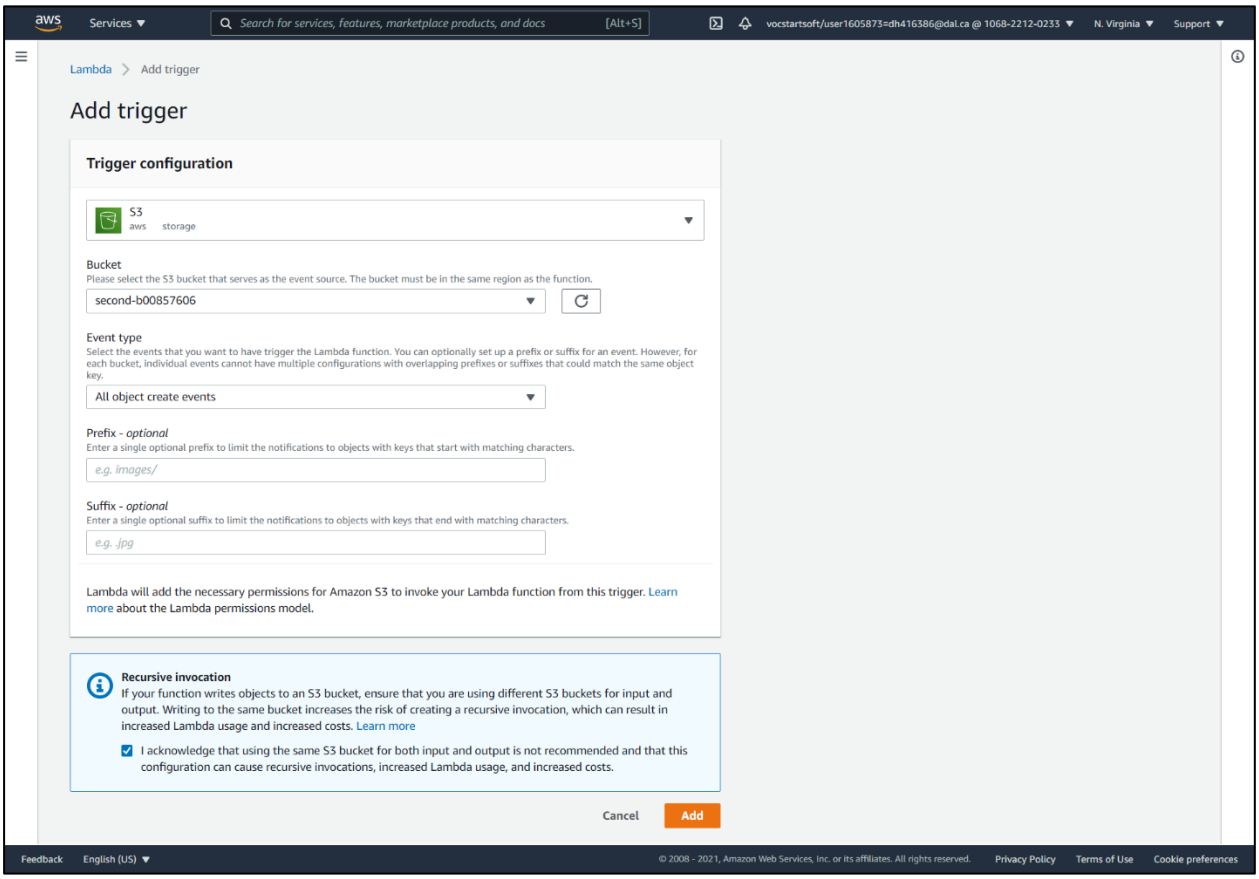

Figure 9 - Creation of trigger on lambda “accessDB” attached to bucket “second-b00857606” [3]

Figure 10 displays the dashboard of the "extractFeatures" lambda function. When an object is uploaded to the "first-b00857606" bucket, the handleRequest method of class ExtractFeaturesLambda in package assignment3.lambda is triggered.

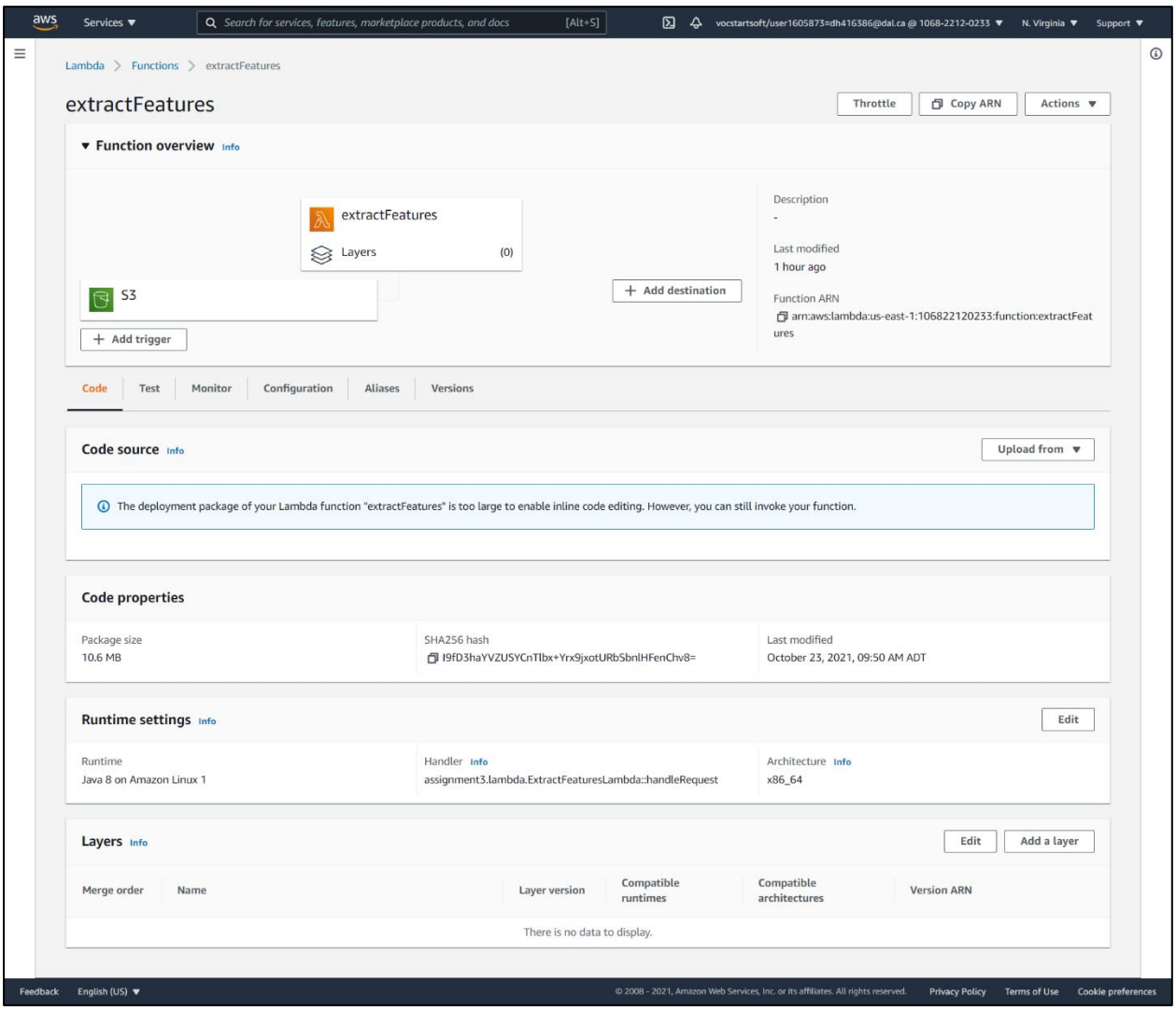

Figure 10 - "extractFeatures" lambda function dashboard [3]

Figure 11 displays the dashboard of the "accessDB" lambda function. When an object is uploaded to the "second-b00857606" bucket, the handleRequest method of class AccessDBLambda in package assignment3.lambda is triggered.

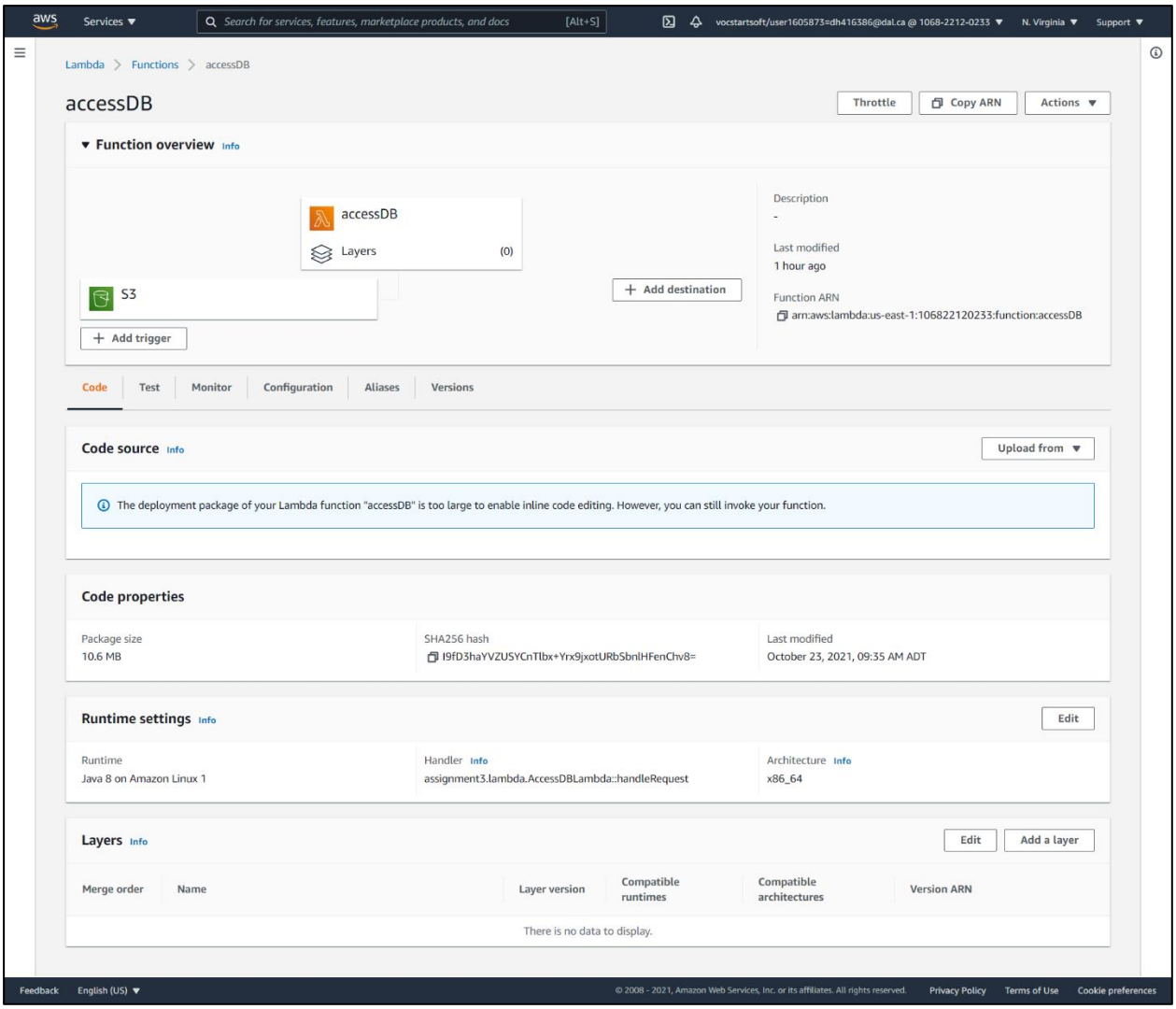

Figure 11 - "accessDB" lambda function dashboard [3]

Figures 12, 13 and **14** display the log messages printed on the console for all files uploaded to the first S3 bucket (i.e., "first-b00857606" S3 bucket).

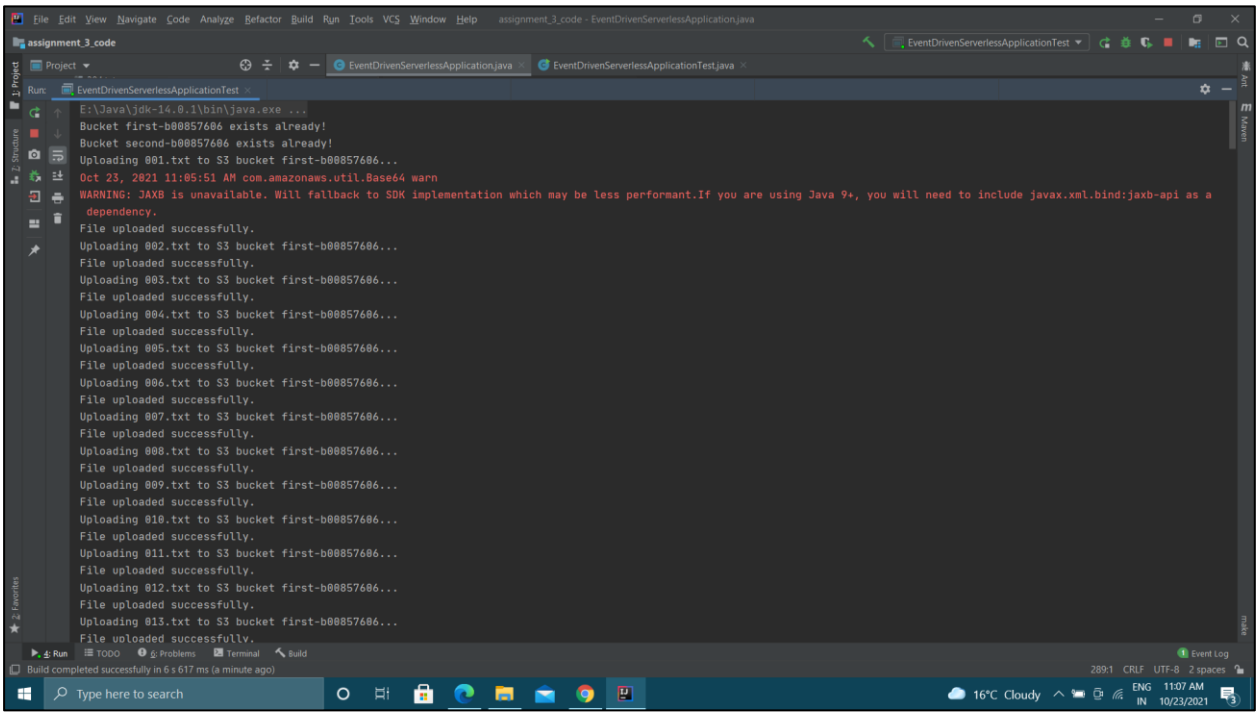

Figure 12 - Log messages of files uploaded to "first-b00857606" S3 bucket

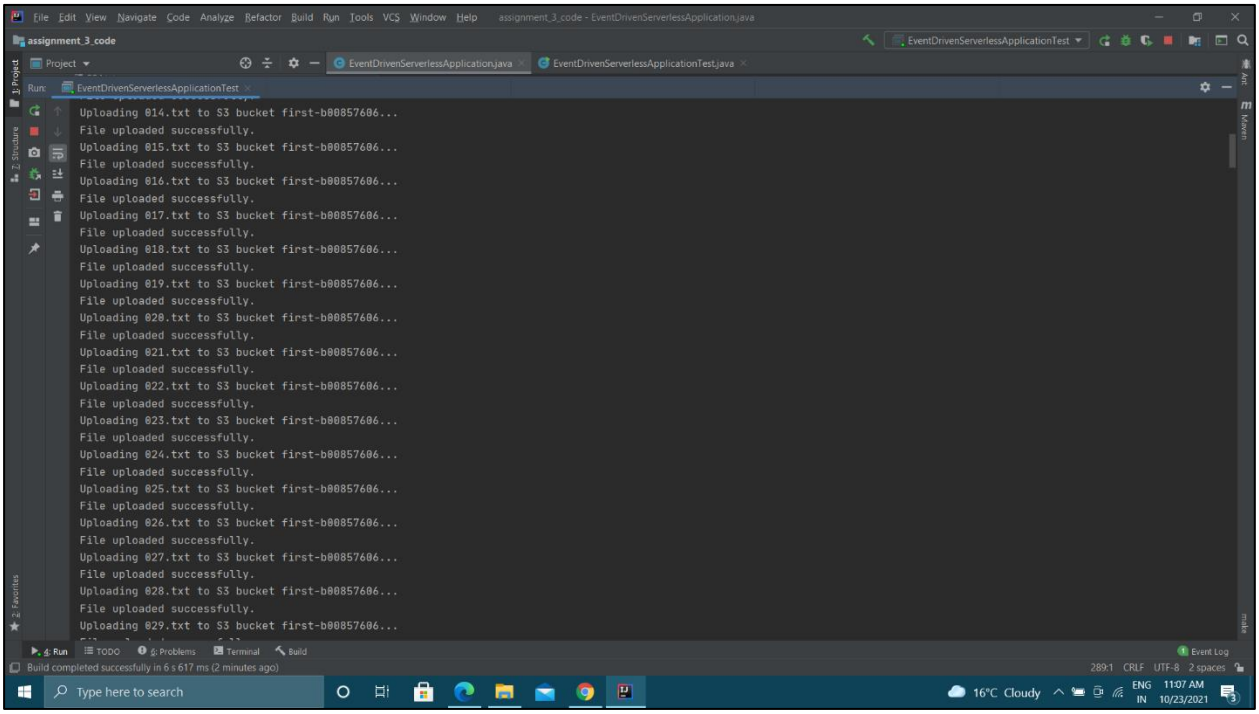

Figure 13 - Log messages of files uploaded to "first-b00857606" S3 bucket (contd.)

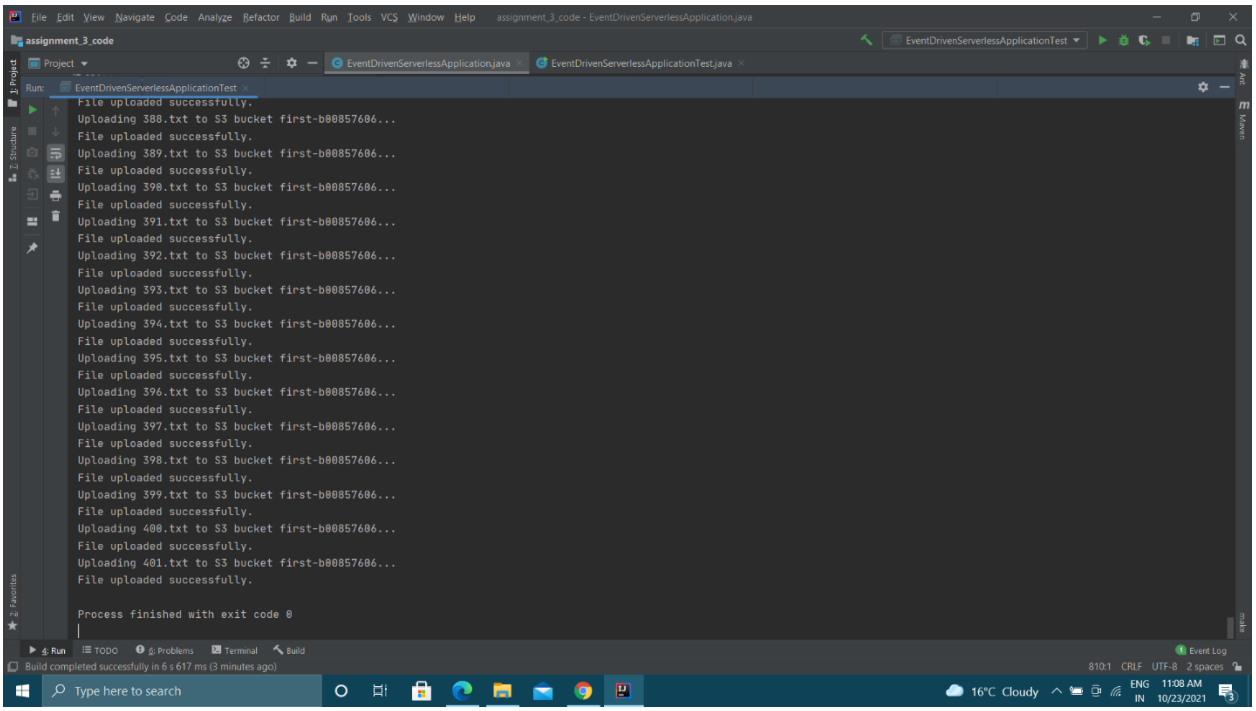

Figure 14 - Log messages of files uploaded to "first-b00857606" S3 bucket (contd.)

Figures 15, 16 and **17** display all files uploaded to the "first-b00857606" S3 bucket.

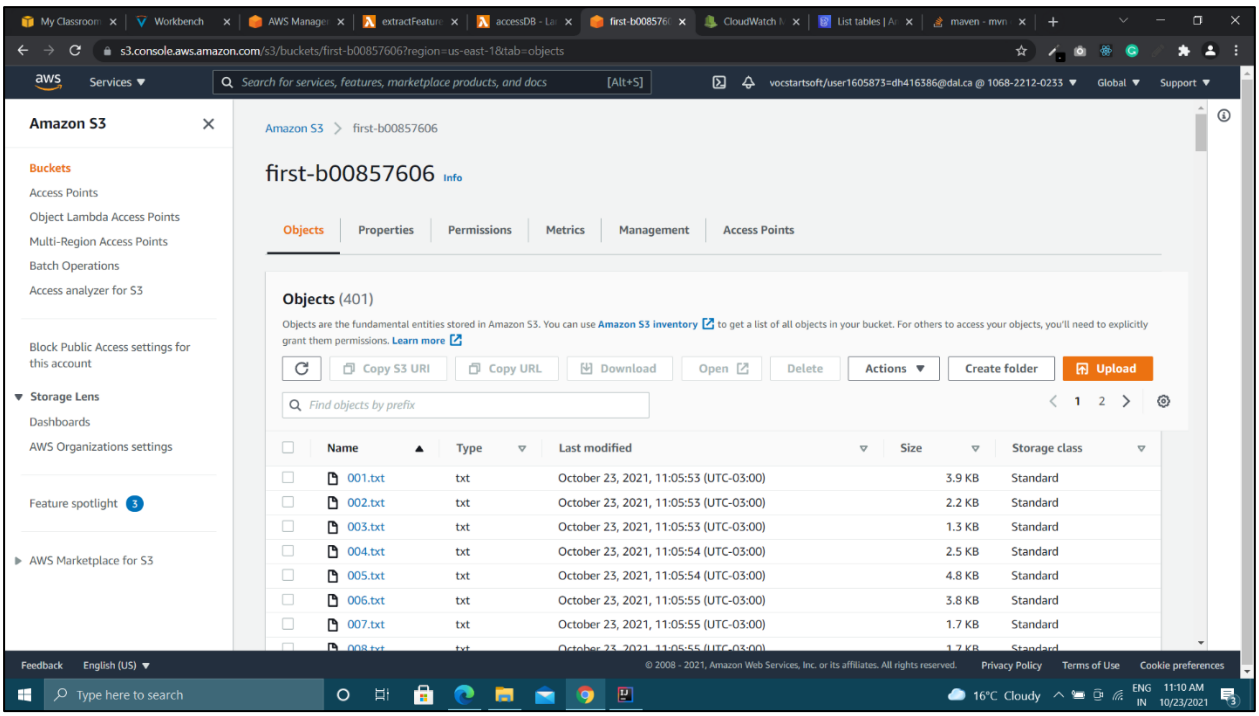

Figure 15 - Files uploaded to "first-b00857606" S3 bucket [1]

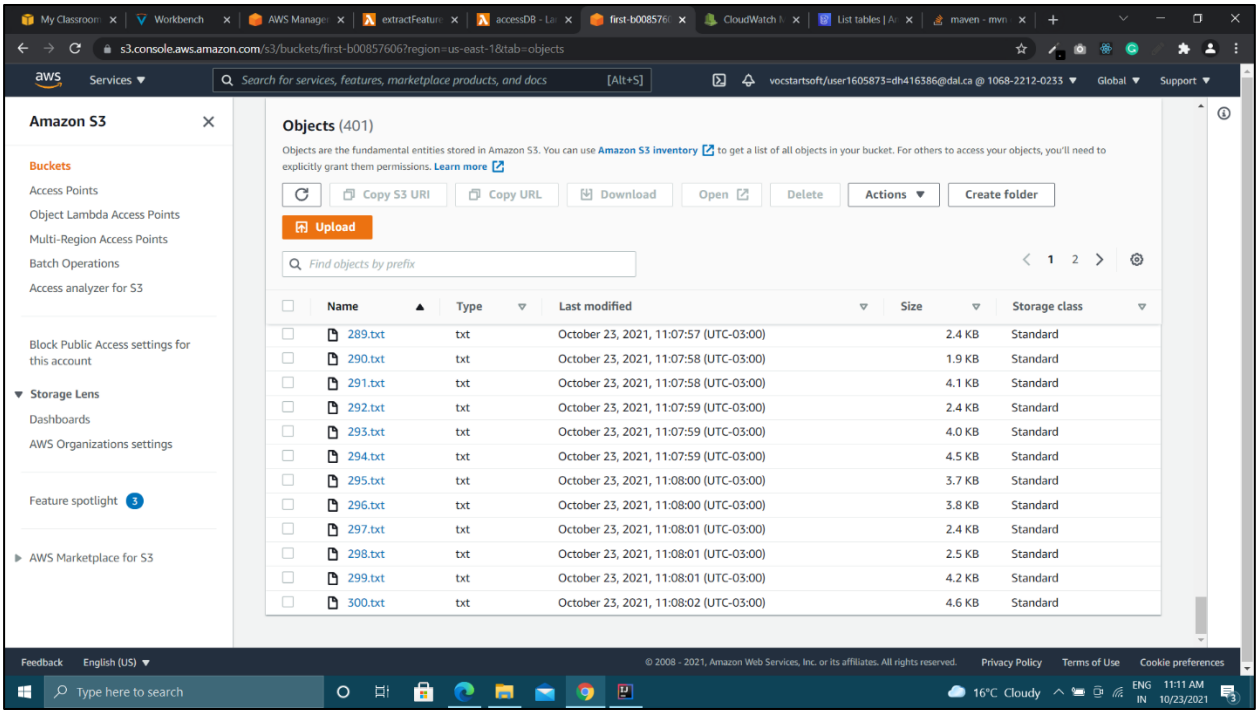

Figure 16 - Files uploaded to "first-b00857606" S3 bucket (contd.) [1]

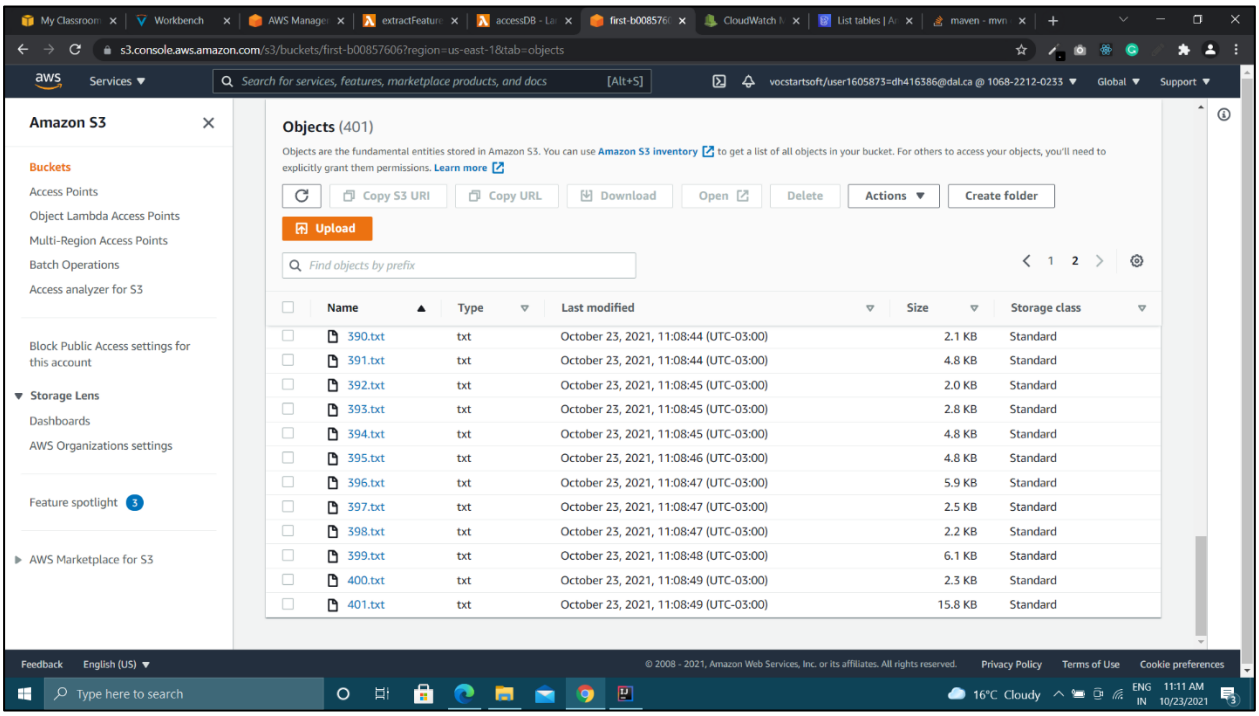

Figure 17 - Files uploaded to "first-b00857606" S3 bucket (contd.) [1]

c. If a file is available on the 1st bucket, then it triggers the extractFeatures Lambda function, which is the 1st lambda function.

Figure 18 displays triggered "extractFeatures" lambda function logs when objects were uploaded to S3.

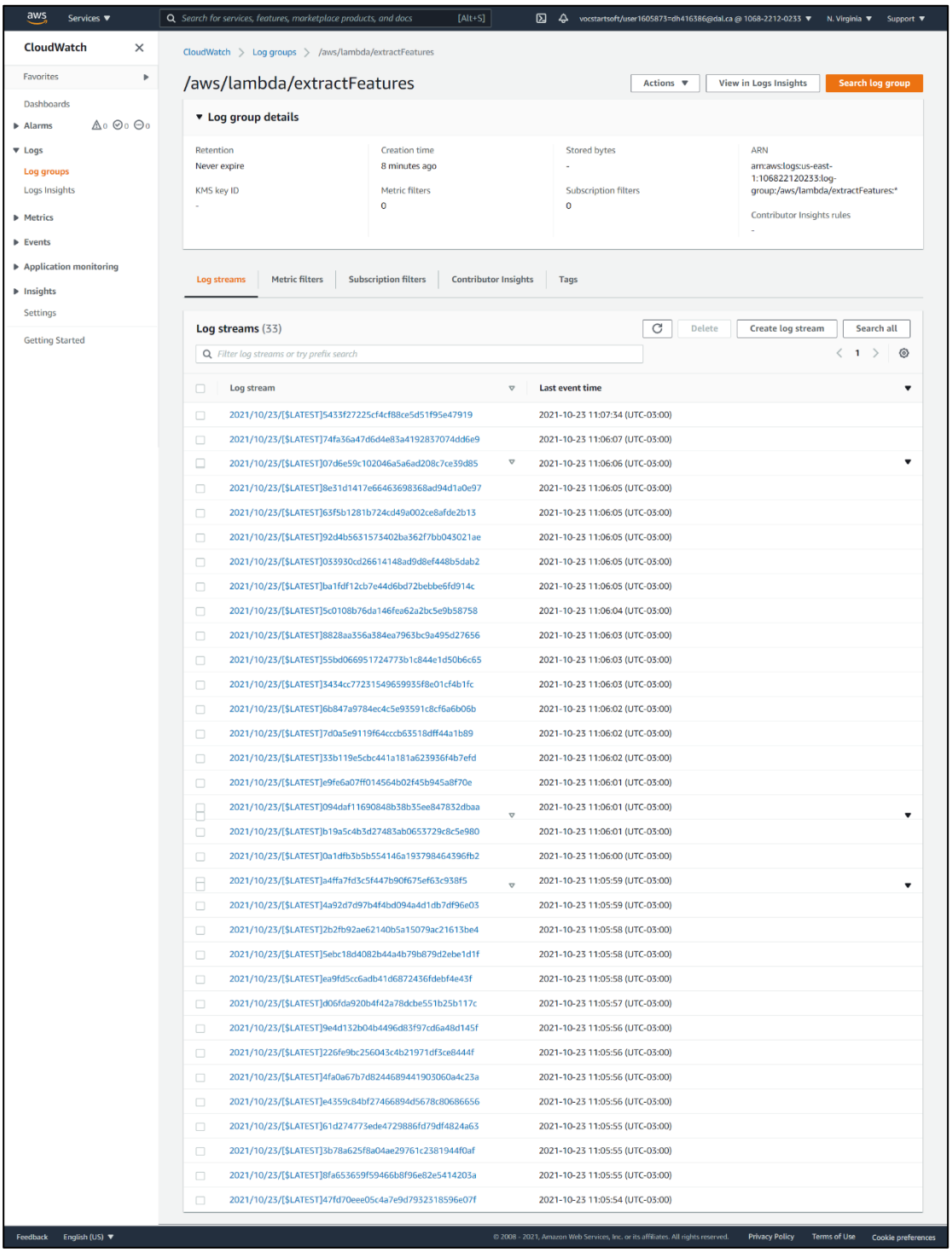

Figure 18 - Triggered "extractFeatures" lambda function logs [4]

Figure 19 displays log messages of a random lambda function. (i.e., when the object is uploaded to “first-b00857606”)

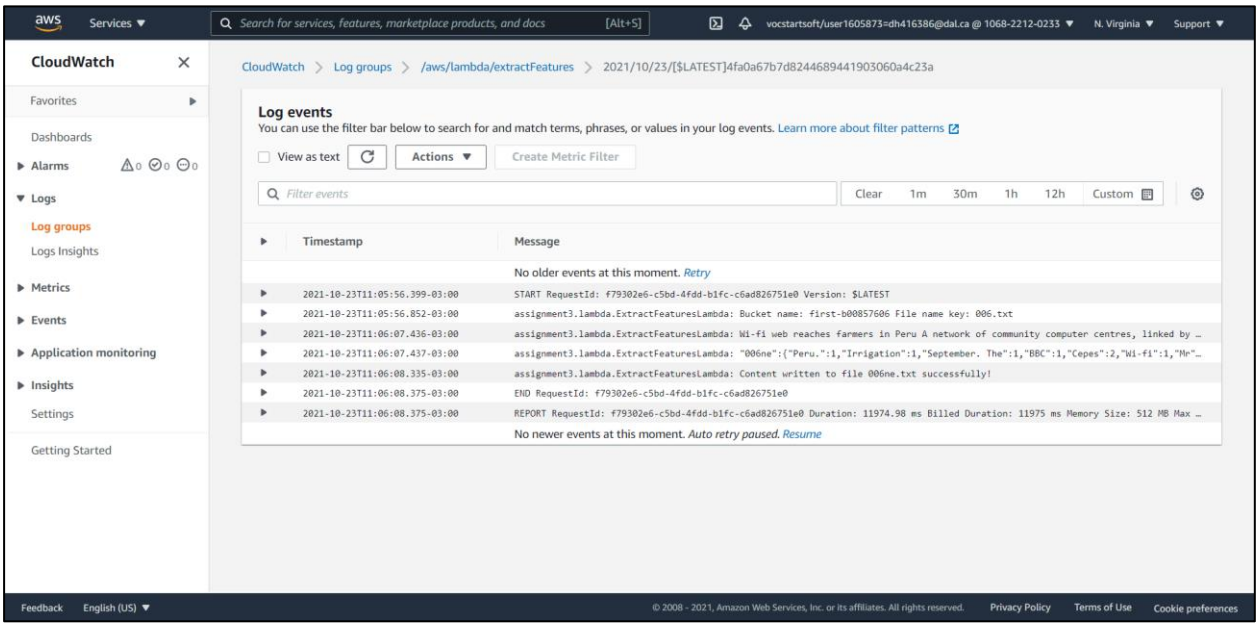

Figure 19 - Random log message (i.e., of file 006.txt when uploaded to "first-b00857606" bucket) [4]

d. This lambda function extracts the Named entities from the file and creates a JSON array of named entities* for that file.

Figure 20 displays a JSON array of Named Entities for a random file in AWS CloudWatch.

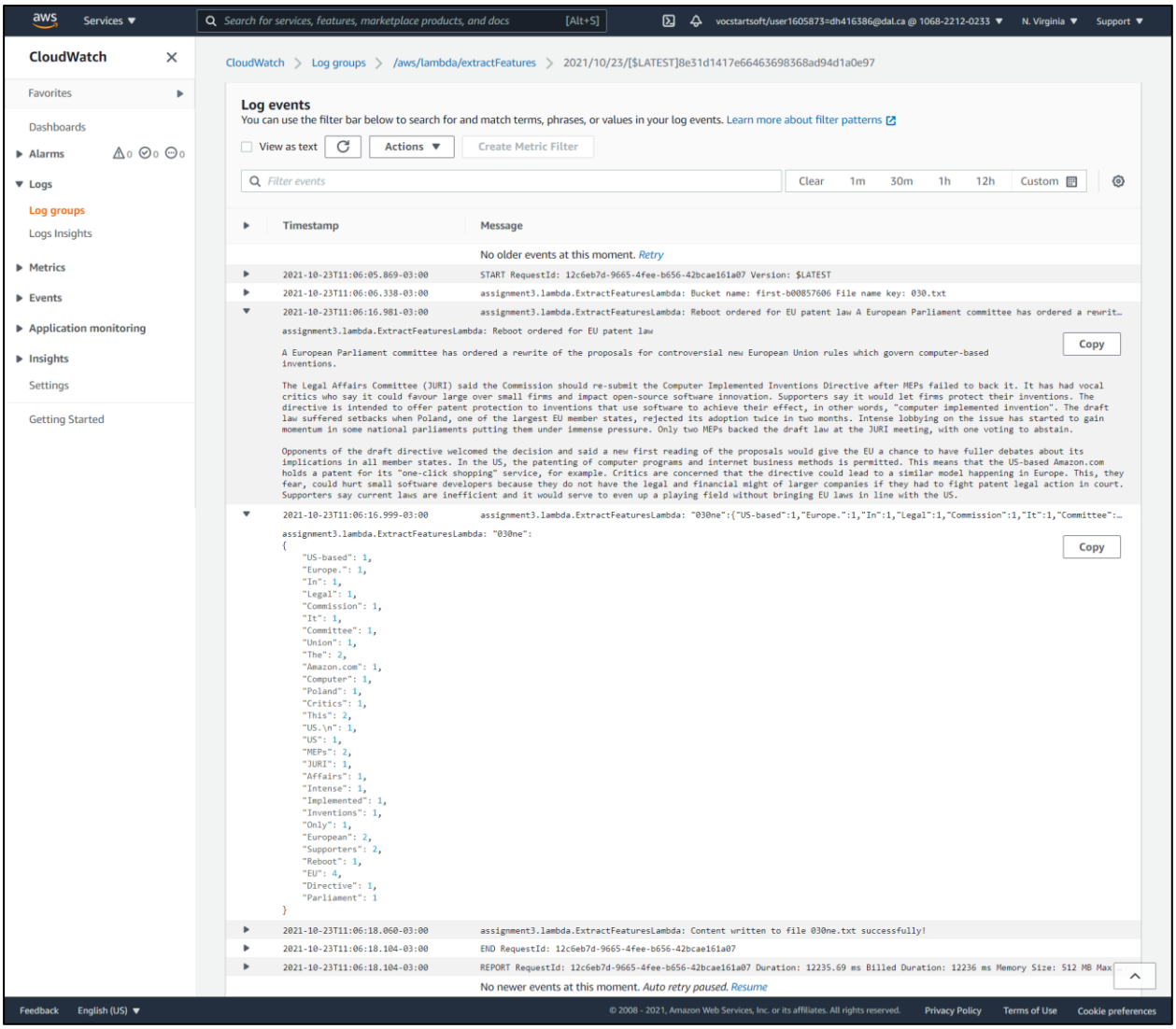

Figure 20 – JSON array of Named Entities [4]

e. E.g., 001.txt contains Asia, Soviet, Serbia etc., then the JSON array created by the function should be "001ne": {"Asia":1, "Soviet":1, etc.…}.

Figures 21 and **22** display JSON array content of random files uploaded to the "second-b00857606" bucket.

Figure 21 - JSON array content of a random file

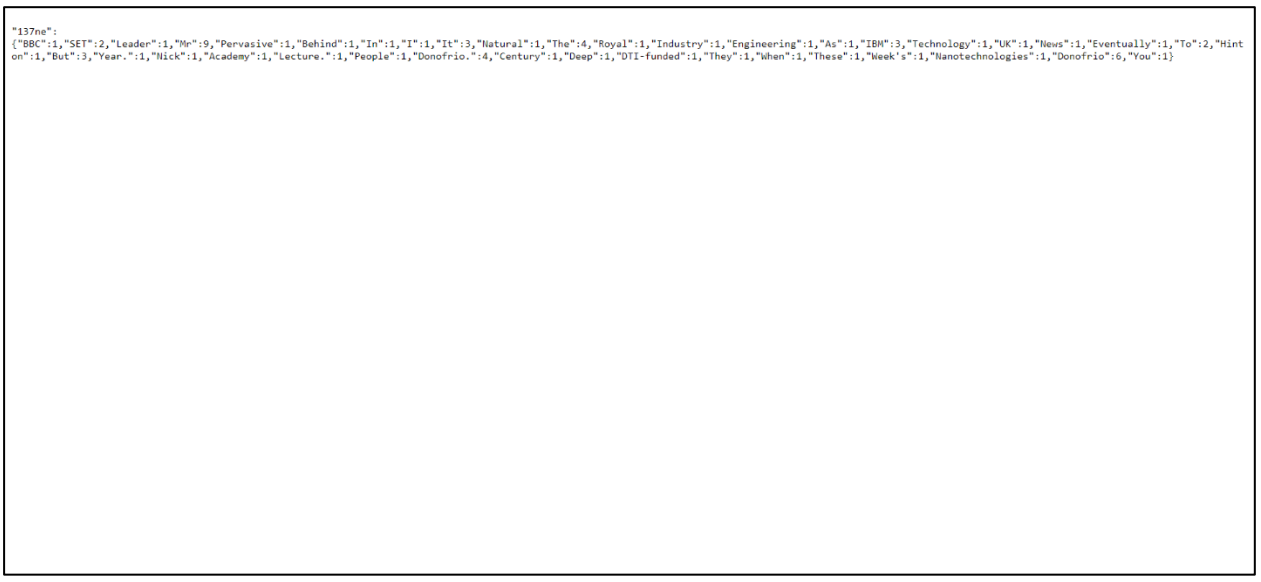

Figure 22 - JSON array content of a random file

f. This file will be saved as 001ne.txt in a new bucket - SecondB00xxxxxx.

Figures 23, 24 and **25** display all files uploaded to the "second-b00857606" S3 bucket.

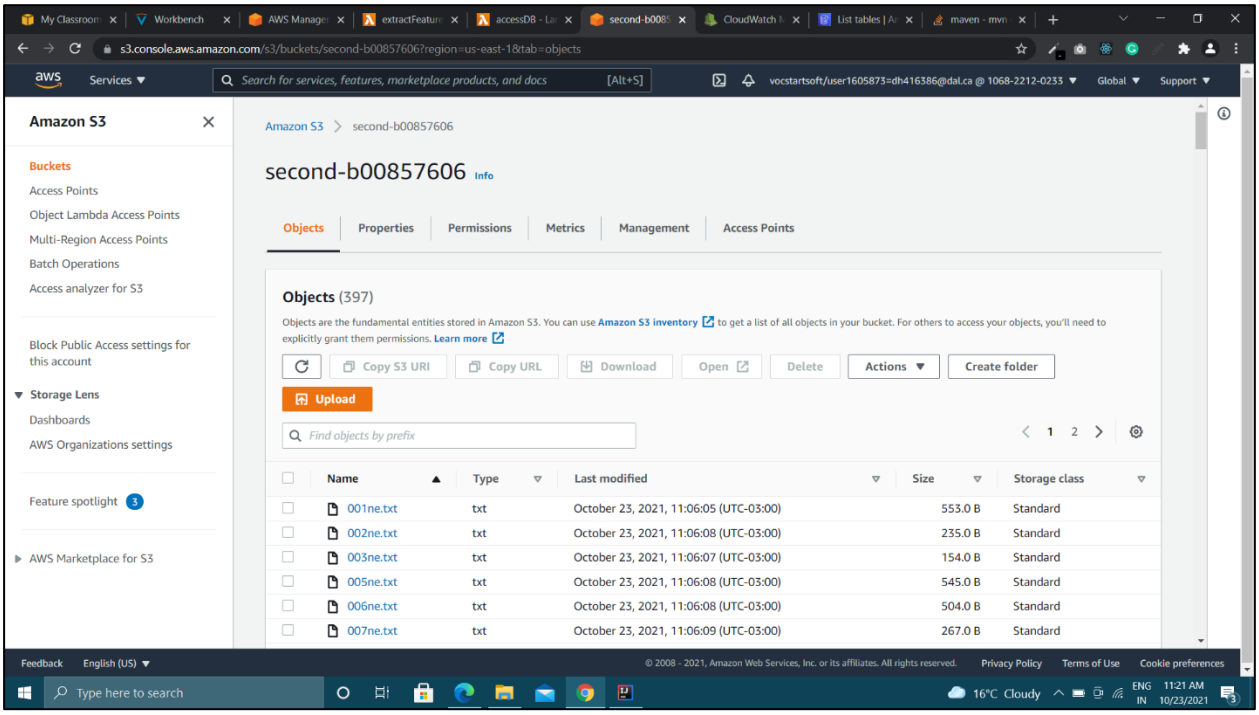

Figure 23 - Files uploaded to "second-b00857606" S3 bucket [1]

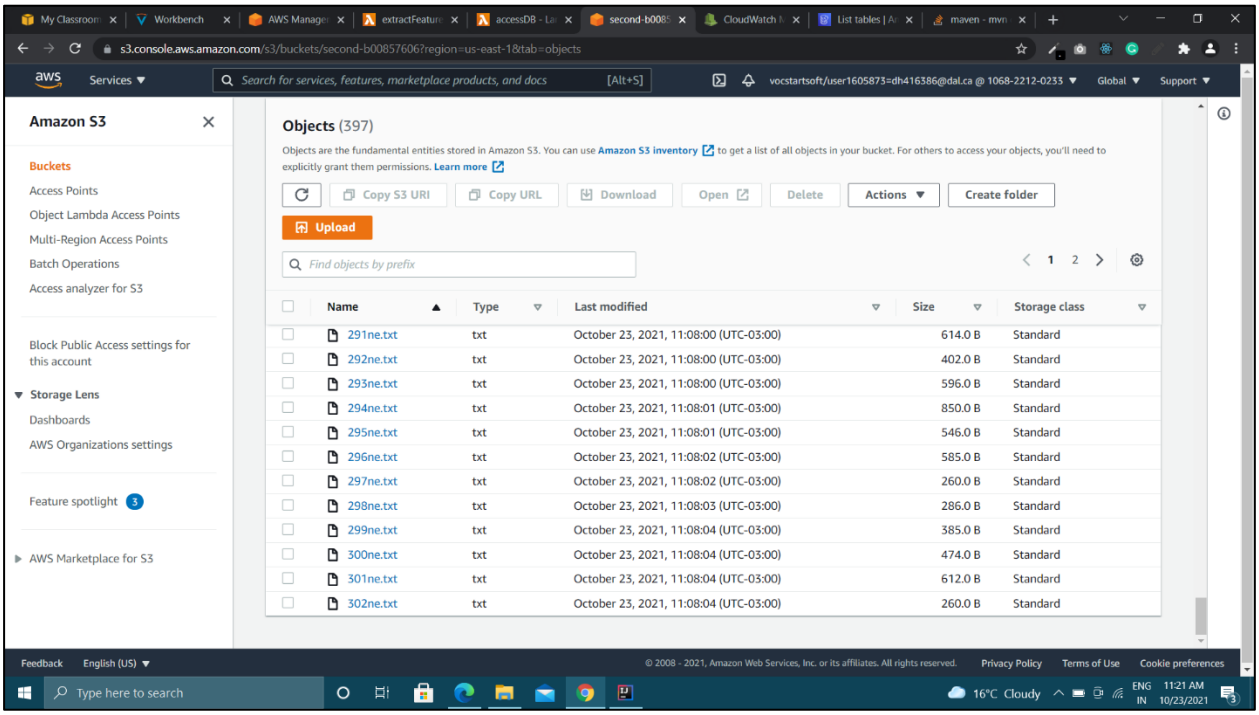

Figure 24 - Files uploaded to "first-b00857606" S3 bucket (contd.) [1]

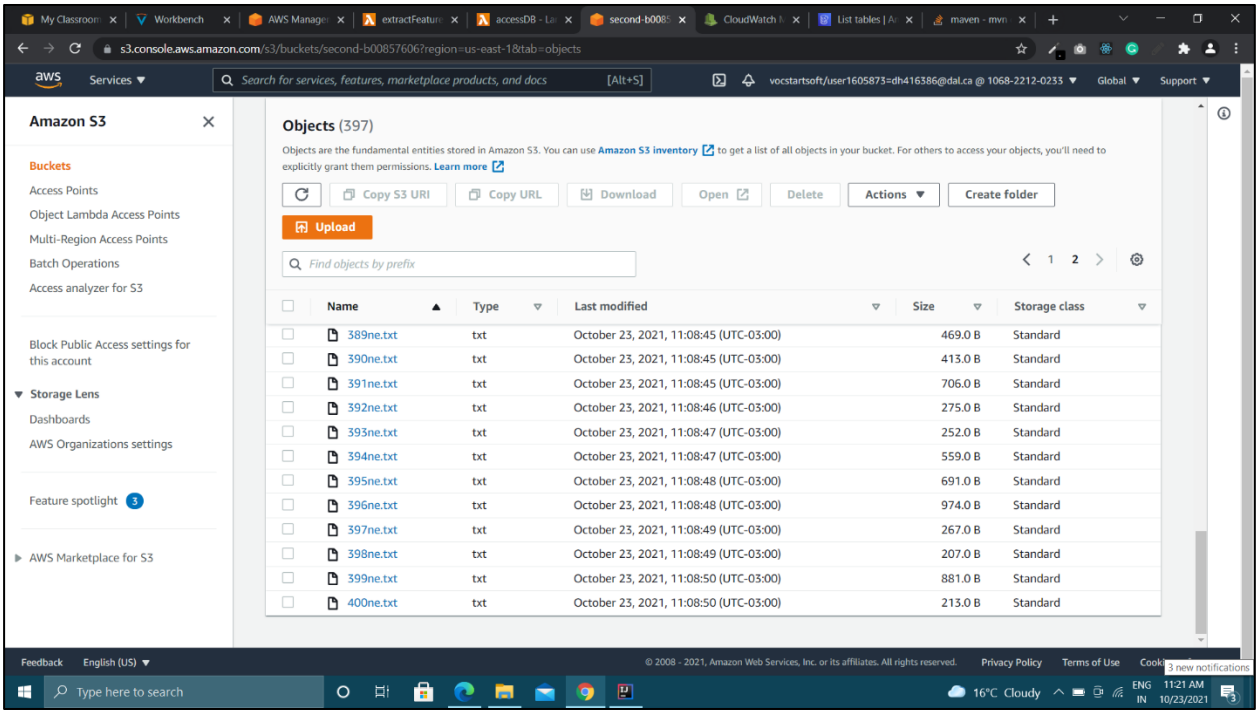

Figure 25 - Files uploaded to "first-b00857606" S3 bucket (contd.) [1]

g. Once the file is available on this 2nd bucket, then the accessDB Lambda function will automatically be triggered.

Figures 26, 27 and **28** display logs in AWS CloudWatch when files are uploaded to the "second-b00857606" bucket.

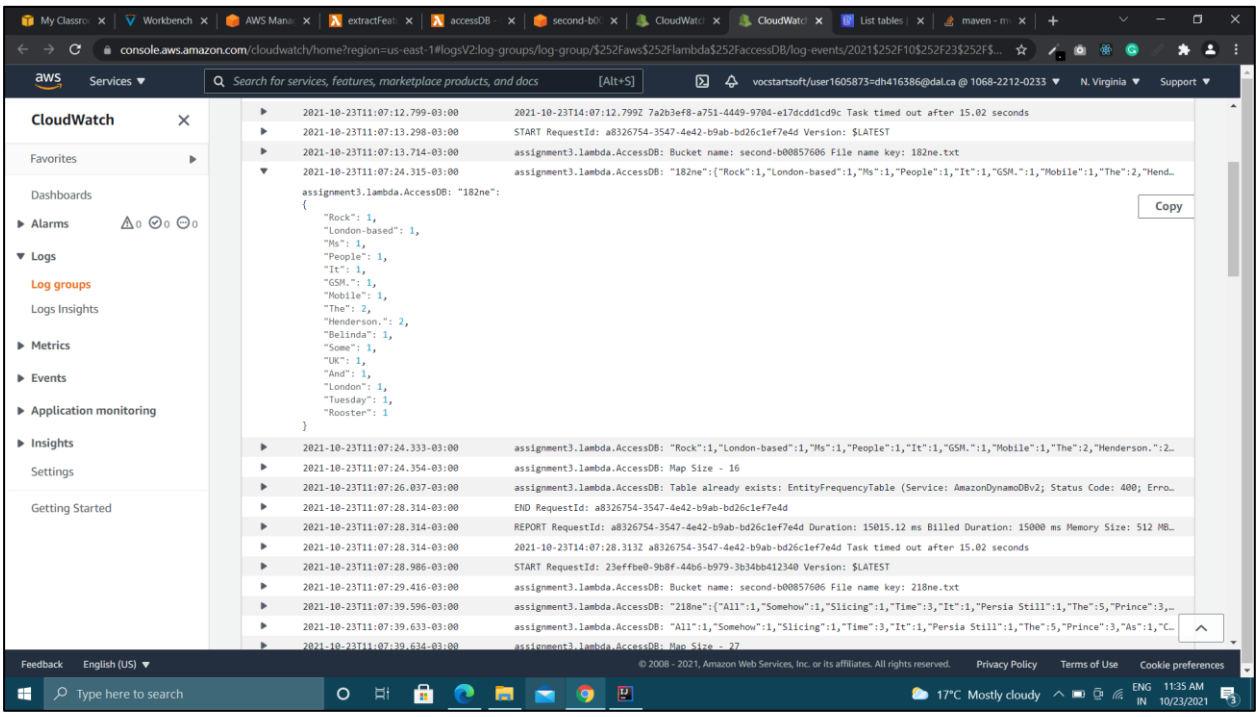

Figure 26 - Triggered "accessDB" lambda function logs when files are uploaded to bucket "second-b00857606" [4]

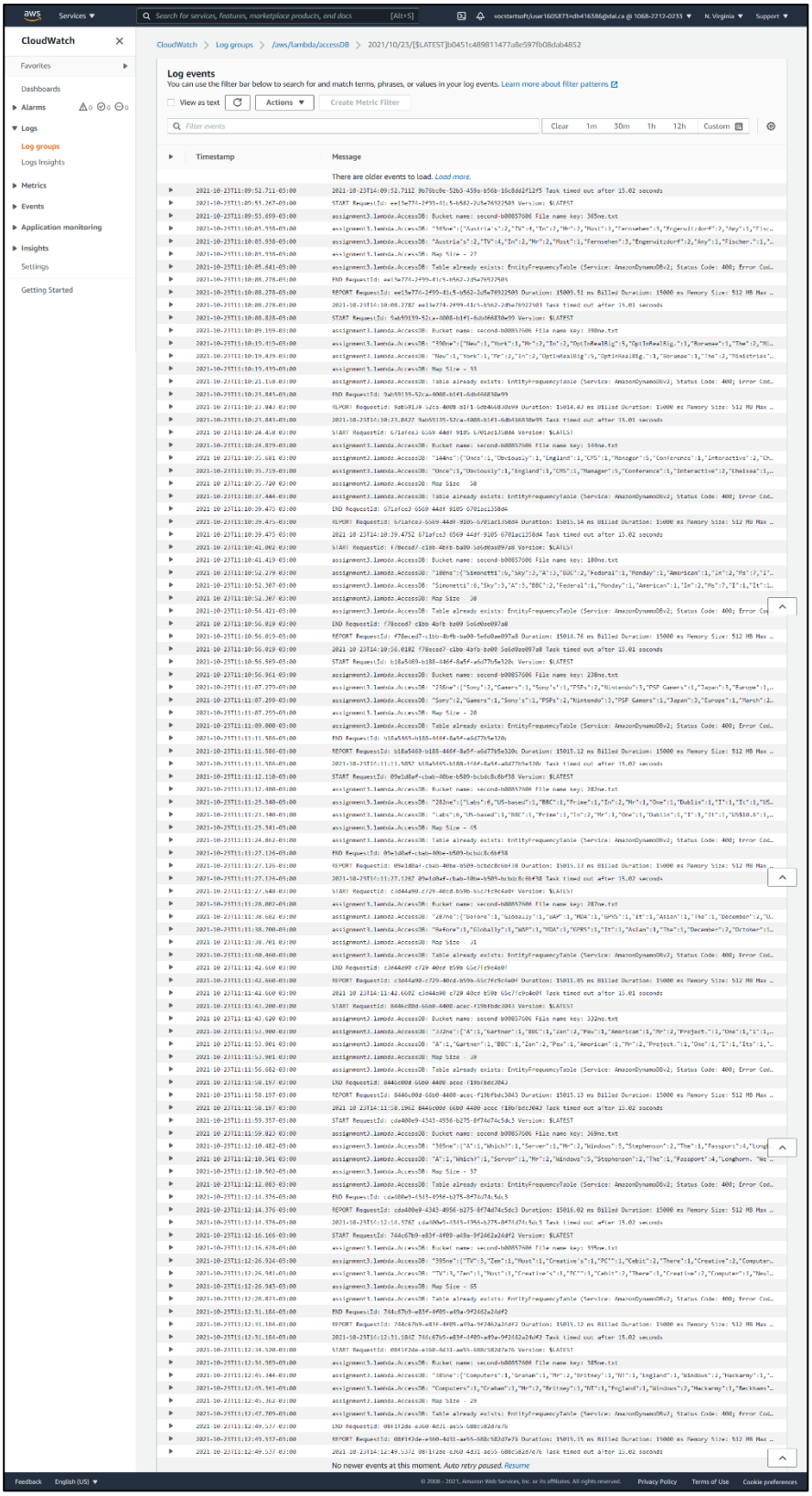

Figure 27 - Triggered "accessDB" lambda function logs when files are uploaded to bucket "second-b00857606" (contd.) [4]

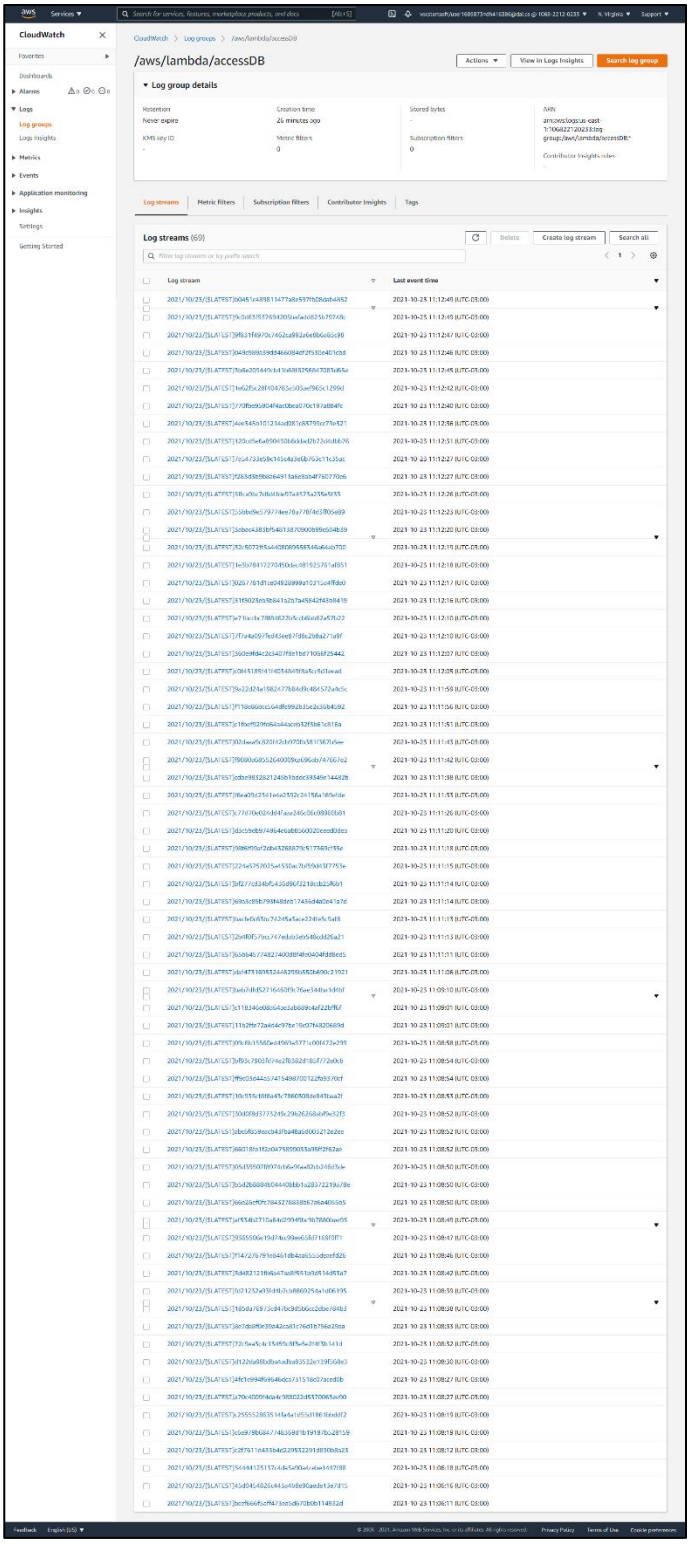

Figure 28 - Triggered "accessDB" lambda function logs when files are uploaded to bucket "second-b00857606" (contd.) [4]

h. accessDB is your 2nd Lambda function. This Lambda function reads each named entity JSON file and updates the DynamoDB database table (three entries/array - NameEntity, Frequency, TimeStamp of Entry).

Figures 29, 27 and **28** display logs in AWS CloudWatch when files are uploaded to the "second-b00857606" bucket.

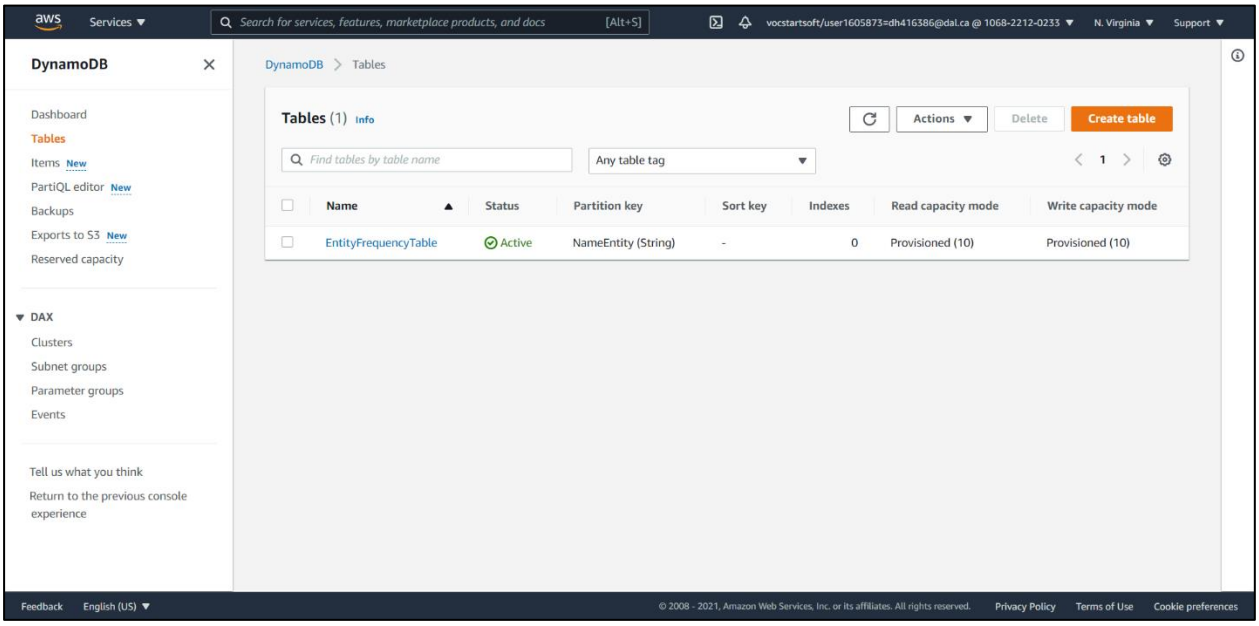

Figure 29 – Table "EntityFrequencyTable" in DynamoDB [5]

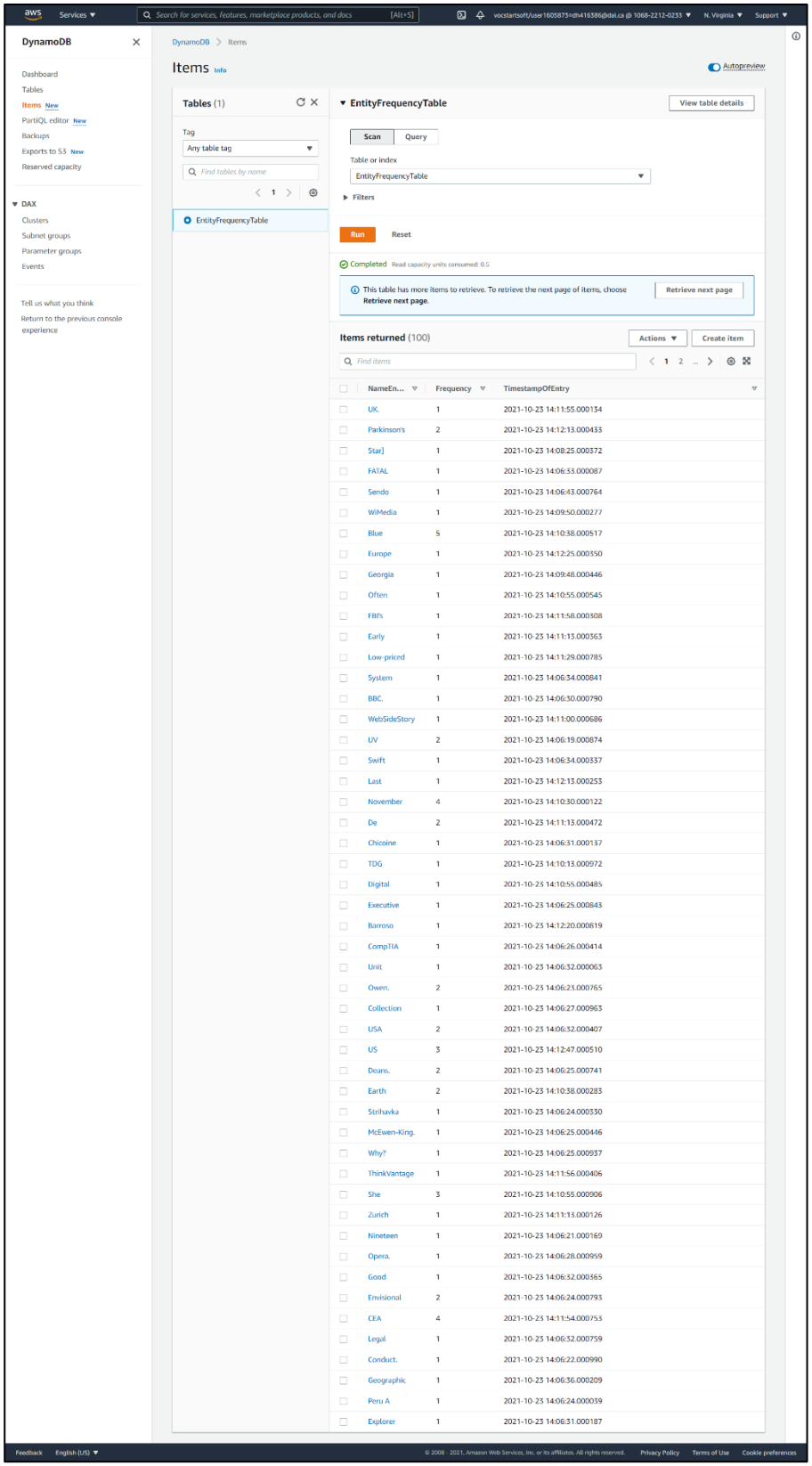

Figure 30 - "EntityFrequencyTable" with entries NameEntity, Frequency, and TimestampOfEntry [5]

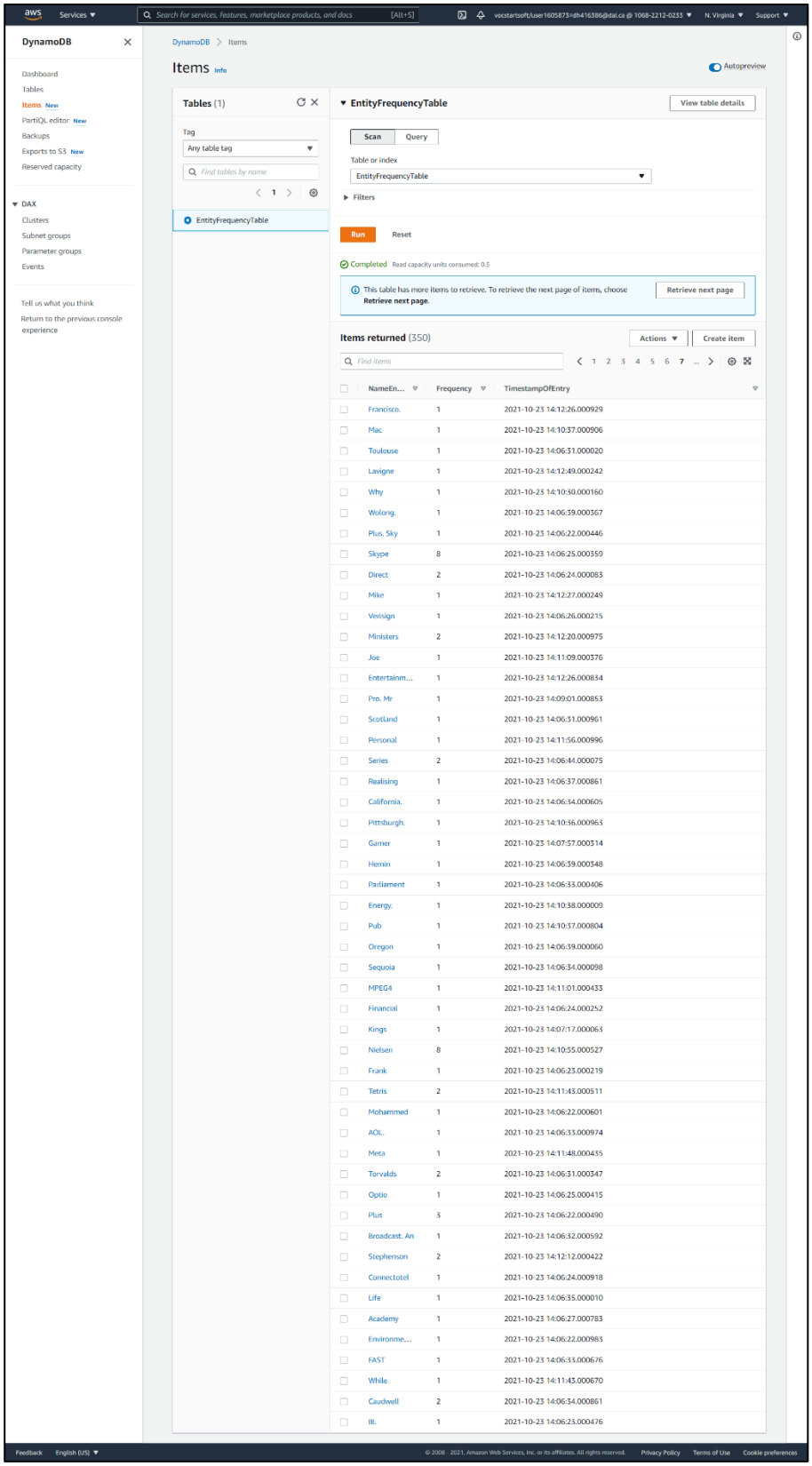

Figure 31 - "EntityFrequencyTable" with entries NameEntity, Frequency, and TimestampOfEntry (contd.) [5]

Program/Scripts

1. **EventDrivenServerlessApplication –** This class contains the core logic for uploading all files in the "tech" folder. The source code is as below:

```
package assignment3.application;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.Bucket;

import java.io.File;
import java.util.List;

/**
 * @author Dhrumil Amish Shah (B00857606 | dh416386@dal.ca)
 */
public class EventDrivenServerlessApplication {
  private static final String AWS_ACCESS_KEY = "<AWS_ACCESS_KEY>";
  private static final String AWS_SECRET_KEY = "<AWS_SECRET_KEY>";
  private static final String AWS_SESSION_TOKEN = "<AWS_SESSION_TOKEN>";
  private static final BasicSessionCredentials AWS_CREDENTIALS = new
BasicSessionCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY, AWS_SESSION_TOKEN);

  private void uploadObject(final AmazonS3 awsS3ClientBuilder,
                            final String bucketName,
                            final File file) {
    final String fileName = file.getName();
    System.out.println("Uploading " + fileName + " to S3 bucket " + bucketName + "...");
    try {
      awsS3ClientBuilder.putObject(bucketName, fileName, file);
      System.out.println("File uploaded successfully.");
    } catch (final AmazonServiceException e) {
      e.printStackTrace();
      System.err.println(e.getMessage());
    }
  }

  private void uploadFilesFromFolderToS3Bucket(final AmazonS3 awsS3ClientBuilder,
                                               final String bucketName,
                                               final String folderName) {
    final File folderPath = new File(folderName);
    final File[] allFiles = folderPath.listFiles();
    if (allFiles != null) {
      for (final File file : allFiles) {
        uploadObject(awsS3ClientBuilder, bucketName, file);
        try {
          Thread.sleep(200);
        } catch (final InterruptedException e) {
          e.printStackTrace();
```

```
        System.err.println(e.getMessage());
      }
    }
  } else {
    System.out.println("No files to upload!");
  }
}

private static Bucket getBucketIfExists(final AmazonS3 awsS3ClientBuilder,
                                        final String bucketName) {
  final List<Bucket> allBuckets = awsS3ClientBuilder.listBuckets();
  for (final Bucket s3Bucket : allBuckets) {
    if (s3Bucket.getName().equals(bucketName)) {
      return s3Bucket;
    }
  }
  return null;
}

private Bucket createBucketIfNotExists(final AmazonS3 awsS3ClientBuilder,
                                       final String bucketName) {
  if (awsS3ClientBuilder.doesBucketExistV2(bucketName)) {
    System.out.println("Bucket " + bucketName + " exists already!");
    return getBucketIfExists(awsS3ClientBuilder, bucketName);
  } else {
    try {
      System.out.println("Creating bucket " + bucketName + "...");
      Bucket bucket = awsS3ClientBuilder.createBucket(bucketName);
      System.out.println("Bucket " + bucketName + " created successfully!");
      return bucket;
    } catch (final AmazonS3Exception e) {
      e.printStackTrace();
      System.err.println(e.getErrorMessage());
    }
  }
  return null;
}

private AmazonS3 createAWSS3ClientBuilder() {
  return AmazonS3ClientBuilder.standard()
      .withCredentials(new AWSStaticCredentialsProvider(AWS_CREDENTIALS))
      .withRegion(Regions.US_EAST_1)
      .build();
}

public void execute(final String folderName) {
  try {
    final String bucket1Name = "first-b00857606";
    final String bucket2Name = "second-b00857606";
    final AmazonS3 awsS3ClientBuilder = createAWSS3ClientBuilder();
    if (awsS3ClientBuilder != null) {
      final Bucket firstBucket = createBucketIfNotExists(awsS3ClientBuilder, bucket1Name);
      final Bucket secondBucket = createBucketIfNotExists(awsS3ClientBuilder, bucket2Name);
      if (firstBucket != null && secondBucket != null) {
        uploadFilesFromFolderToS3Bucket(awsS3ClientBuilder, firstBucket.getName(), folderName);
      } else {
```

```
        System.err.println("Error working with bucket.");
      }
    } else {
      System.err.println("Error creating AWS S3 client builder instance!");
    }
  } catch (final Exception e) {
    e.printStackTrace();
    System.err.println(e.getMessage());
  }
 }
}
```

2. **EventDrivenServerlessApplicationTest –** This class contains the logic to execute **EventDrivenServerlessApplication** class. The source code is as below:

```
package assignment3.application;

/**
 * @author Dhrumil Amish Shah (B00857606 | dh416386@dal.ca)
 */
public class EventDrivenServerlessApplicationTest {
 public static void main(String[] args) {
  final String folderName = "tech";
  final EventDrivenServerlessApplication serverlessApplication = new EventDrivenServerlessApplication();
  serverlessApplication.execute(folderName);
 }
}
```

3. **ExtractFeaturesLambda –** This class contains the logic which is executed when files are uploaded to the first S3 bucket (i.e., “first-b00857606” S3 bucket). “extractFeatures” lambda function logic. The source code is as below:

```
package assignment3.lambda;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import java.net.URLDecoder;
import java.util.HashMap;
import java.util.Map;

/**
 * @author Dhrumil Amish Shah (B00857606 | dh416386@dal.ca)
 */
public class ExtractFeaturesLambda implements RequestHandler<S3Event, String> {
 final static String LOG_TAG = "assignment3.lambda.ExtractFeaturesLambda";
 final static String UTF_8 = "UTF-8";
 final static String OKAY = "okay";
 final static String ERROR = "error";
```

```
 @Override
 public String handleRequest(final S3Event s3Event,
                             final Context context) {
  try {
    final S3EventNotification.S3EventNotificationRecord record = s3Event.getRecords().get(0);

    // Get the bucket name and key for the uploaded S3 object that caused this lambda to be triggered.
    final String bucketName = record.getS3().getBucket().getName();
    final String fileNameKey = URLDecoder.decode(record.getS3().getObject().getKey().replace('+', ' '), UTF_8);
    context.getLogger().log(LOG_TAG + ": Bucket name: " + bucketName + " File name key: " + fileNameKey);

    // Read the source file as text.
    // https://stackoverflow.com/questions/30651502/how-to-get-contents-of-a-text-file-from-aws-s3-using-a-
lambda-function
    final AmazonS3 s3Client = AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
    String body = s3Client.getObjectAsString(bucketName, fileNameKey);
    context.getLogger().log(LOG_TAG + ": " + body);

    // Filter body.
    final String generalFilter = "(\\\\[ntr])";
    final String commaFilter = ",";
    final String bracketFilter = "[{}]";
    body = body.replaceAll(generalFilter, " ");
    body = body.replaceAll(commaFilter, "");
    body = body.replaceAll(bracketFilter, "");

    // Core logic to extract Named Entities.
    final String[] words = body.split(" ");
    final Map<String, Integer> namedEntities = new HashMap<>();
    for (final String currentWord : words) {
      if (Character.isUpperCase(currentWord.charAt(0))) {
        if (namedEntities.containsKey(currentWord)) {
          namedEntities.put(currentWord, namedEntities.get(currentWord) + 1);
        } else {
          namedEntities.put(currentWord, 1);
        }
      }
    }

    // Prepare the JSON string of Named Entities ("001ne": {"Asia":1, "Soviet":1...etc}).
    final StringBuilder jsonNamedEntity = new StringBuilder();
    jsonNamedEntity.append("\"").append(fileNameKey.split("\\.")[0]).append("ne").append("\"");
    jsonNamedEntity.append(":").append("{");
    for (final Map.Entry<String, Integer> namedEntity : namedEntities.entrySet()) {

jsonNamedEntity.append("\"").append(namedEntity.getKey()).append("\"").append(":").append(namedEntity.getVal
ue());
      jsonNamedEntity.append(",");
    }
    jsonNamedEntity.replace(jsonNamedEntity.length() - 1, jsonNamedEntity.length(), "");
    jsonNamedEntity.append("}");
    context.getLogger().log(LOG_TAG + ": " + jsonNamedEntity.toString());

    // Add JSON string to a new file and upload it on S3 in a bucket named "second-b00857606".
    final String newFileName = fileNameKey.split("\\.")[0] + "ne" + ".txt";
```

```
    final String bucket2Name = "second-b00857606";
    s3Client.putObject(bucket2Name, newFileName, jsonNamedEntity.toString());
    context.getLogger().log(LOG_TAG + ": " + "Content written to file " + newFileName + " successfully!");

    return OKAY;
   } catch (final Exception e) {
    context.getLogger().log(LOG_TAG + ": " + e.getMessage());
    return ERROR;
   }
  }
}
```

4. **AccessDBLambda –** This class contains the logic which is executed when files are uploaded to the second S3 bucket (i.e., "second-b00857606" S3 bucket). "accessDB" lambda function logic. The source code is as below:

```
package assignment3.lambda;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.CreateTableResult;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;

import java.net.URLDecoder;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

/**
 * @author Dhrumil Amish Shah (B00857606 | dh416386@dal.ca)
 */
public class AccessDBLambda implements RequestHandler<S3Event, String> {
  final static String LOG_TAG = "assignment3.lambda.AccessDB";
  final static String UTF_8 = "UTF-8";
  final static String OKAY = "okay";
  final static String ERROR = "error";

  @Override
  public String handleRequest(final S3Event s3Event,
```

```
                        final Context context) {
    try {
      final S3EventNotification.S3EventNotificationRecord record = s3Event.getRecords().get(0);

      // Get the bucket name and key for the uploaded S3 object that caused this lambda to be triggered.
      final String bucketName = record.getS3().getBucket().getName();
      final String fileNameKey = URLDecoder.decode(record.getS3().getObject().getKey().replace('+', ' '), UTF_8);
      context.getLogger().log(LOG_TAG + ": Bucket name: " + bucketName + " File name key: " + fileNameKey);

      // Read the source file as text.
      final AmazonS3 s3Client = AmazonS3ClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
      final String body = s3Client.getObjectAsString(bucketName, fileNameKey);
      context.getLogger().log(LOG_TAG + ": " + body);

      // Extract JSON response.
      final String startingBracketFilter = "\\{";
      final String endingBracketFilter = "}";
      final String namedEntitiesString = body.split(startingBracketFilter)[1].replaceAll(endingBracketFilter, "");
      context.getLogger().log(LOG_TAG + ": " + namedEntitiesString);

      // Prepare named entities map.
      final Map<String, Integer> namedEntitiesMap = new HashMap<>();
      final String[] namedEntitiesArray = namedEntitiesString.split(",");
      for (String namedEntity : namedEntitiesArray) {
        final String namedEntityKey = namedEntity.split(":")[0].replaceAll("\"", "");
        final Integer namedEntityVal = Integer.parseInt(namedEntity.split(":")[1]);
        namedEntitiesMap.put(namedEntityKey, namedEntityVal);
      }
      context.getLogger().log(LOG_TAG + ": Map Size - " + namedEntitiesMap.size());

      // Prepare AmazonDynamoDB client builder.
      final AmazonDynamoDB amazonDynamoDB =
AmazonDynamoDBClientBuilder.standard().withRegion(Regions.US_EAST_1).build();

      // DynamoDB entities.
      final String entityFrequencyTable = "EntityFrequencyTable";
      final String nameEntityCol = "NameEntity";
      final String frequencyCol = "Frequency";
      final String timestampOfEntryCol = "TimestampOfEntry";

      // Create table if not exists already.
      try {
        final CreateTableResult result = amazonDynamoDB.createTable(new CreateTableRequest()
            .withAttributeDefinitions(new AttributeDefinition(nameEntityCol, ScalarAttributeType.S))
            .withKeySchema(new KeySchemaElement(nameEntityCol, KeyType.HASH))
            .withProvisionedThroughput(new ProvisionedThroughput(10L, 10L))
            .withTableName(entityFrequencyTable));
        context.getLogger().log(LOG_TAG + ": Table name - " + result.getTableDescription().getTableName());
      } catch (final AmazonServiceException e) {
        context.getLogger().log(LOG_TAG + ": " + e.getMessage());
      }

      // Put values in table.
      for (final Map.Entry<String, Integer> nameEntity : namedEntitiesMap.entrySet()) {
        final Map<String, AttributeValue> items = new HashMap<>();
        items.put(nameEntityCol, new AttributeValue(nameEntity.getKey()));
```

```
      items.put(frequencyCol, new AttributeValue(String.valueOf(nameEntity.getValue())));
      items.put(timestampOfEntryCol, new AttributeValue(new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss.SSSSSS")
          .format(new Date())));

      try {
        amazonDynamoDB.putItem(entityFrequencyTable, items);
      } catch (final ResourceNotFoundException e) {
        context.getLogger().log(LOG_TAG + ": Error: The table " + entityFrequencyTable + " can't be found.");
        context.getLogger().log(LOG_TAG + ": " + e.getMessage());
      } catch (final AmazonServiceException e) {
        context.getLogger().log(LOG_TAG + ": " + e.getMessage());
      }
    }
    return OKAY;
  } catch (final Exception e) {
    context.getLogger().log(LOG_TAG + ": " + e.getMessage());
    context.getLogger().log(LOG_TAG + ": " + e.toString());
    return ERROR;
  }
 }
}
```

References

[1] AWS, "Amazon S3," Amazon, [Online]. Available: https://aws.amazon.com/s3/. [Accessed 22 October 2021].

[2] AWS, "AWS Identity and Access Management (IAM)," Amazon, [Online]. Available: https://aws.amazon.com/iam/. [Accessed 22 October 2021].

[3] AWS, "AWS Lambda," Amazon, [Online]. Available: https://aws.amazon.com/lambda/. [Accessed 22 October 2021].

[4] AWS, "Amazon CloudWatch," Amazon, [Online]. Available: https://aws.amazon.com/cloudwatch/. [Accessed 22 October 2021].

[5] AWS, "Amazon DynamoDB," Amazon, [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed 22 October 2021].

[6] D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.