SCI 5410
Serverless Data Processing

Assignment 3
Part B
Event-driven serverless application using AWS lambda

Ritva Katrodiya
B00930131

Git Repository Link: [Files · A3 · Ritva Ashvinbhai Katrodiya / CSCI5410-SUMMER-23-B00930131 · GitLab (dal.ca)]

Table of Contents

Table of Figures 3
Creation of DynamoDB table using SDK 6
S3 Bucket creation 9
Lambda fucntion creation 12
Notification trigger of lambda and S3 bucket 16
Files uploaded to sampledatab00930131 bucket using Java SDK 22
Program Script 46
Test Cases 56
References 61

Table of Figures

Figure 1 AWS DynamoDB dashboard [1] 6
Figure 2 Existing Tables on DynamoDB [1] 6
Figure 3 Table named **CSCI5410_Ass3_Part2** with primary attribute **key** Created using Java SDK [5] 7
Figure 4 Table named **CSCI5410_Ass3_Part2** created successfully on DynamoDB [1] 8
Figure 5 There is no data in table **CSCI5410_Ass3_Part2 [1]** 8
Figure 6 AWS S3 service dashboard [2] 9
Figure 7 There is not any created bucket [2] 9
Figure 8 Two buckets named **sampledatab00930131** and **tagsb00930131** created using Java SDK [5] 10
Figure 9 **sampledatab00930131** and **tagsb00930131** created successfully [2] 11
Figure 10 Creation of lambda function named **extractFeatures** [3] 12
Figure 11 Creation of lambda function named **extractFeatures [3]** 12
Figure 12 Lambda function named **extractFeatures** created sucessfully [3] 13
Figure 13 Created lambda function named **extractFeatures** is listed [3] 13
Figure 14 Creation of lambda function named **accessDB** [3] 14
Figure 15 Creation of lambda function named **accessDB** [3] 14
Figure 16 Lambda function named **accessDB** created successfully [3] 15
Figure 17 Created lambda function named **accessDB** is listed [3] 15
Figure 18 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2] 16
Figure 19 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2] 17
Figure 20 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2] 17
Figure 21 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2] 18
Figure 22 Lambda **extractFeatures** is associated with notification trigger of S3 bucket named **sampledatab00930131** [2] 18
Figure 23 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2] 19
Figure 24 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2] 19
Figure 25 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2] 20
Figure 26 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2] 20
Figure 27 Lambda **accessDB** is associated with notification trigger of S3 bucket named **tagsb00930131** [2] 21
Figure 28 File named **001.txt** uploaded successfully to **sampledatab00930131** using Java SDK [5] 22
Figure 29 **001.txt** file uploaded to **sampeldatab00930131** successfully [2] 23
Figure 30 Execution logs of lambda **extractFeatures** to extract name entity of **001.txt** file from bucket sampeldatab00930131 [4] 24

Figure 31 **001ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [3] 25
Figure 32 store key value from **001ne.txt** file to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1] 26
Figure 33 Execution logs of lambda **accessDB** to store the key value of **001ne.txt** file and store it to DynamoDB table [4] 26
Figure 34 File named 002.txt uploaded successfully to sampledatab00930131 using Java SDK [5] 27
Figure 35 **002.txt** file uploaded to **sampeldatab00930131** successfully [2] 28
Figure 36 Execution logs of lambda **extractFeatures** to extract name entity of **002.txt** file from bucket **sampeldatab00930131** [4] 29
Figure 37 **002ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [2] 30
Figure 38 store key value from **002ne.txt** file to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1] 31
Figure 39 Execution logs of lambda **accessDB** to get the key value of **002ne.txt** file and store it to DynamoDB table [4] 31
Figure 40 File named **003.txt** uploaded successfully to **sampledatab00930131** using Java SDK [5] 32
Figure 41 **003.txt** file uploaded to **sampeldatab00930131** successfully [2] 33
Figure 42 Execution logs of lambda **extractFeatures** to extract name entity of **003.txt** file from bucket **sampeldatab00930131** [4] 34
Figure 43 **003ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [2] 35
Figure 44 store key value from **003ne.txt file** to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1] 36
Figure 45 Execution logs of lambda **accessDB** to get the key value from **003ne.txt** file and store it to DynamoDB table [4] 36
Figure 46 File named **004.txt** uploaded successfully to **sampledatab00930131** using Java SDK [5] 37
Figure 47 **004.txt** file uploaded to **sampeldatab00930131** successfully [2] 38
Figure 48 Execution logs of lambda **extractFeatures** to extract name entity of **004.txt** file from bucket **sampeldatab00930131** [4] 39
Figure 49 **004ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [2] 40
Figure 50 store key value from **004ne.txt file** to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1] 41
Figure 51 Execution logs of lambda **accessDB** to get the key value from **004ne.txt** file and store it to DynamoDB table [4] 41
Figure 52 All files are uploaded to **samplebuacketb00930131** to extract the name entities [3]. 42
Figure 53 All files are uploaded to **samplebuacketb00930131** to extract the name entities [3]. 42
Figure 54 Lambda **accessDB** triggered, created all the files form **001ne.txt to 400ne.txt** and uploaded to **tagsb00930131** bucket [3] 43
Figure 55 Lambda **accessDB** triggered, created all the files form **001ne.txt** to **400ne.txt** and uploaded to **tagsb00930131** bucket [3] 43
Figure 56 DynamoDB table with all key value stored using **accessDB** lambda [1] 44

Figure 57 DynamoDB table with all key value stored using **accessDB** lambda [1]..... 44
Figure 58 DynamoDB table with all key value stored using **accessDB** lambda [1]..... 45
Figure 59 **File 003.txt** uploaded to **sampledatab00930131** bucket [2] 56
Figure 60 test event configured to test the lambda **extractFeatures [3]**..... 57
Figure 61 lambda **extarctFeatures** tested with 200 status code [3]..... 57
Figure 62 On successfully execution of lambda, **003ne.txt** file is created and uploaded to bucket **tagsb00930131** [2]..... 58
Figure 63 test event configured for second lambda **accessDB** [3] 58
Figure 64 lambda **accessDB** tested with 200 status code [3]..... 59
Figure 65 On successful execution of lambda **accessDB**, key value pair is stored in **DynamoDB** [1]..... 59
Figure 66 Output file named 003be.txt 60

Creation of DynamoDB table using SDK

Figure 1 shows the dashboard of AWS DynamoDB service and Figure 2 shows the existing tables on DynamoDB.

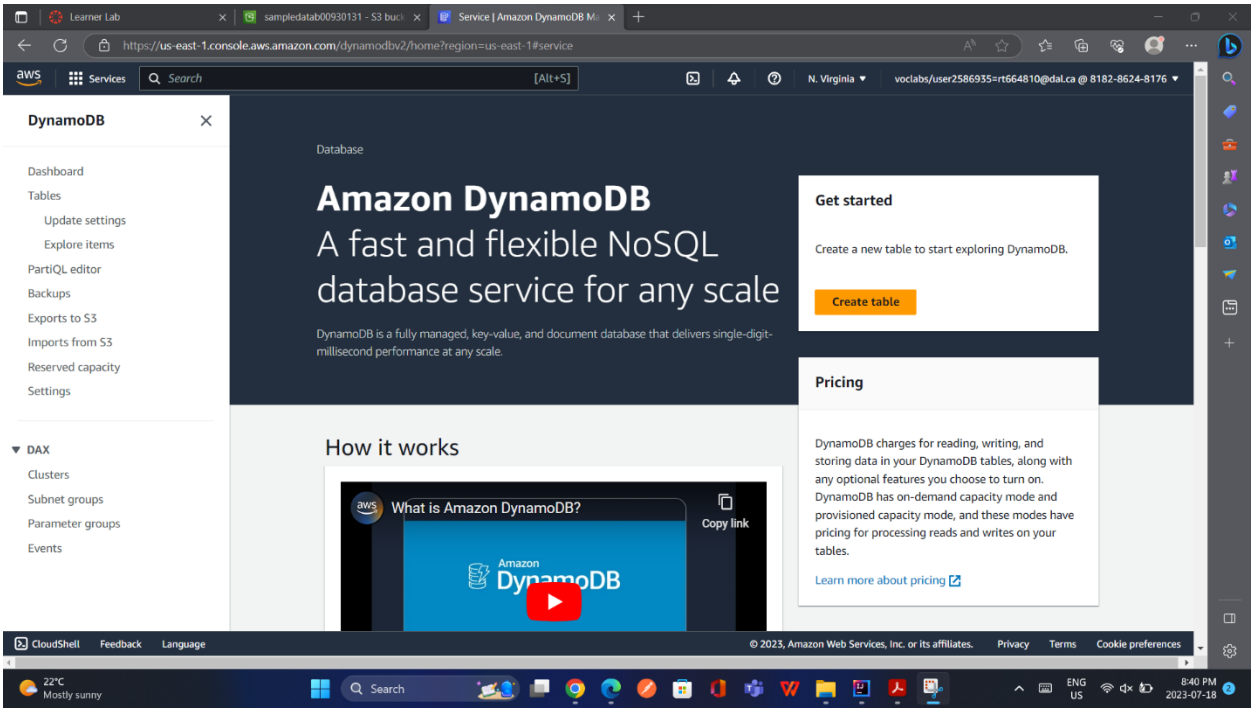

Figure 1 AWS DynamoDB dashboard [1]

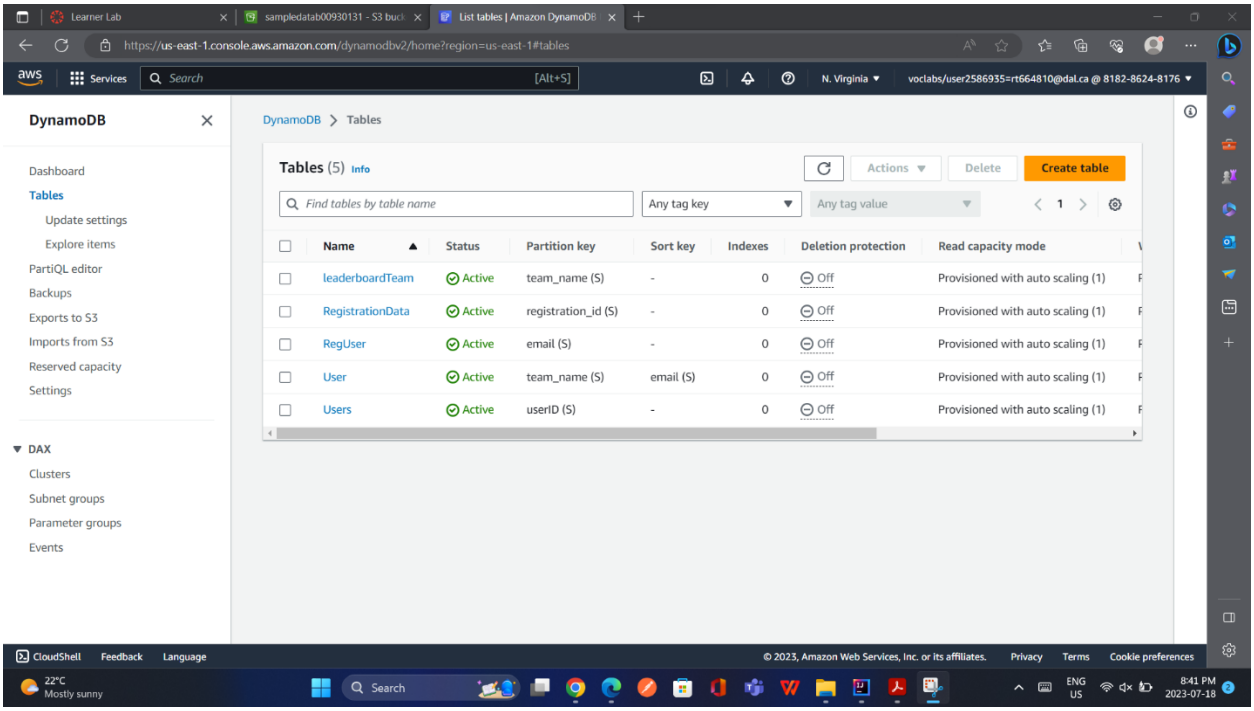

Figure 2 Existing Tables on DynamoDB [1]

Figure 3 defines that DynamoDB table named **CSCI5410_Ass3_Part2** created and given **key** as primary attributes using java SDK. And logs say **CSCI5410_Ass3_Part2** created successfully.

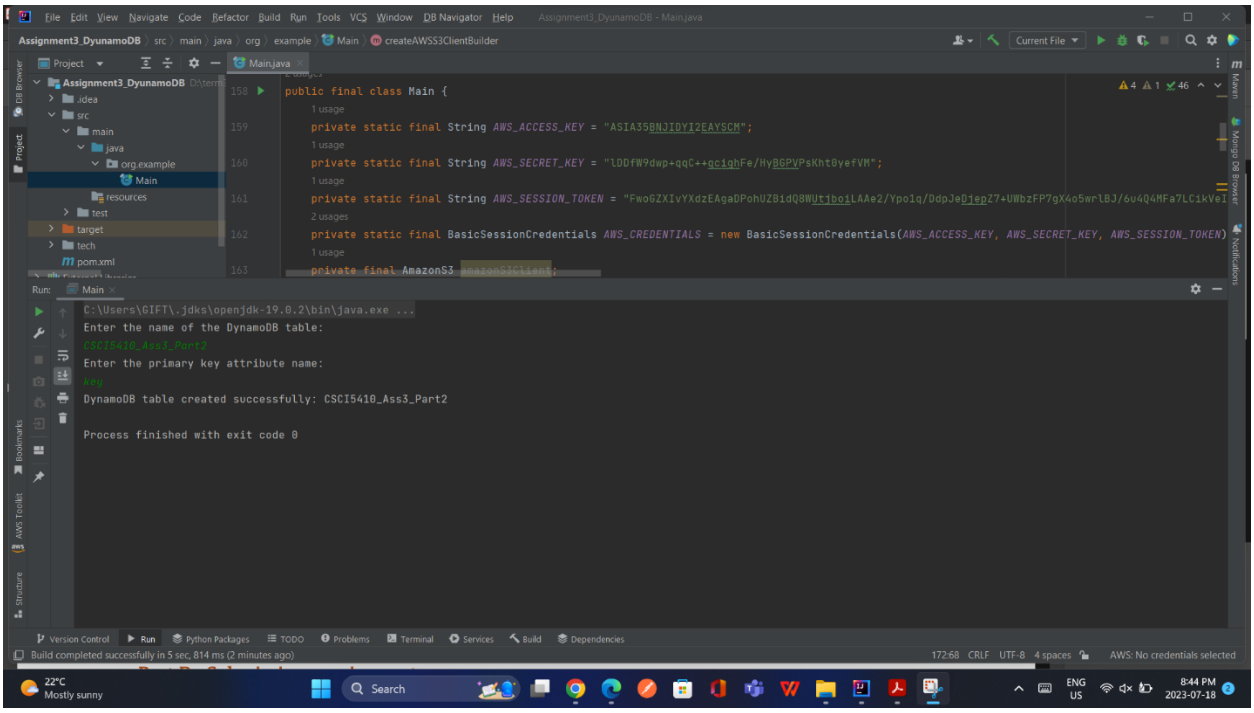

Figure 3 Table named **CSCI5410_Ass3_Part2** with primary attribute **key** Created using Java SDK [5]

Figure 4 shows that table named **CSCI5410_Ass3_Part2** created successfully and listed in DynamoDB tables.

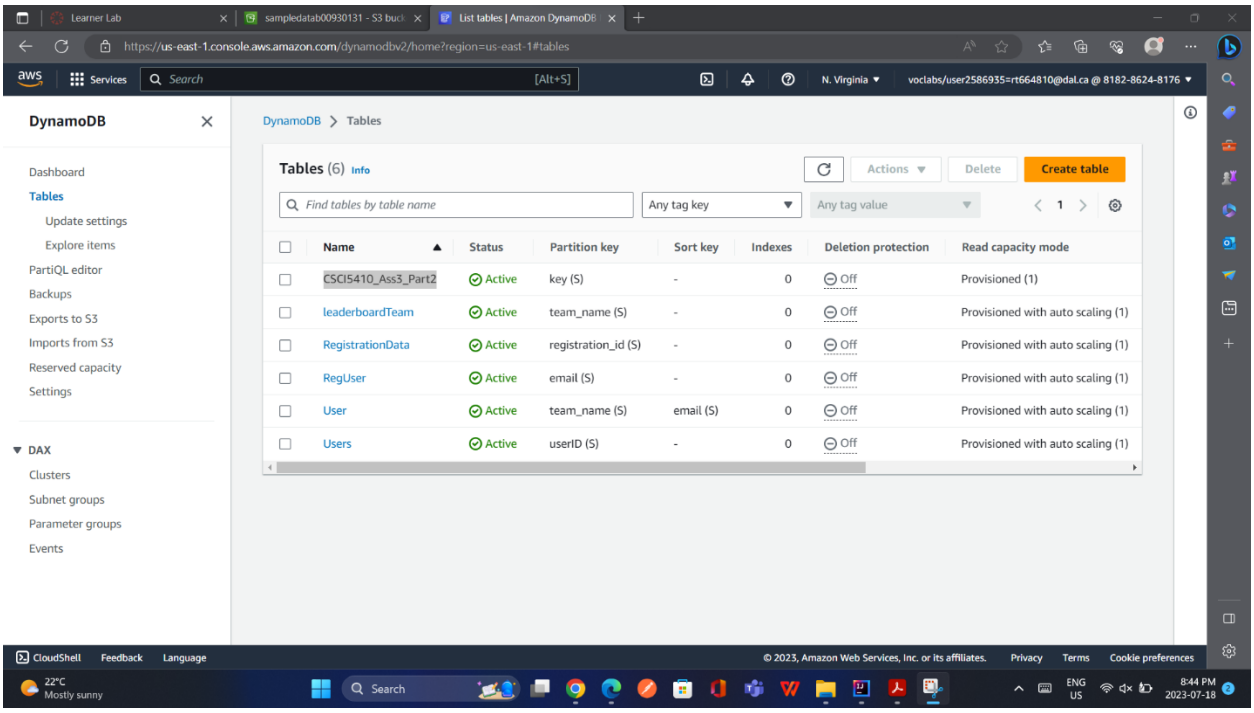

Figure 4 Table named **CSCI5410_Ass3_Part2** created successfully on DynamoDB [1]

Figure 5 shows that table **CSCI5410_Ass3_Part2** is empty, there is not any data.

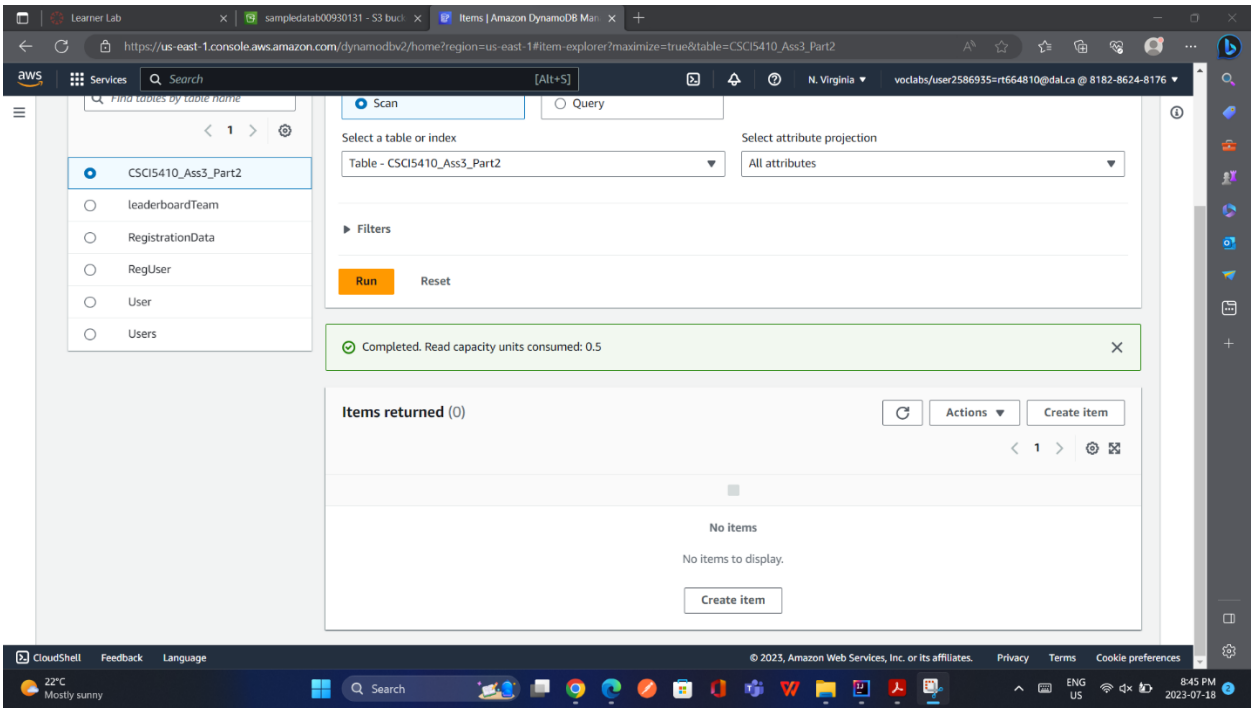

Figure 5 There is no data in table **CSCI5410_Ass3_Part2 [1]**

S3 Bucket creation

Figure 6 shows the dashboard of AWS S3 service and figure 7 defined that there is not any created bucket in S3 buckets.

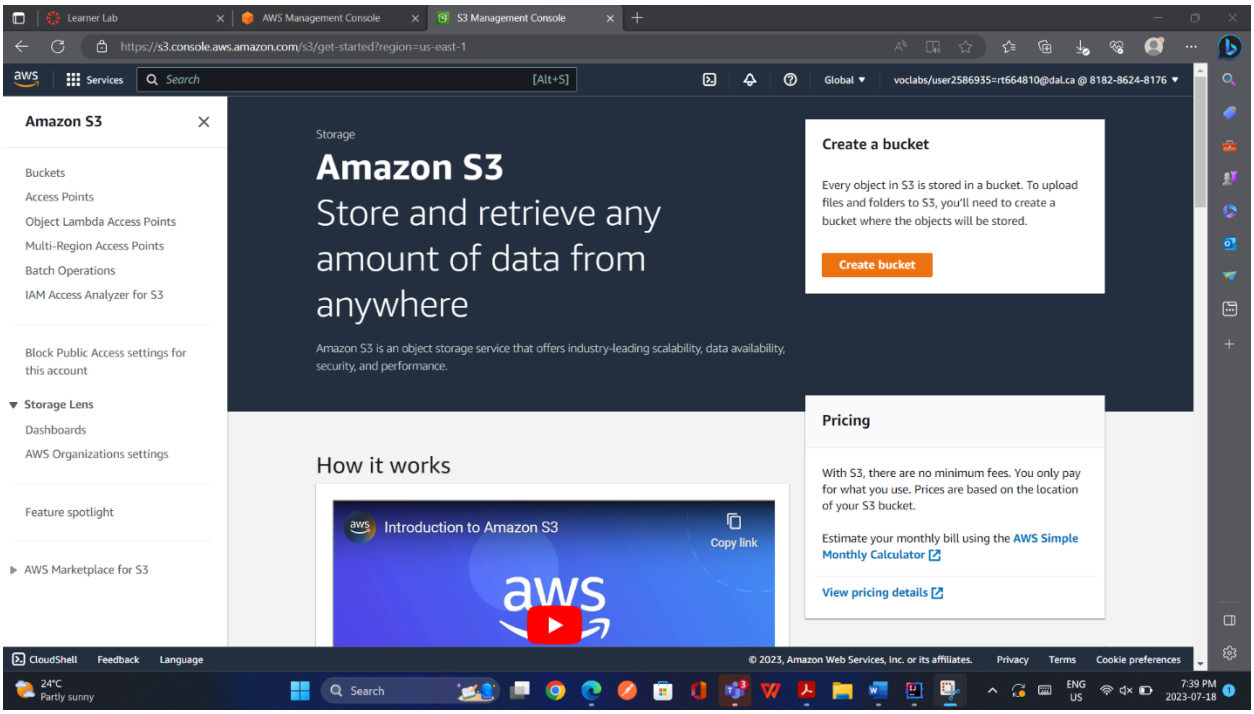

Figure 6 AWS S3 service dashboard [2]

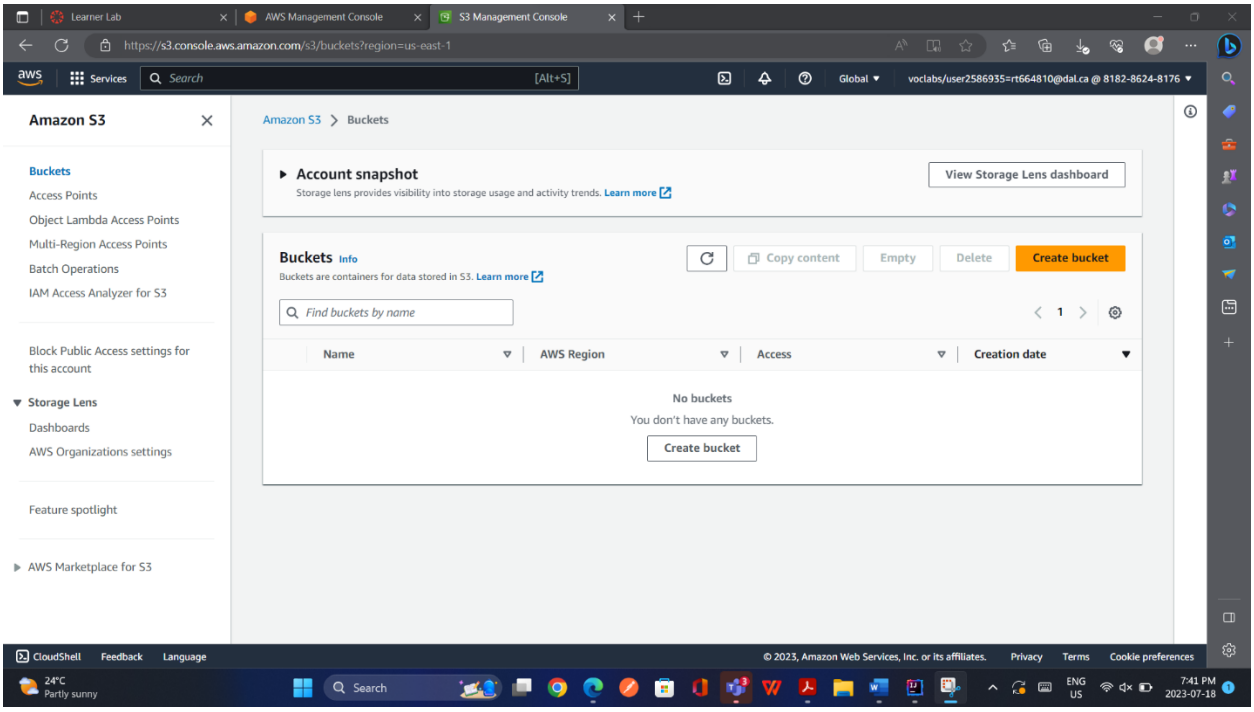

Figure 7 There is not any created bucket [2]

Figure 8 shows that two buckets named **sampledatab00930131** and **tagsb00930131** created successfully using java SDK.

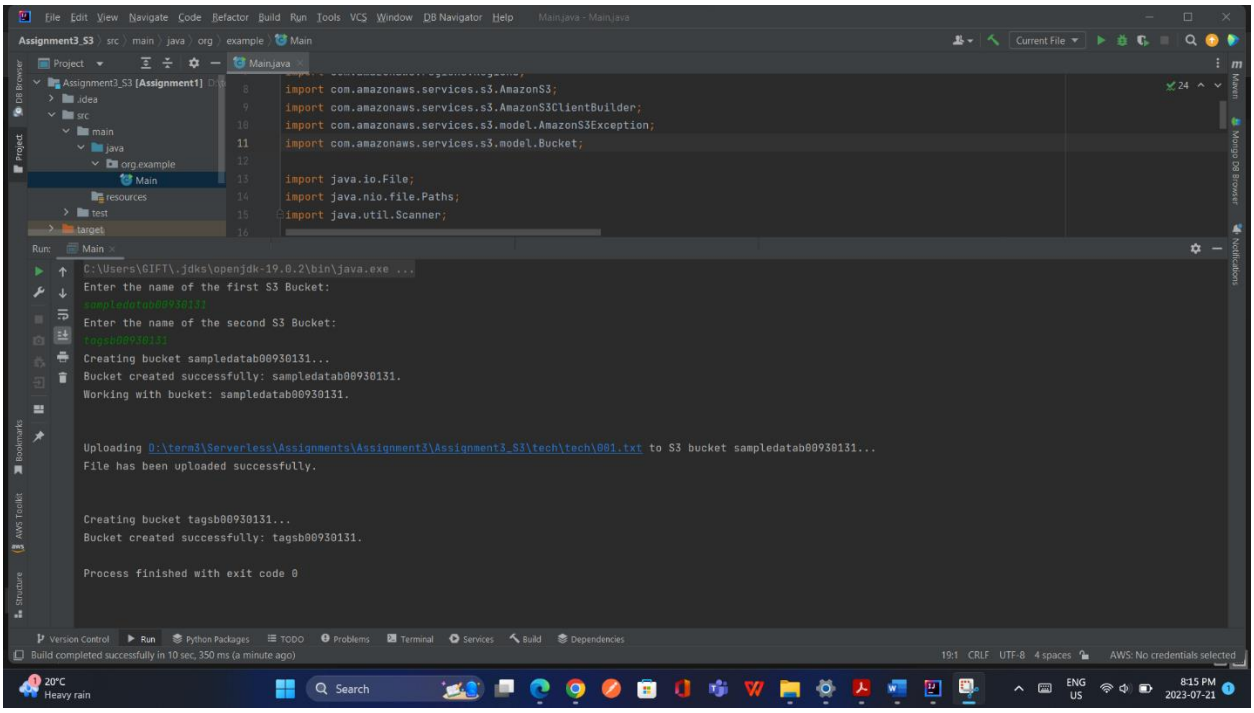

Figure 8 Two buckets named **sampledatab00930131** and **tagsb00930131** created using Java SDK [5]

Figure 9 illustrates those two buckets two buckets named **sampledatab00930131** and **tagsb00930131** created successfully and listed in AWS S3 buckets.

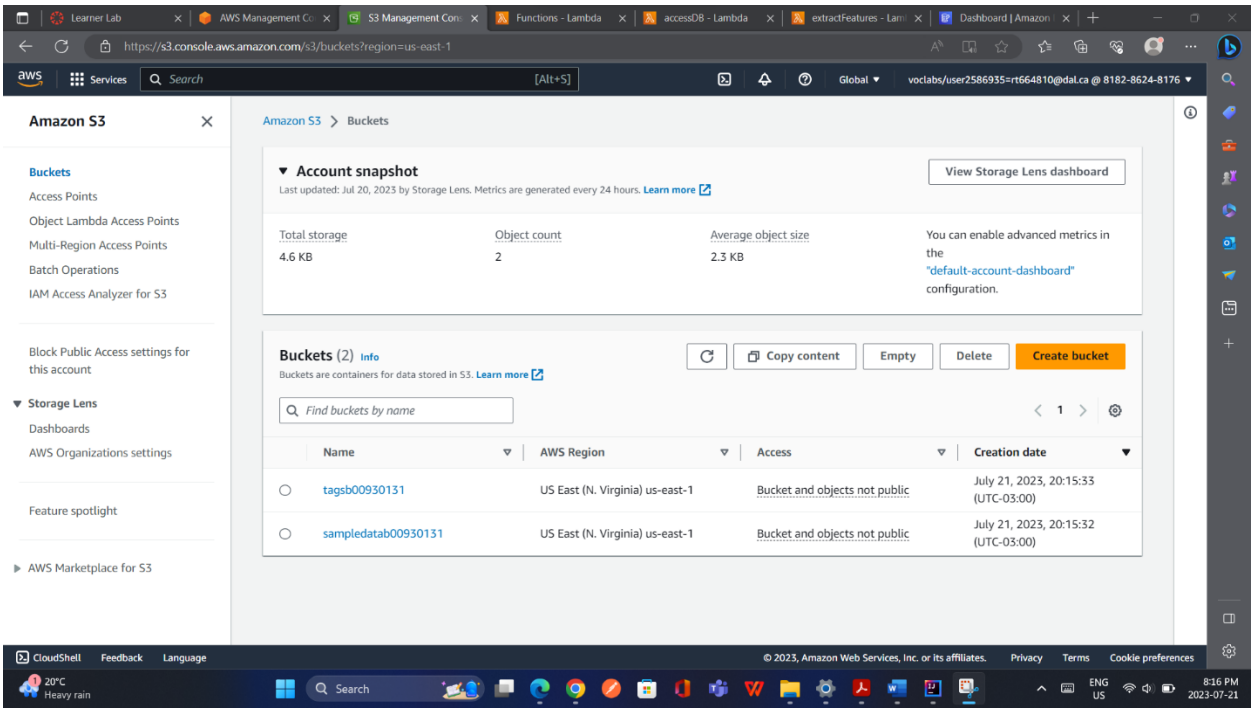

Figure 9 **sampledatab00930131** and **tagsb00930131** created successfully [2]

Lambda fucntion creation

Figures 10 and 11 show the steps require to cerate lambda function named **extractFeatures**. For creating lambda fucntion I chose Python to write a code. This lambda fucntion will trigger when file will be available in S3 bucket **sampledatab00930131** and get the text file from bucket.

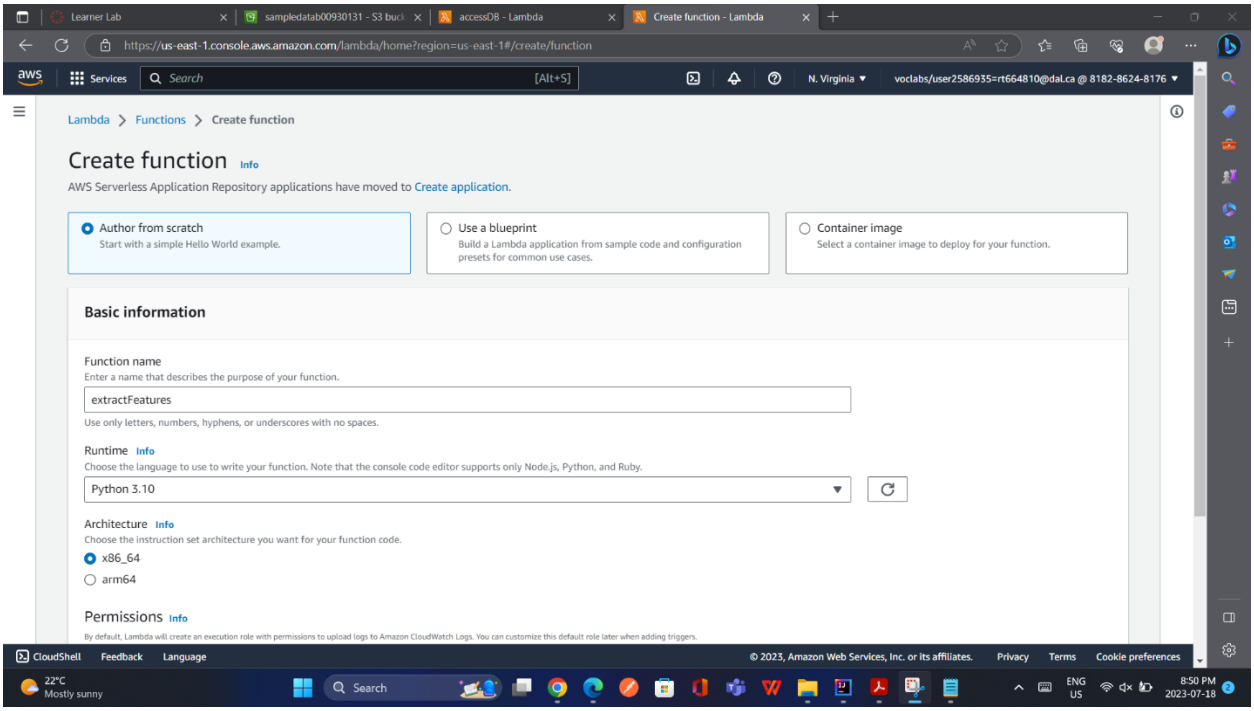

Figure 10 Creation of lambda function named **extractFeatures** [3]

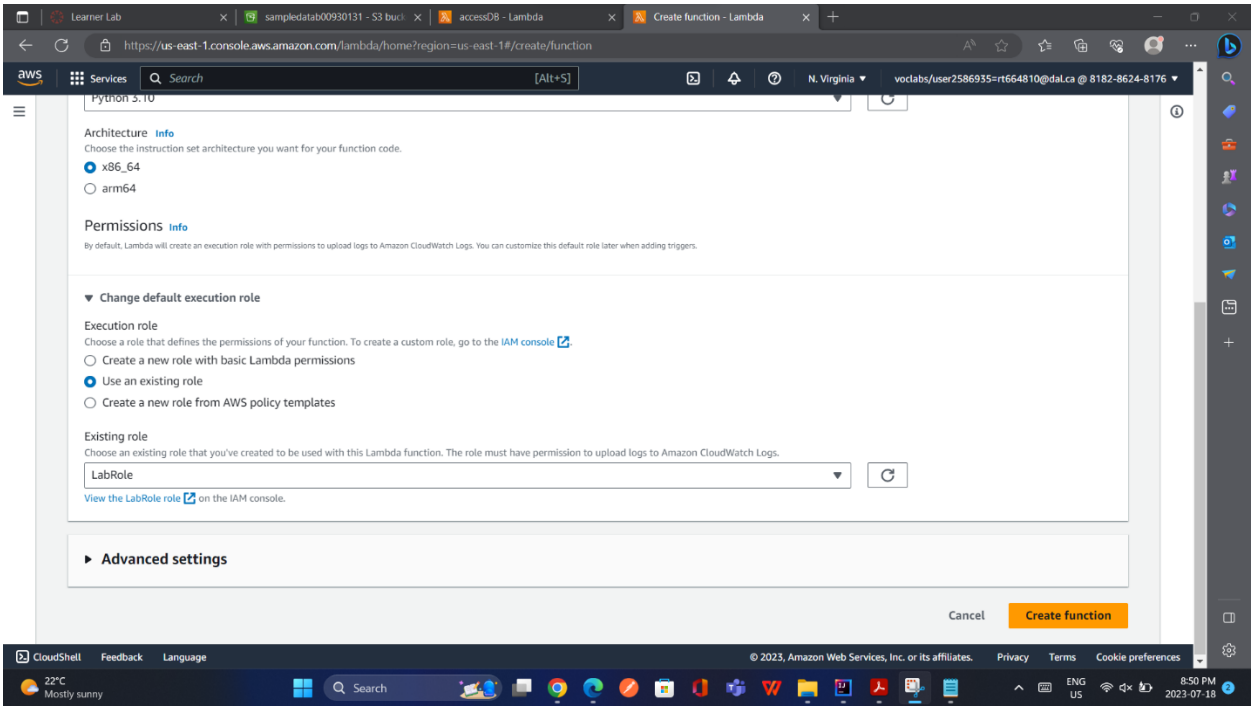

Figure 11 Creation of lambda function named **extractFeatures [3]**

Figure 12 shows that lambda function **extractFeatures** created successfully on AWS lambda functions.

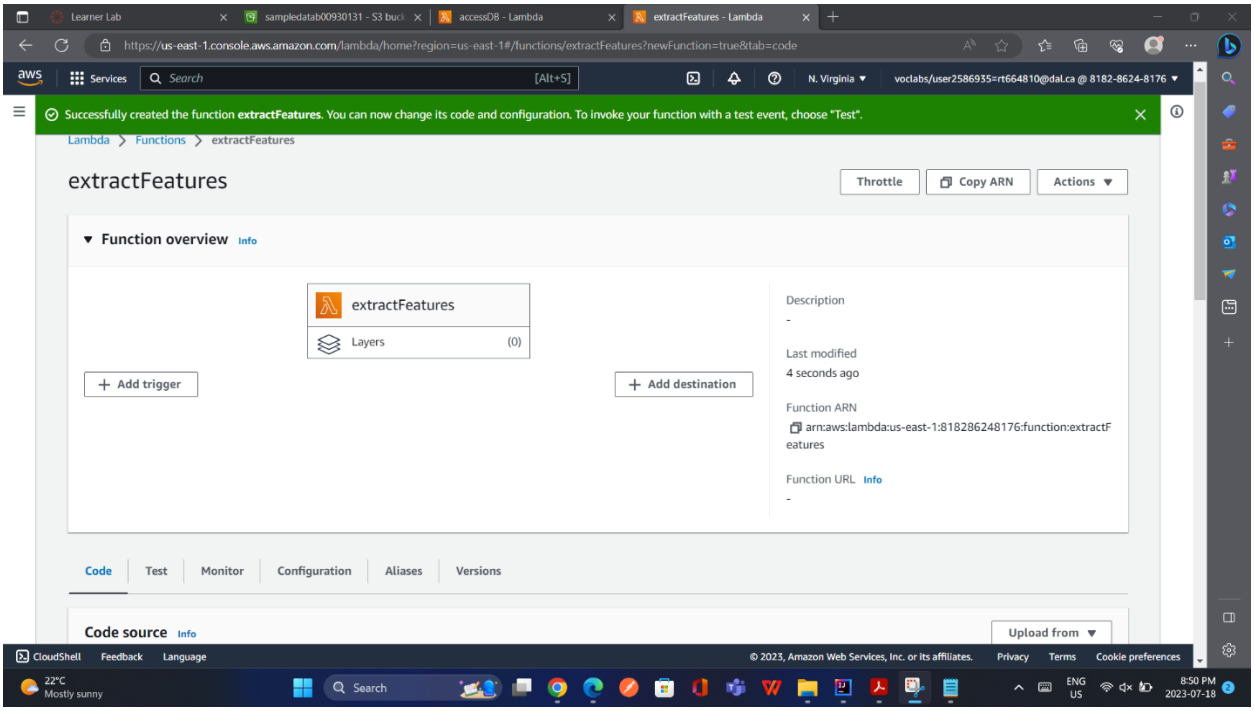

Figure 12 Lambda function named **extractFeatures** created sucessfully [3]

Figure 13 shows that lambda function **extractFeatures** created successfully and listed in AWS lambda functions.

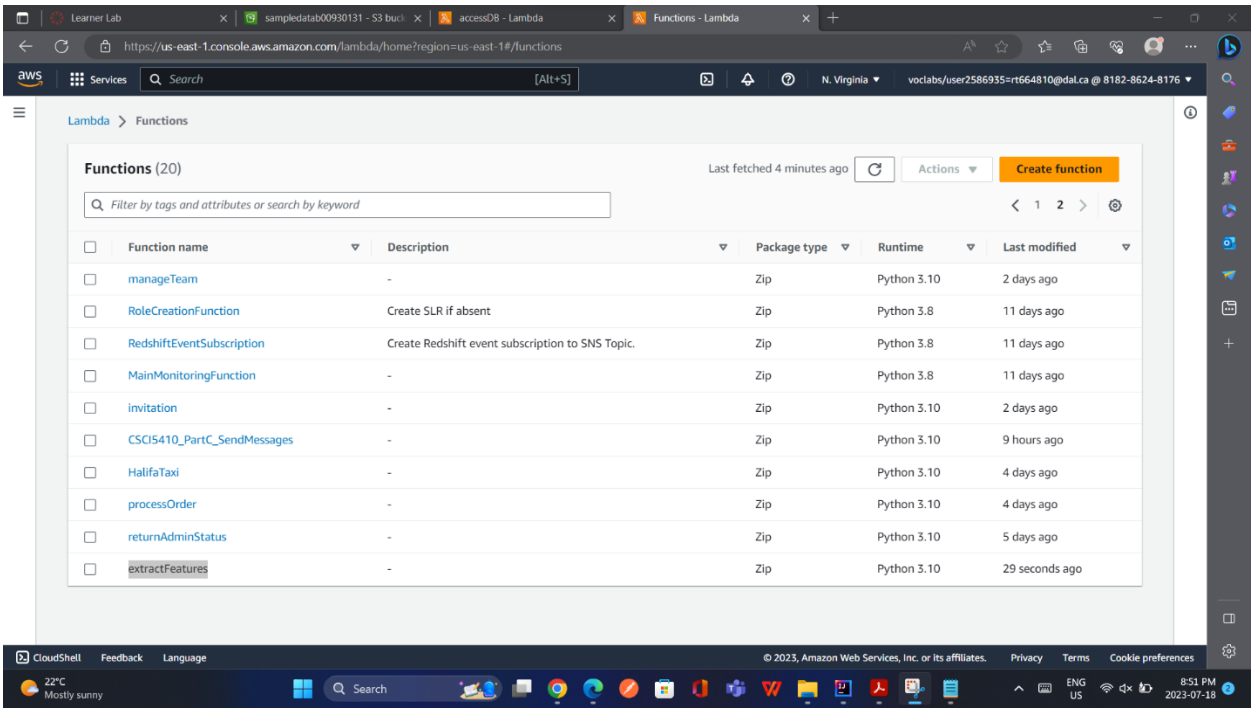

Figure 13 Created lambda function named **extractFeatures** is listed [3]

Figures 14 and 15 show the steps require to cerate lambda function named **accessDB**. For creating lambda fucntion I chose Python to write a code. This lambda fucntion will trigger when file will be available in S3 bucket **tagsb00930131** and get the text file from bucket. From that file, lambda fucntion will store key value in DynamoDB table.

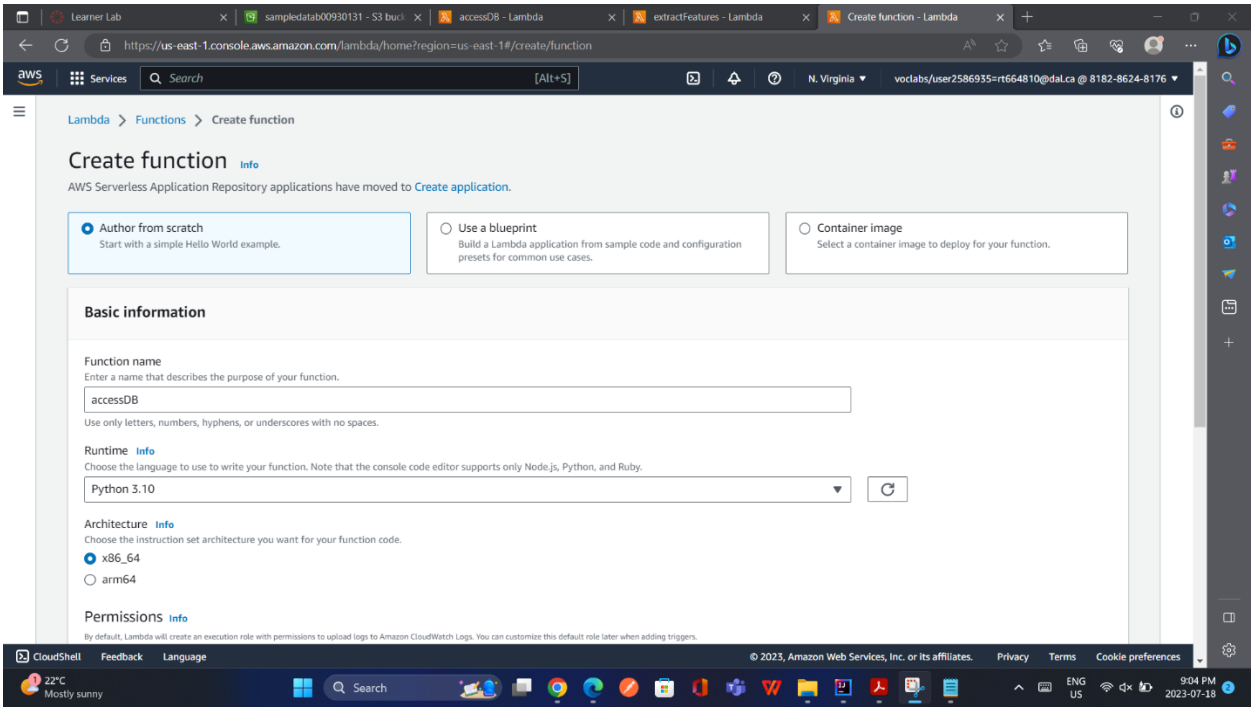

Figure 14 Creation of lambda function named **accessDB** [3]

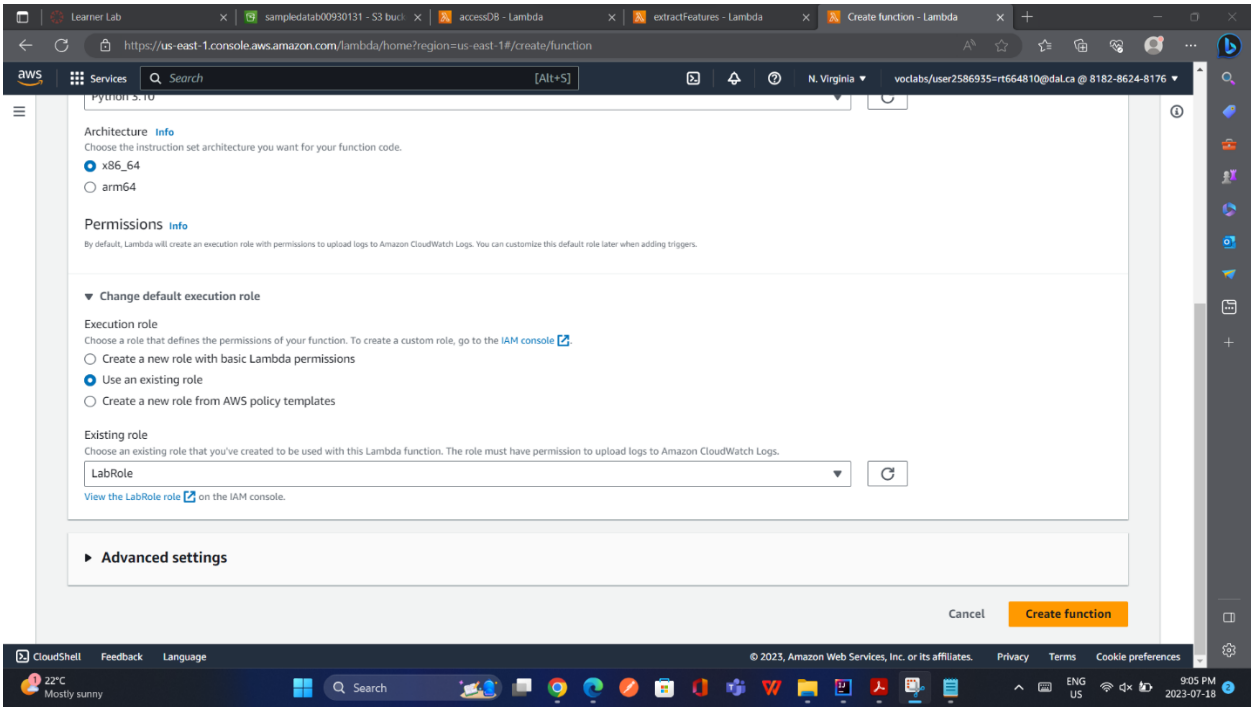

Figure 15 Creation of lambda function named **accessDB** [3]

Figure 16 shows that lambda function **accessDB** created successfully on AWS lambda functions.

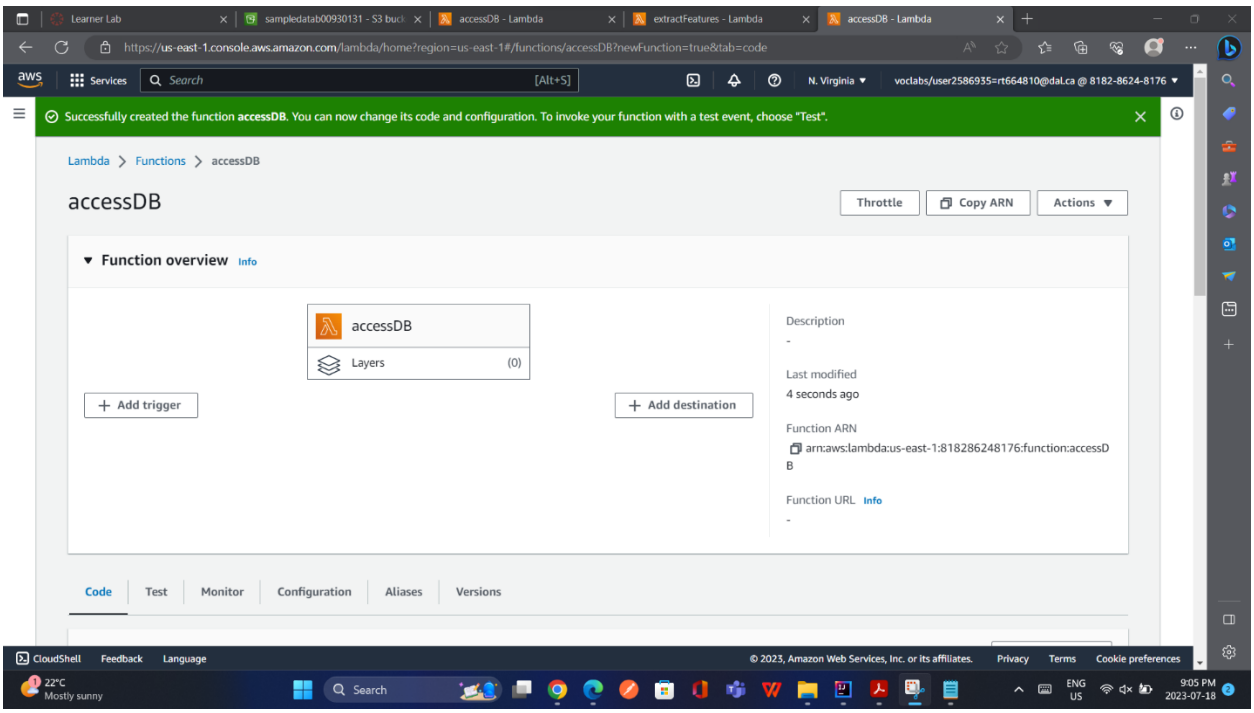

Figure 16 Lambda function named **accessDB** created successfully [3]

Figure 17 shows that lambda function **accessDB** created successfully and listed in AWS lambda functions.

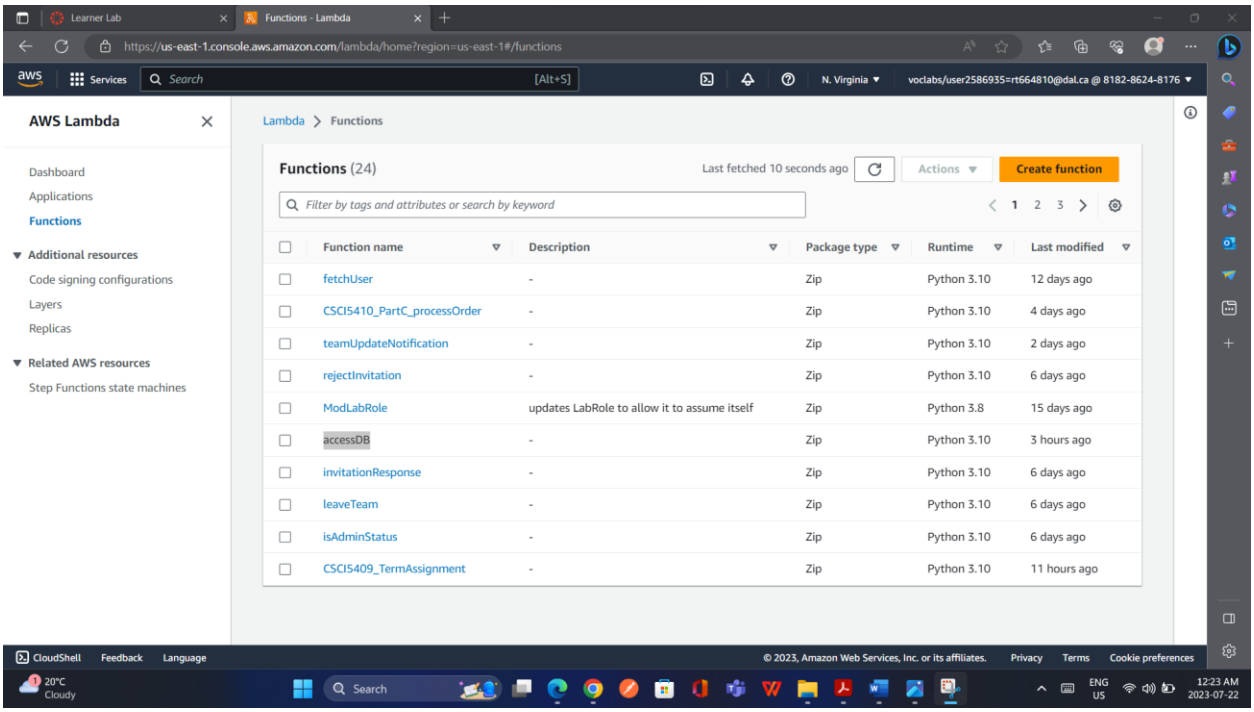

Figure 17 Created lambda function named **accessDB** is listed [3]

Notification trigger of lambda and S3 bucket

Figures 18,19,20,21 and 22 show steps required to trigger event notification between lambda fucntion named **extractFeatures** and bucket named sampledatab00930131. Figure 23 shows that lambda fucntion is successfully associated with s3 bucket named **sampledatab00930131**.

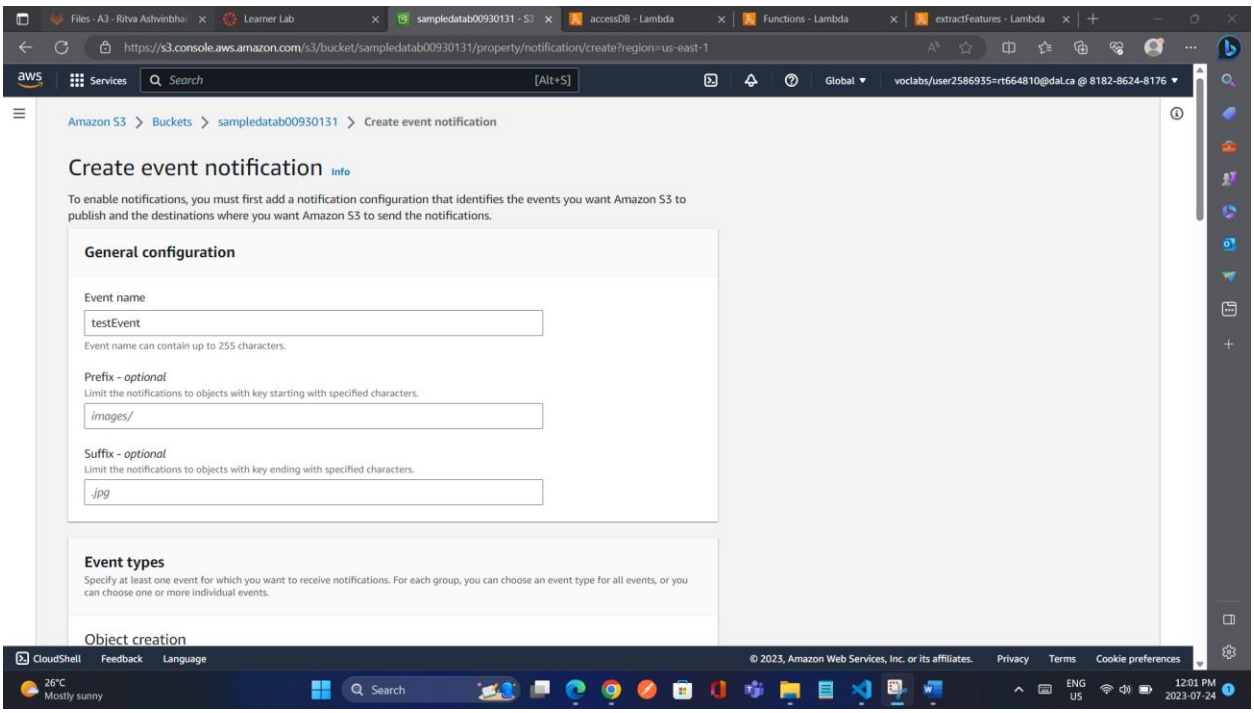

Figure 18 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2]

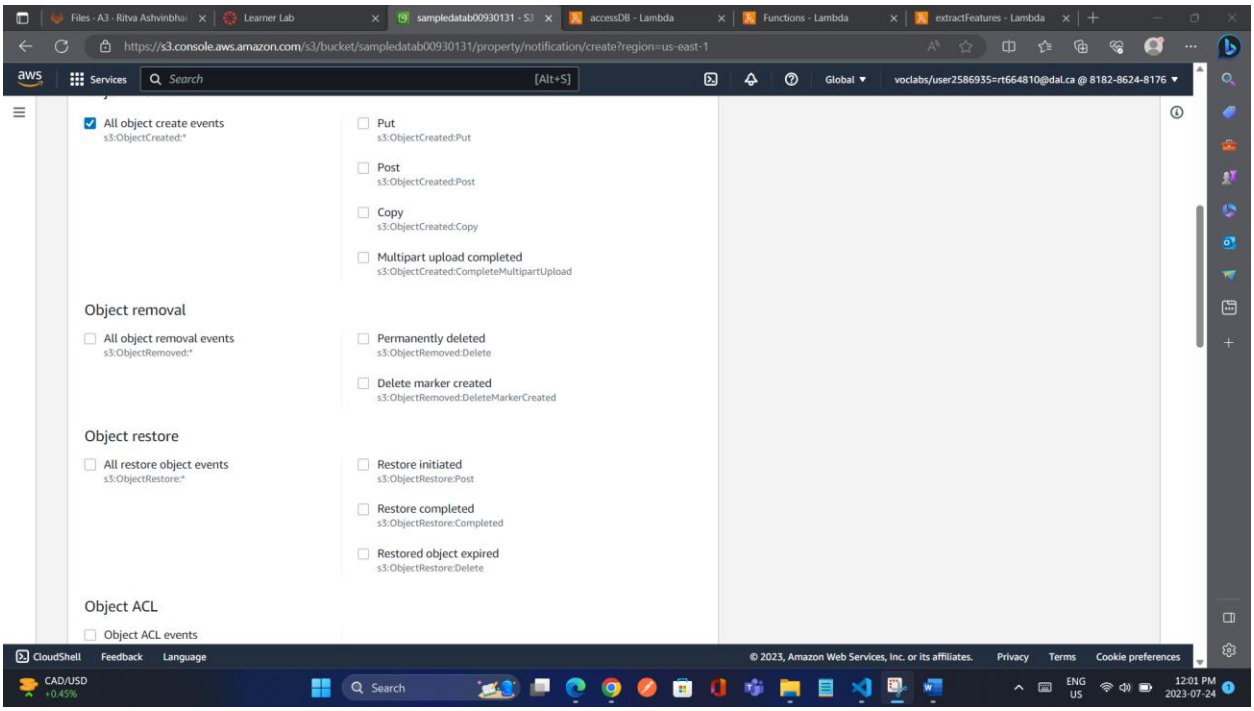

Figure 19 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2]

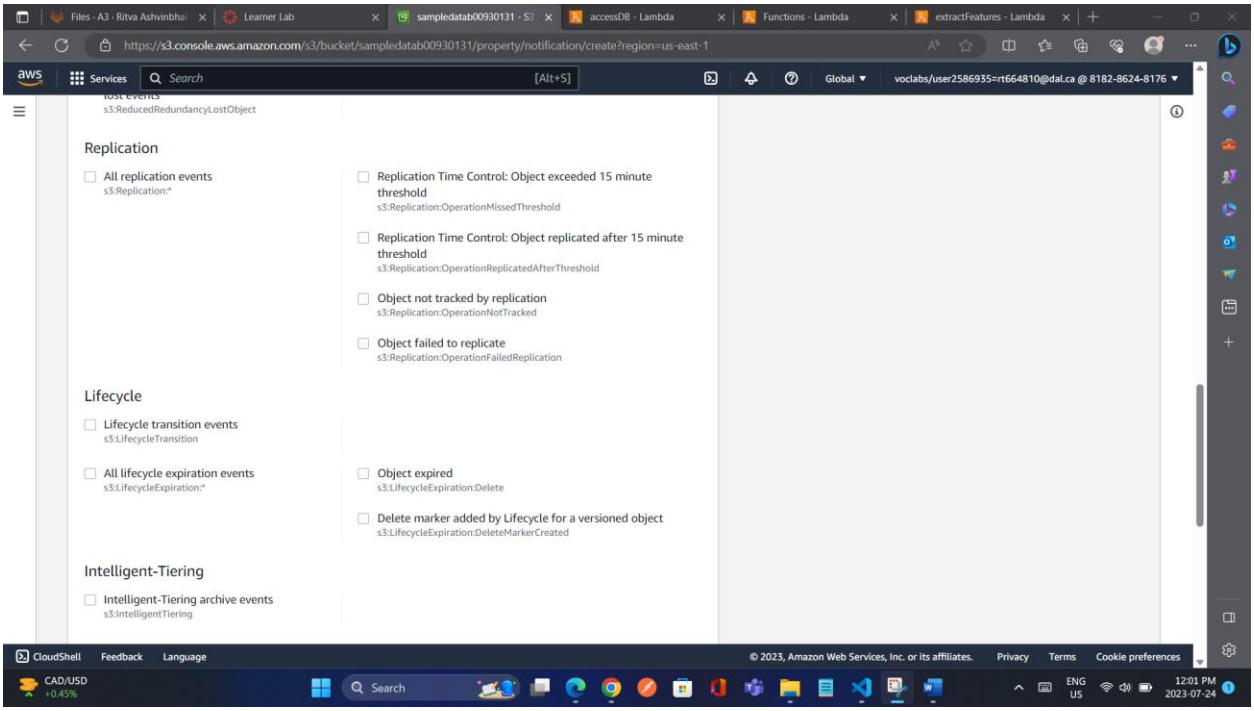

Figure 20 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2]

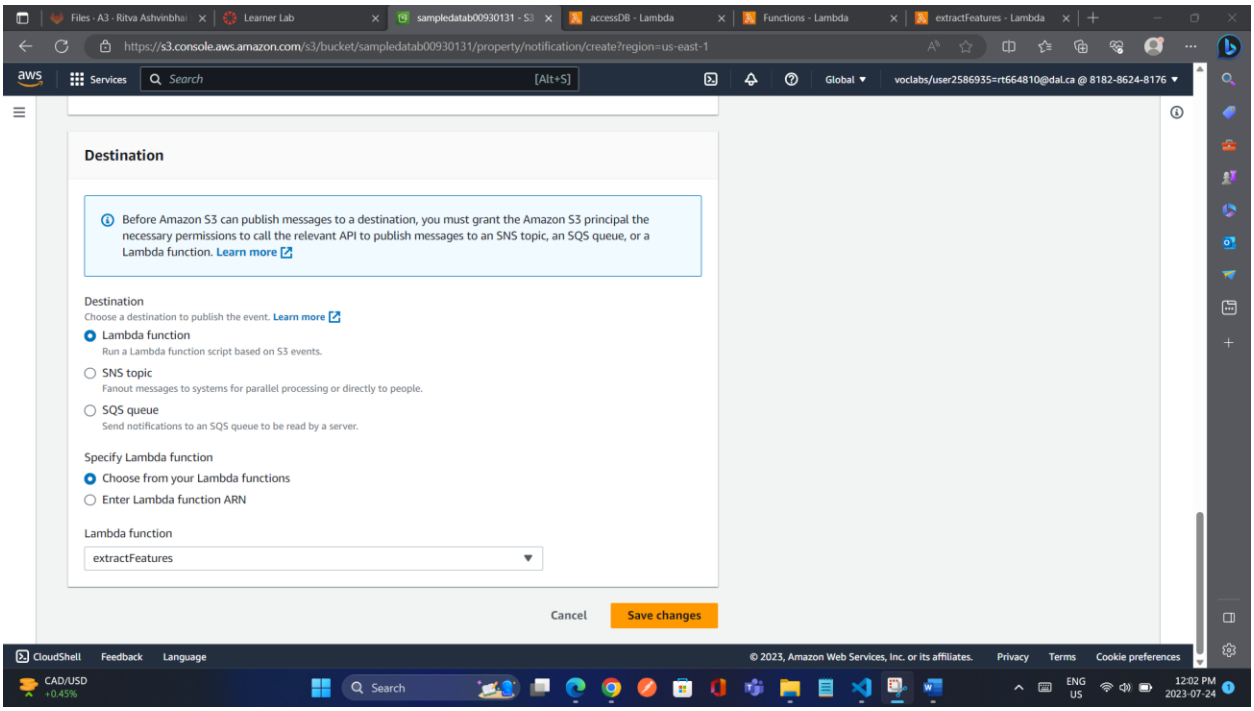

Figure 21 Creation of notification trigger between lambda named **extarctFeatures** and bucket named **sampledatab00930131** [2]

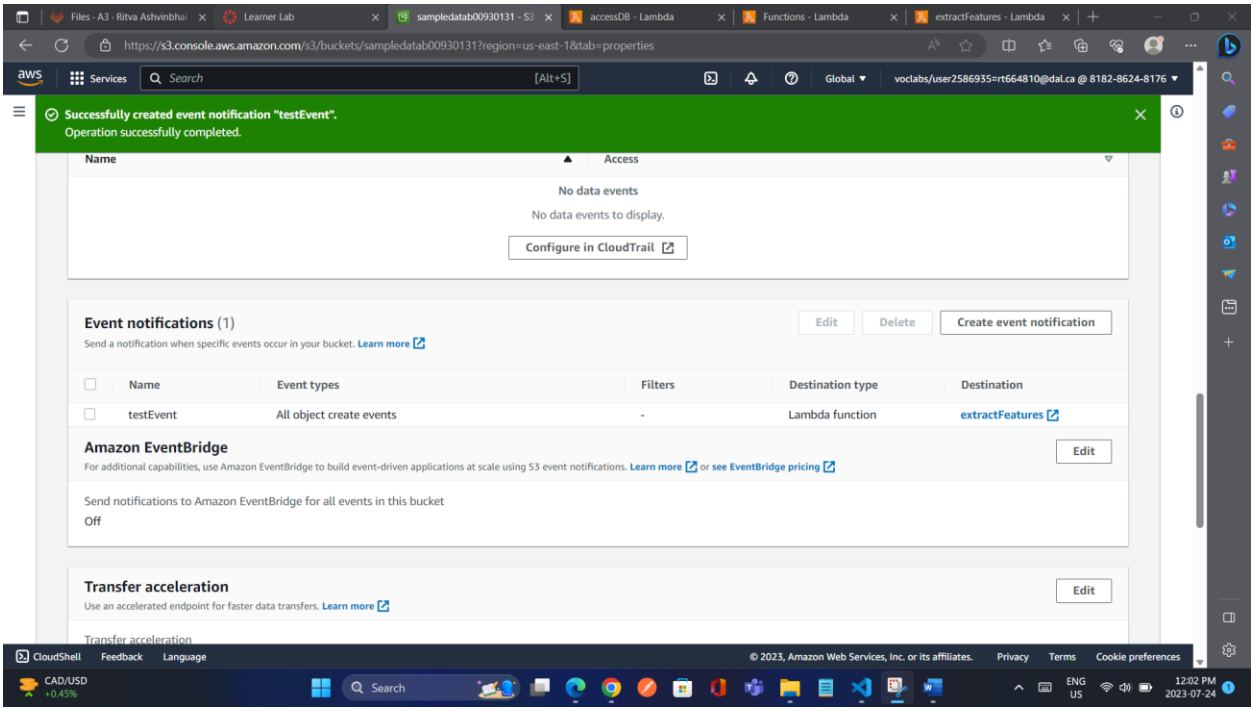

Figure 22 Lambda **extractFeatures** is associated with notification trigger of S3 bucket named **sampledatab00930131** [2]

Figures 23,24,25,26 and 27 show steps required to trigger event notification between lambda fucntion named **acessDB** and bucket named sampledatab00930131. Figure 27 shows that lambda fucntion is successfully associated with s3 bucket named **tagsb00930131**.

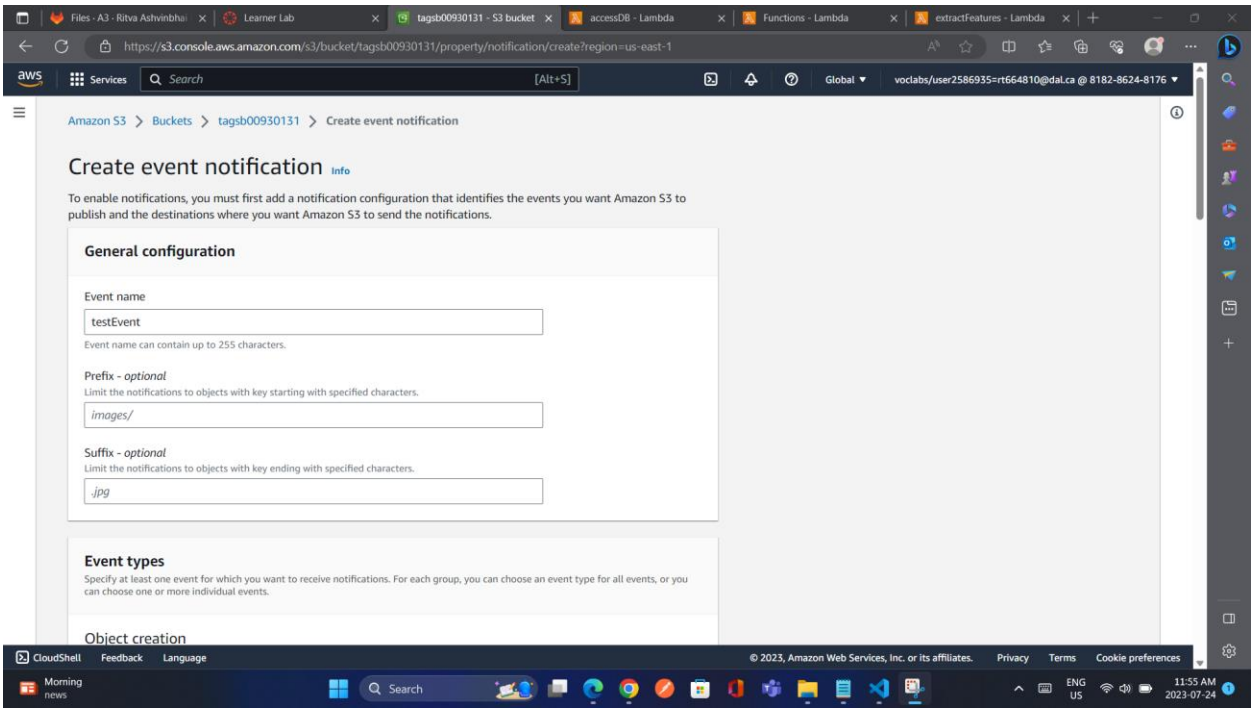

Figure 23 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2]

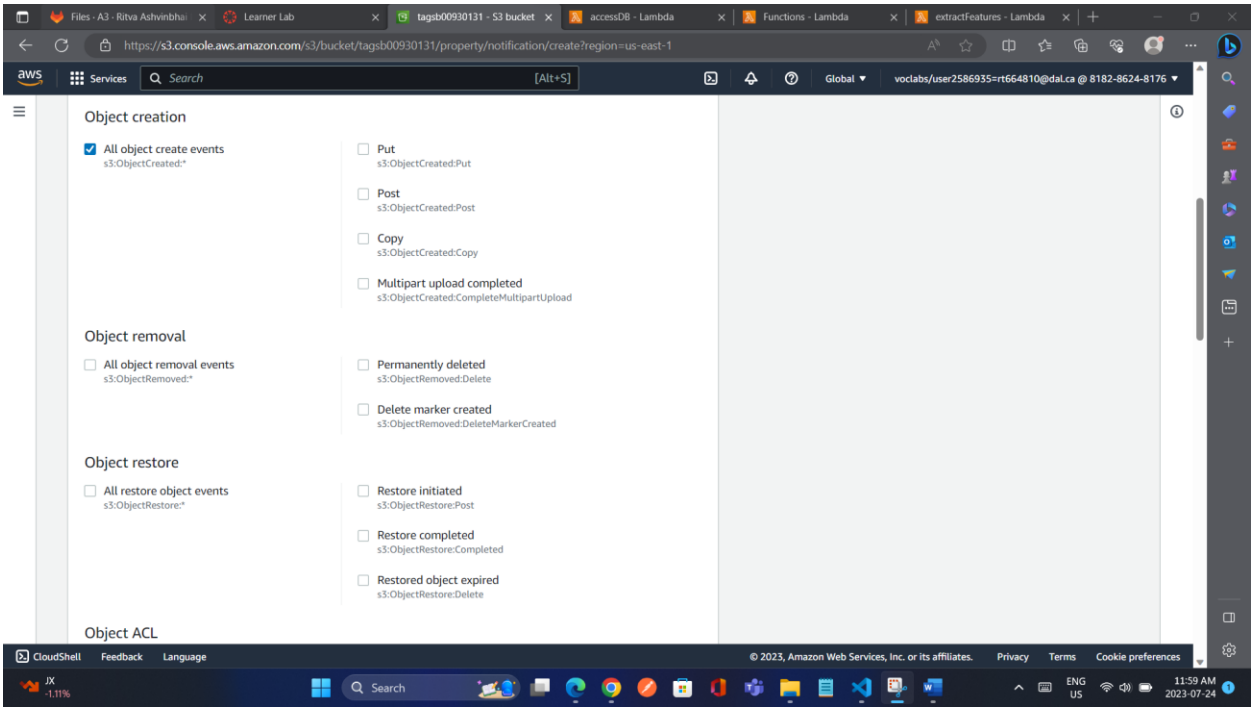

Figure 24 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2]

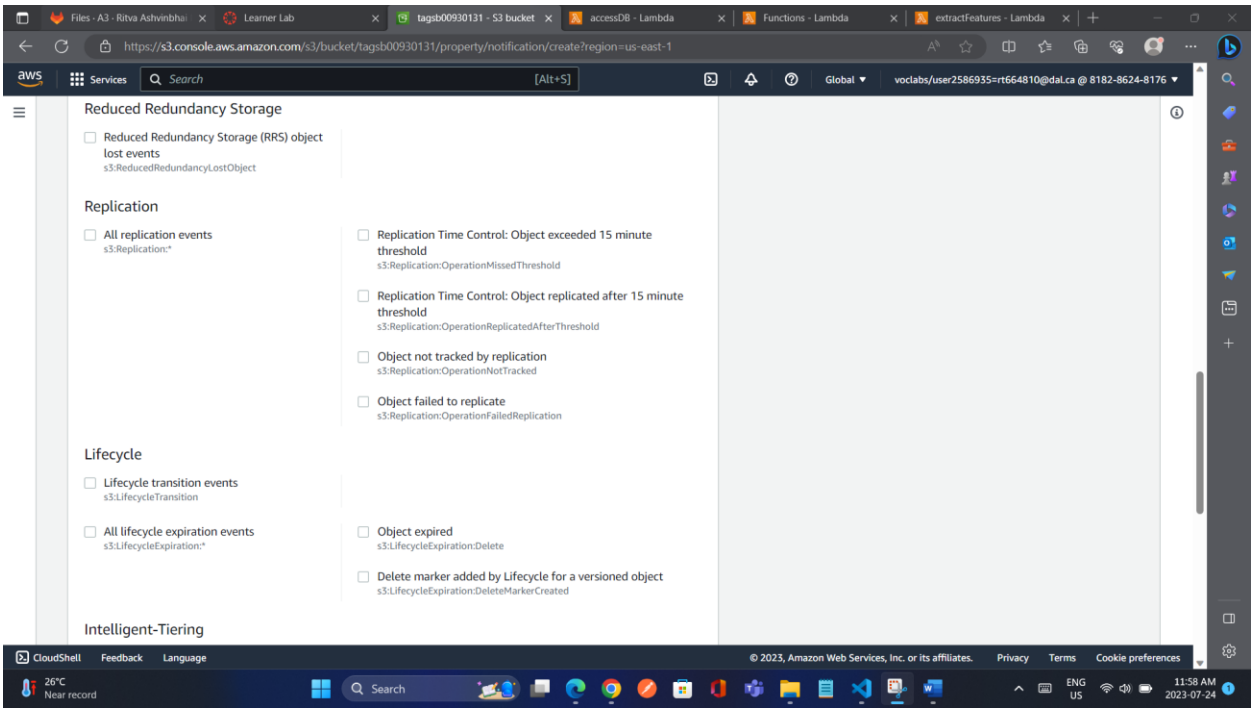

Figure 25 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2]

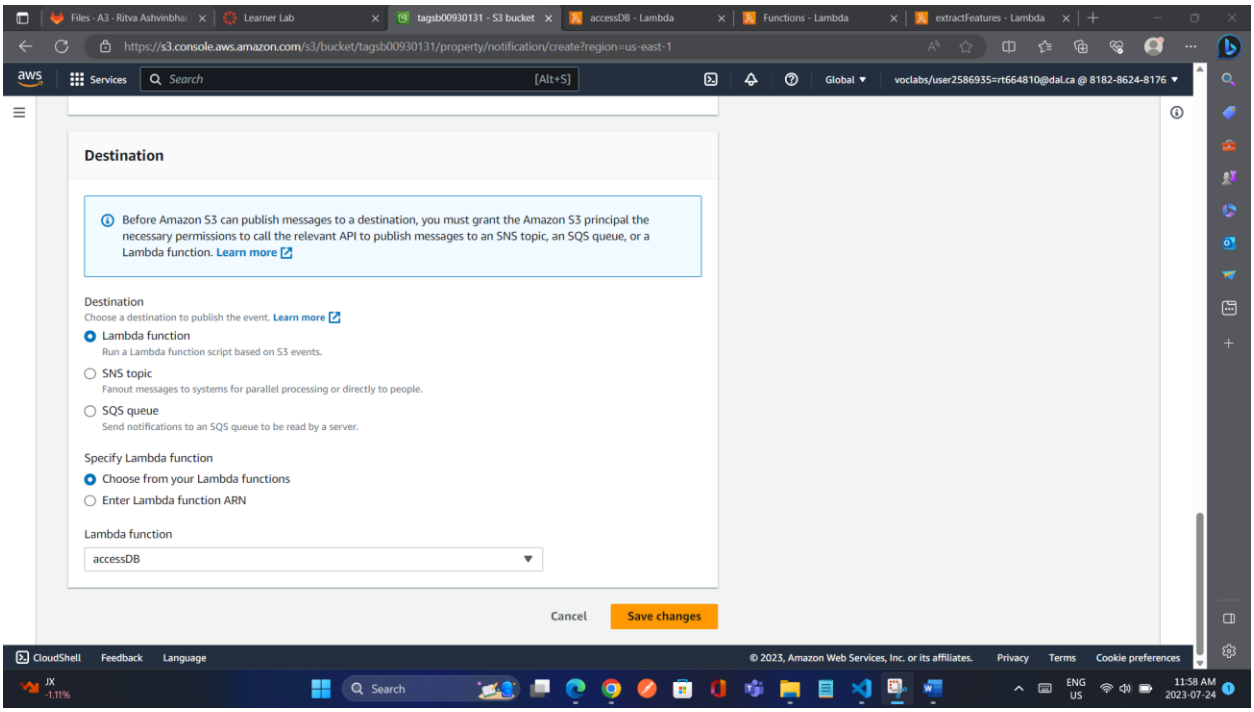

Figure 26 Creation of notification trigger between lambda named **accessDB** and bucket named **tagsb00930131** [2]

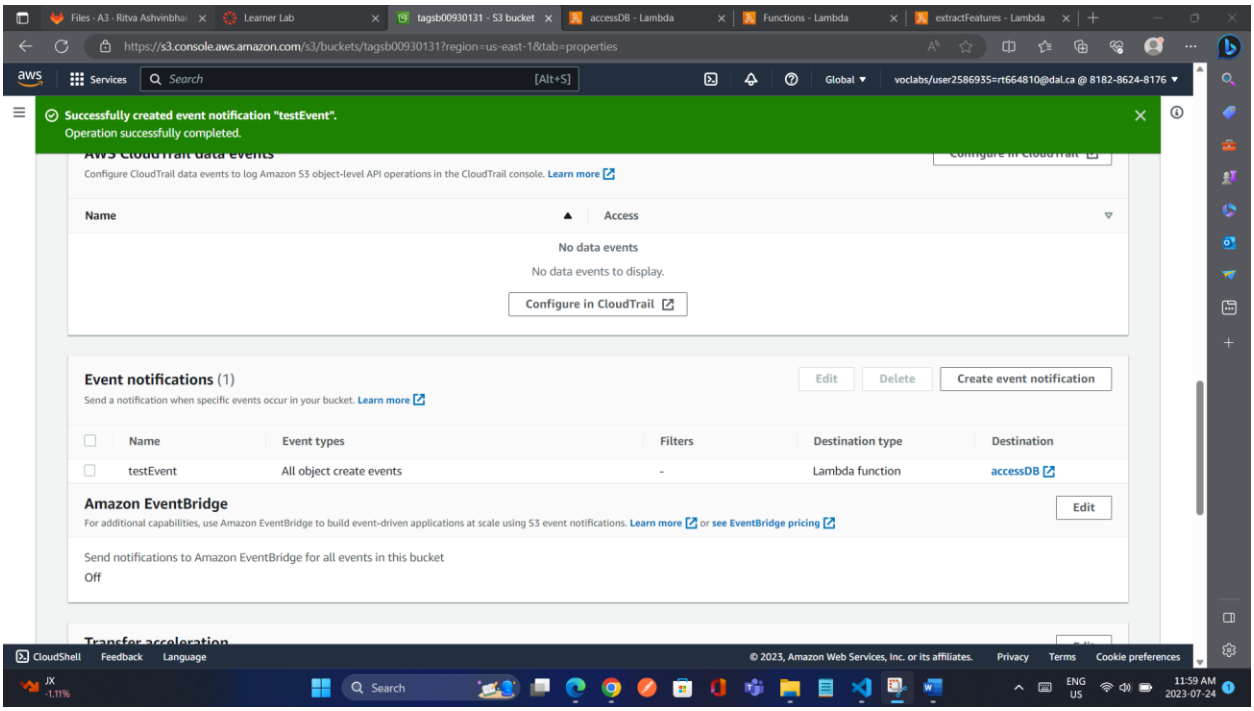

Figure 27 Lambda **accessDB** is associated with notification trigger of S3 bucket named **tagsb00930131** [2]

Files uploaded to sampledatab00930131 bucket using Java SDK

Figure 28 shows that file named **001.txt** uploaded to bucket **sampledatab00930131** using Java SDK.

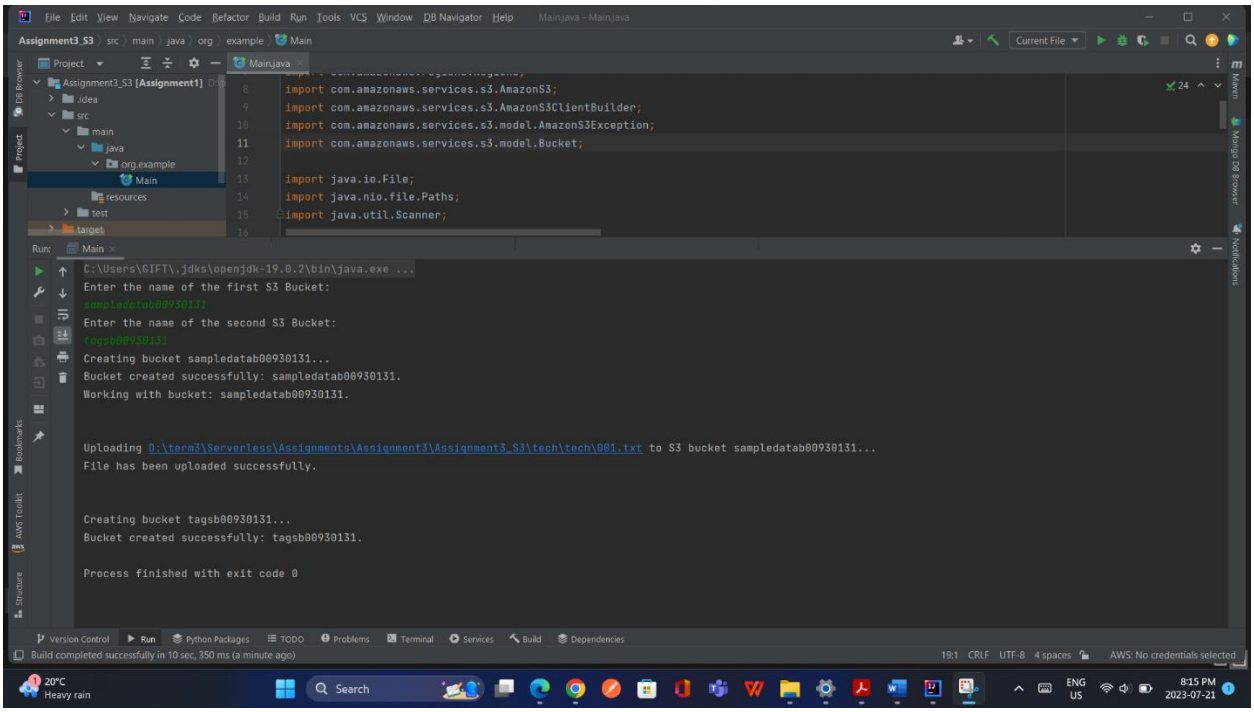

Figure 28 File named **001.txt** uploaded successfully to **sampledatab00930131** using Java SDK [5]

Figure 29 shows that file **001.txt** uploaded successfully to bucket **sampleb00930131**. When file upload to bucket **sampledatab0093031**, lambda **extractFeatures** will be trigger. Lambda will extract the name entities from the **001.txt file**, create another text file named **001ne.txt** in the bucket **tagsb00930131** and store extracted name entities in the **001ne.txt** file in json format. Figure 30 shows the execution logs of lambda fucntion **extractFeatures.**

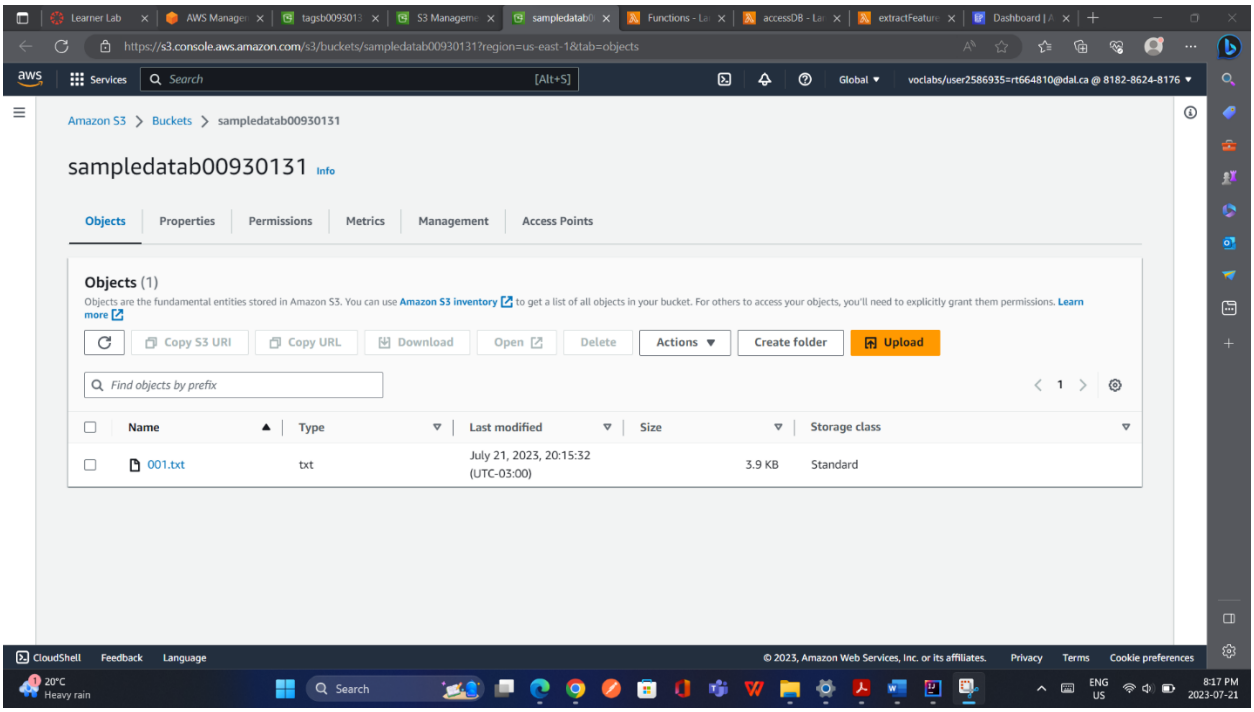

Figure 29 **001.txt** file uploaded to **sampeldatab00930131** successfully [2]

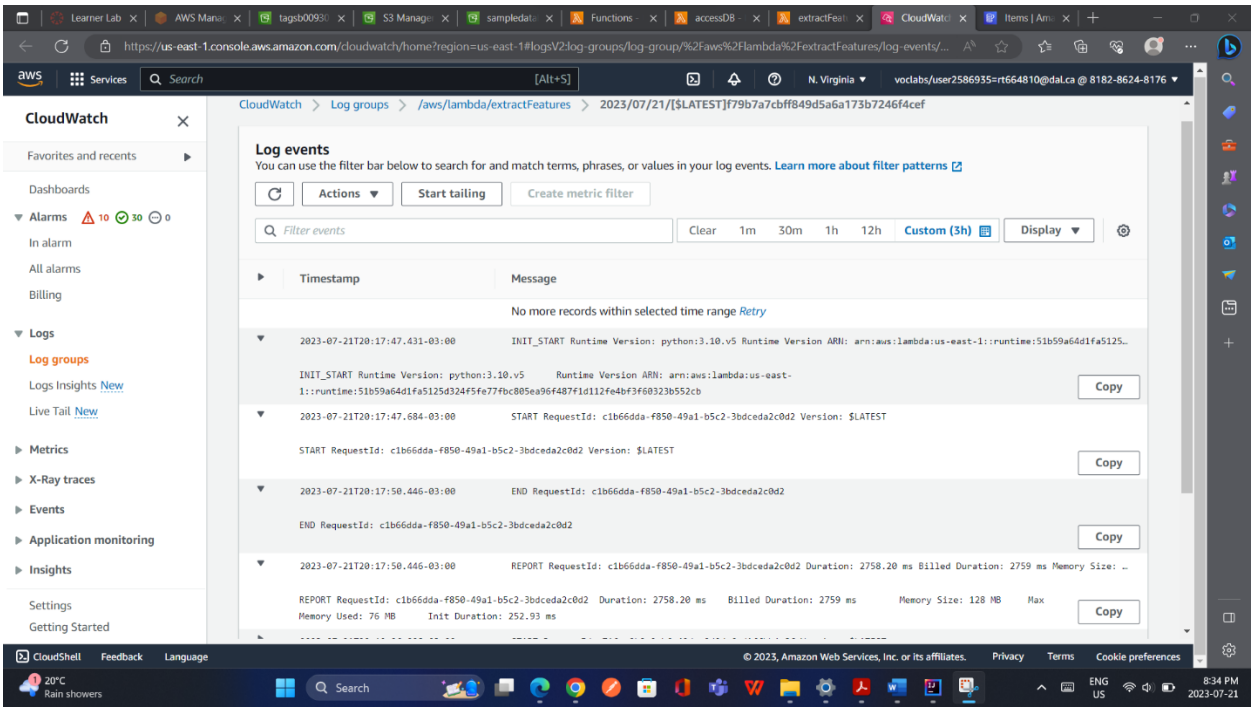

Figure 30 Execution logs of lambda **extractFeatures** to extract name entity of **001.txt** file from bucket sampeldatab00930131 [4]

Figure 31 shows that file named **001ne.txt** created and uploaded to bucket **tagsb00930131** by lambda **extractFeatures**. When file will be available in bucket **tagsb00930131**, lambda **accessDB** will be trigger and store the name entities in DynamoDB **CSCI5410_Ass3_part2** table in key value pair. Figure 32 shows that lambda **accessDB** successfully uploaded key value pair in table **CSCI5410_Ass3_part2**. Figure 33 show the execution log of lambda **accessDB**.

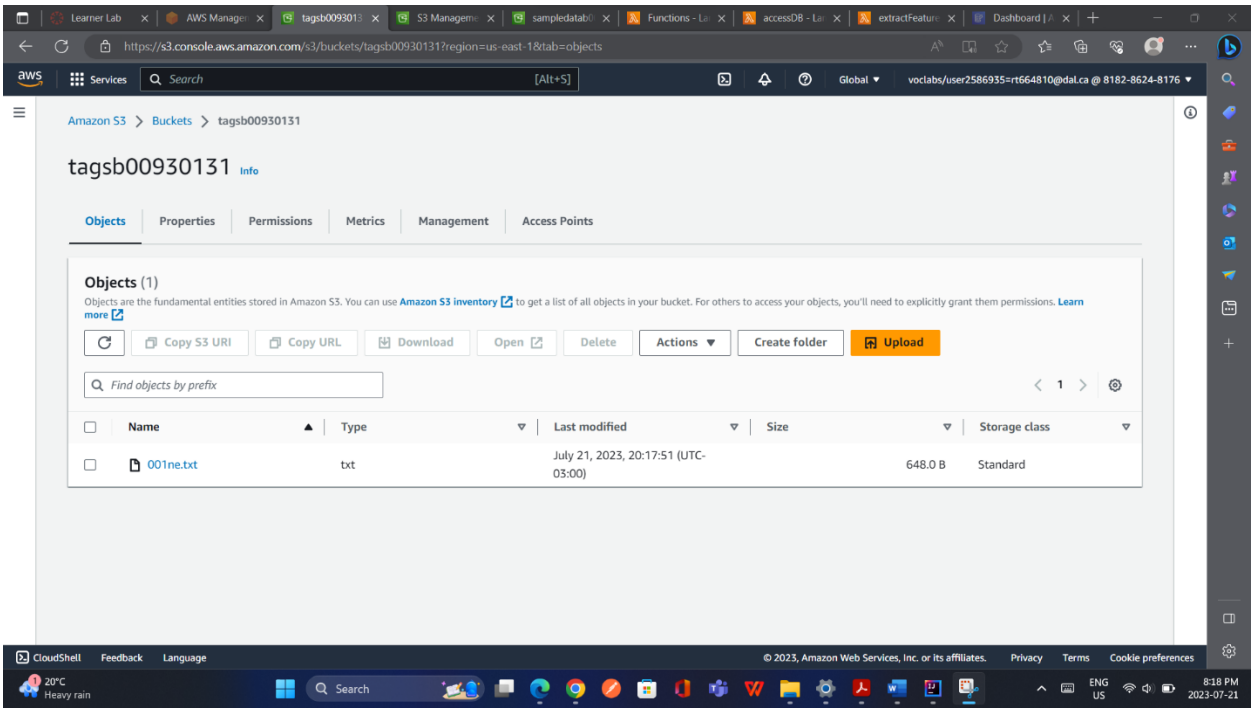

Figure 31 **001ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [3]

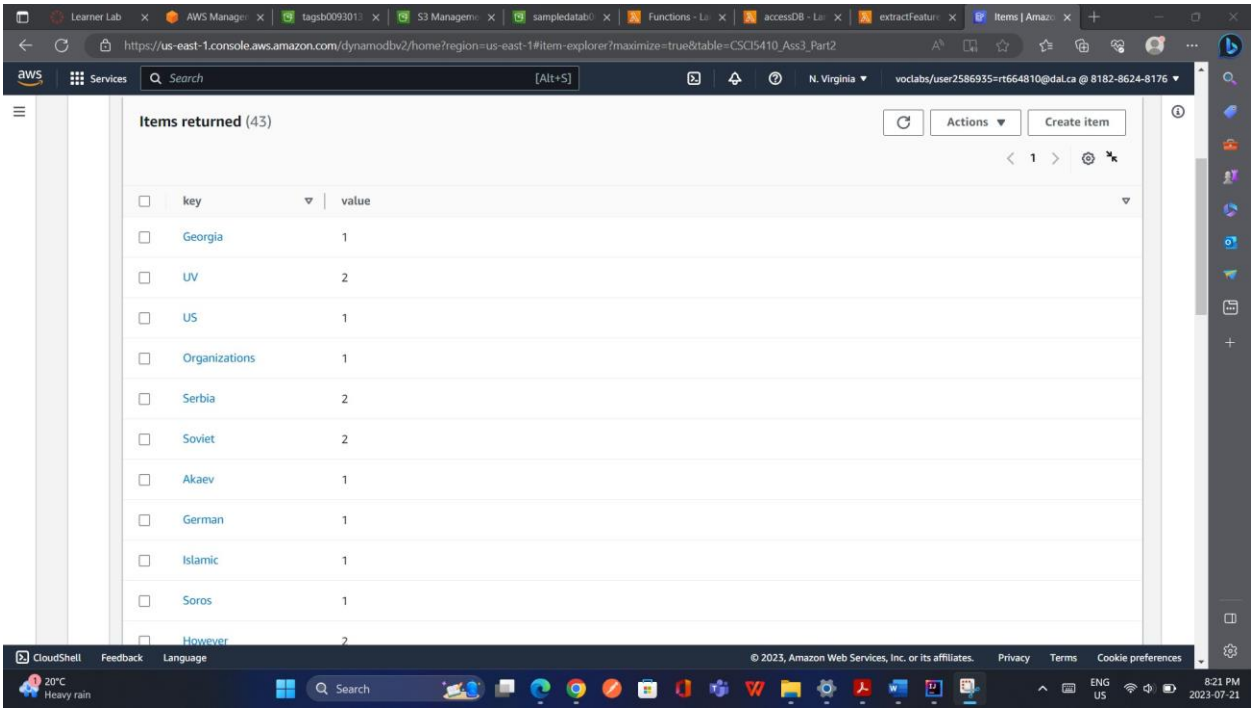

Figure 32 store key value from **001ne.txt** file to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1]

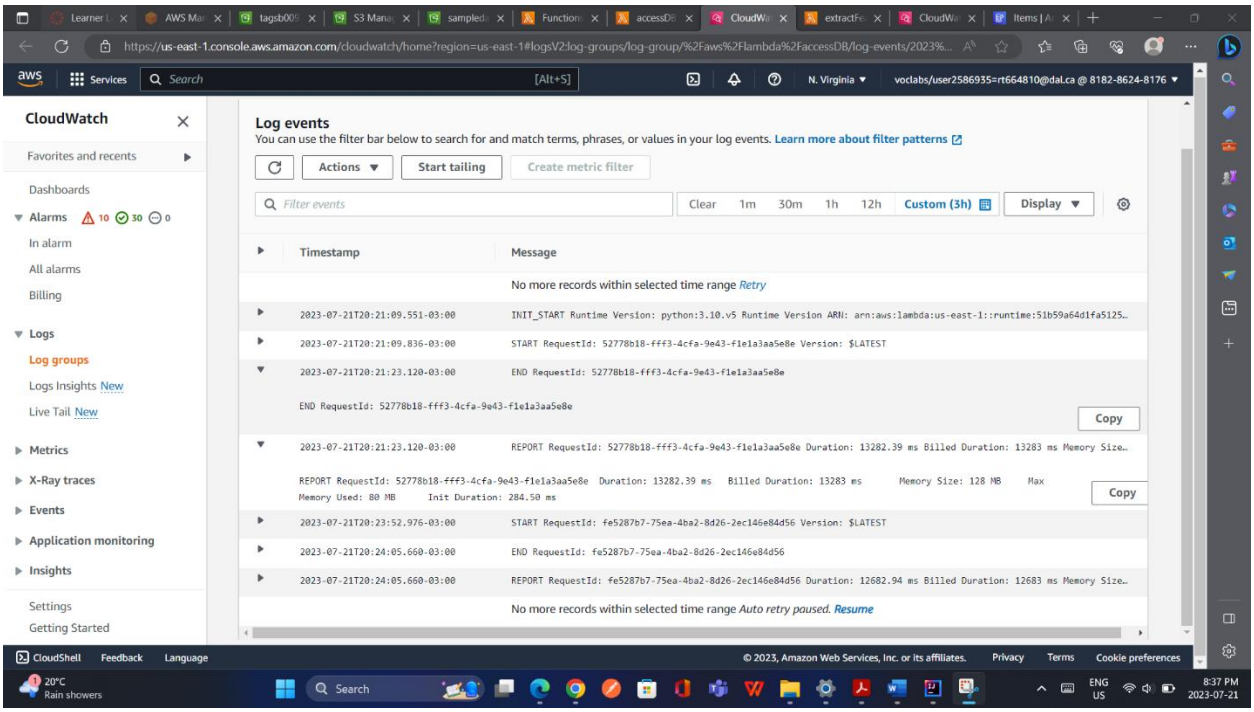

Figure 33 Execution logs of lambda **accessDB** to store the key value of **001ne.txt** file and store it to DynamoDB table [4]

Figure 34 shows that file named **002.txt** uploaded to bucket **sampledatab00930131** using Java SDK.

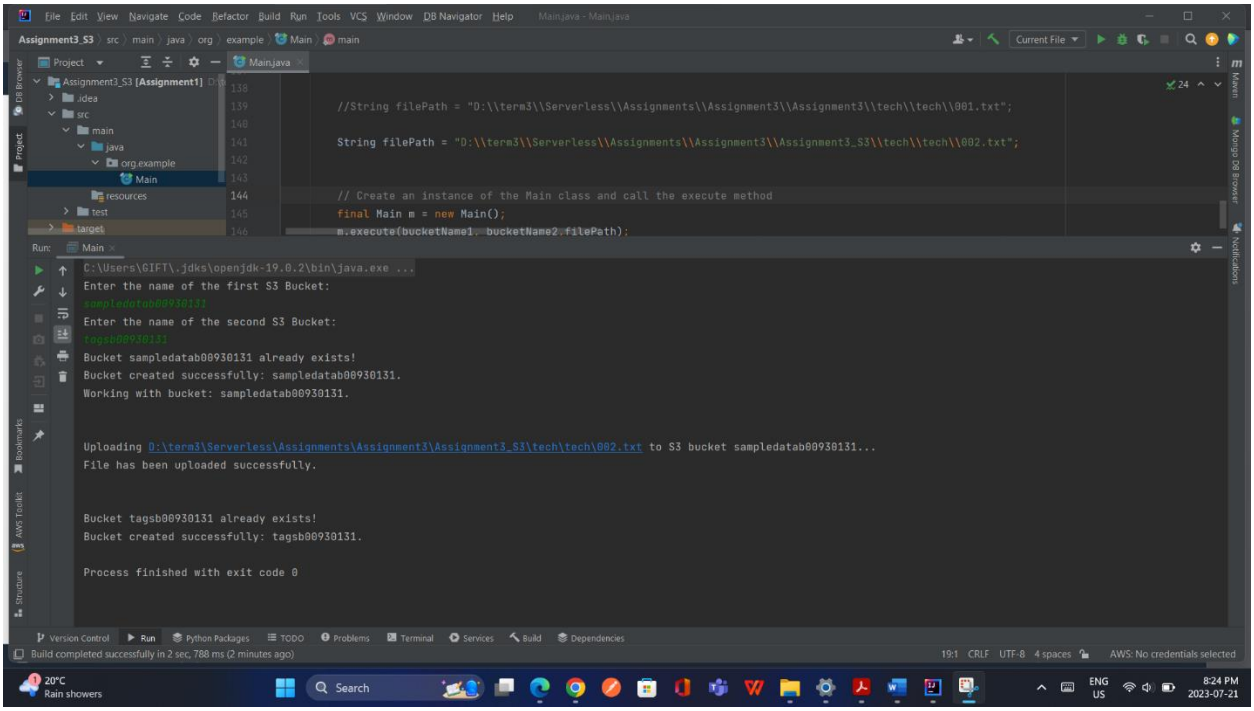

Figure 34 File named 002.txt uploaded successfully to sampledatab00930131 using Java SDK [5]

Figure 35 shows that file **002.txt** uploaded successfully to bucket **sampleb00930131**. When file upload to bucket **sampledatab0093031**, lambda **extractFeatures** will be trigger. Lambda will extract the name entities from the **002.txt file**, create another text file named **002ne.txt** in the bucket **tagsb00930131** and store extracted name entities in the **001ne.txt** file in json format. Figure 36 shows the execution logs of lambda fucntion **extractFeatures.**

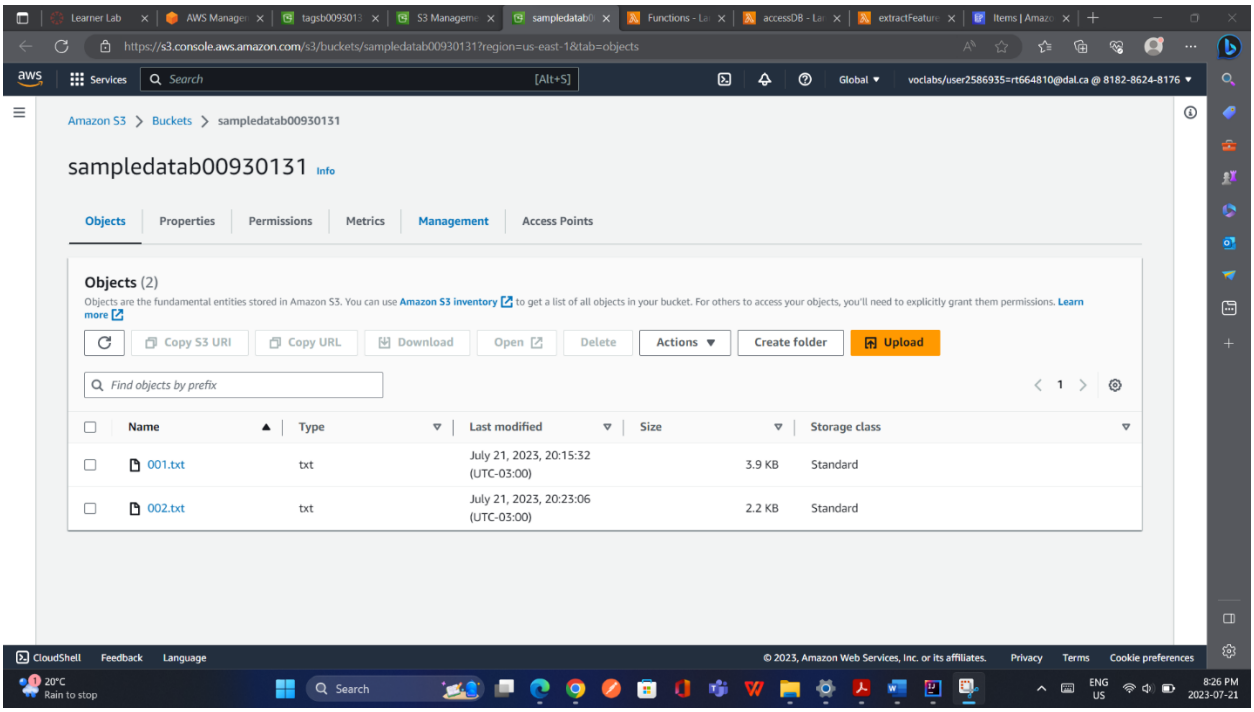

Figure 35 **002.txt** file uploaded to **sampeldatab00930131** successfully [2]

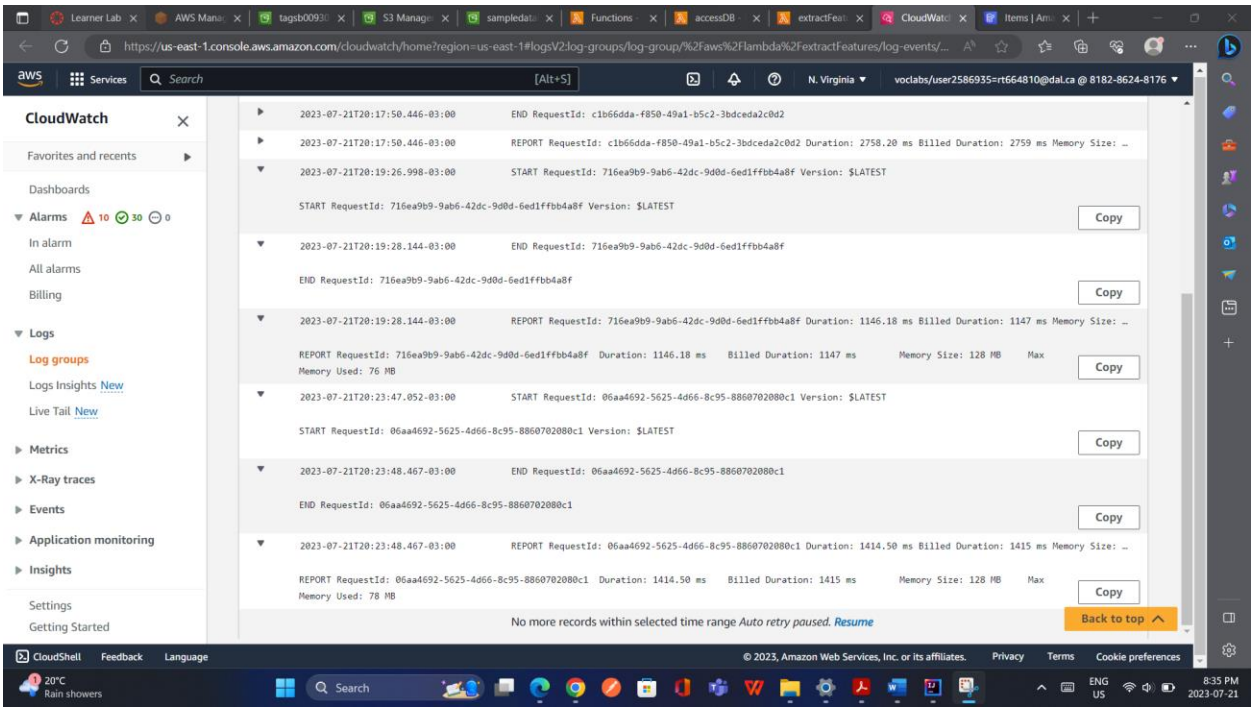

Figure 36 Execution logs of lambda **extractFeatures** to extract name entity of **002.txt** file from bucket **sampeldatab00930131** [4]

Figure 37 shows that file named **002ne.txt** created and uploaded to bucket **tagsb00930131** by lambda **extractFeatures**. When file will be available in bucket **tagsb00930131**, lambda **accessDB** will be trigger and store the name entities in DynamoDB **CSCI5410_Ass3_part2** table in key value pair. Figure 38 shows that lambda **accessDB** successfully uploaded key value pair in table **CSCI5410_Ass3_part2**. Figure 39 show the execution log of lambda **accessDB**.

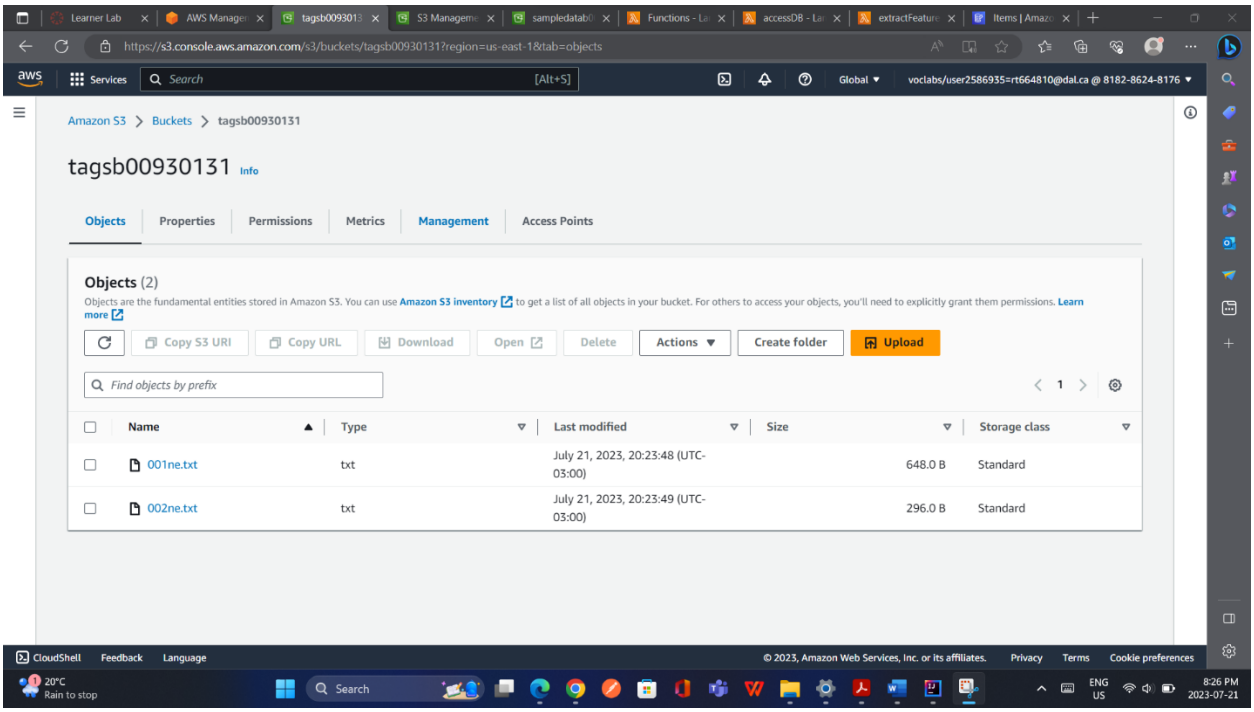

Figure 37 **002ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [2]

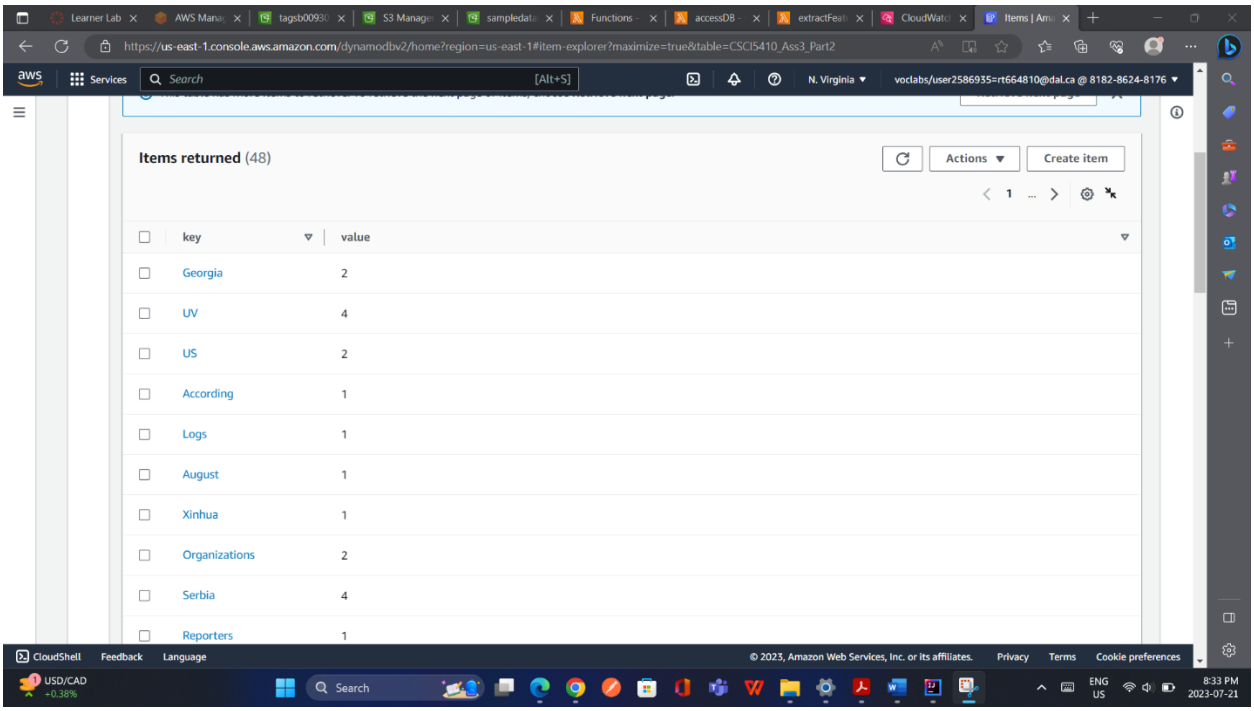

Figure 38 store key value from **002ne.txt** file to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1]

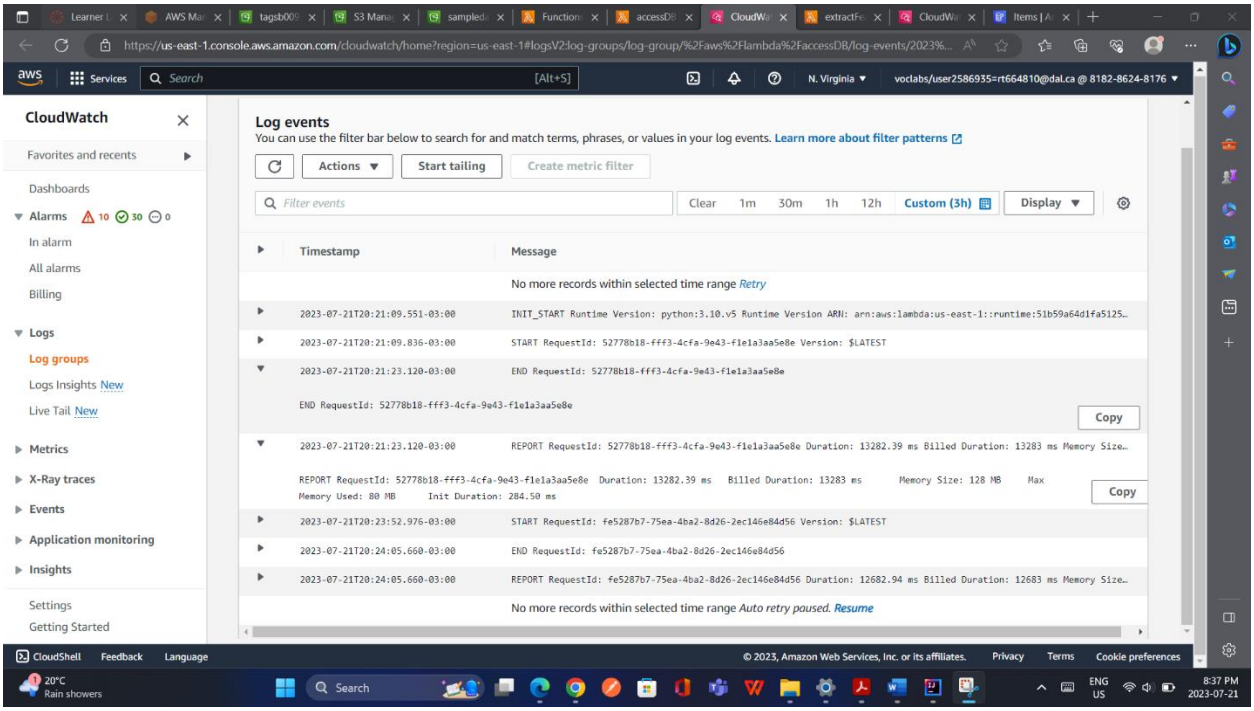

Figure 39 Execution logs of lambda **accessDB** to get the key value of **002ne.txt** file and store it to DynamoDB table [4]

Figure 40 shows that file named **003.txt** uploaded to bucket **sampledatab00930131** using Java SDK.

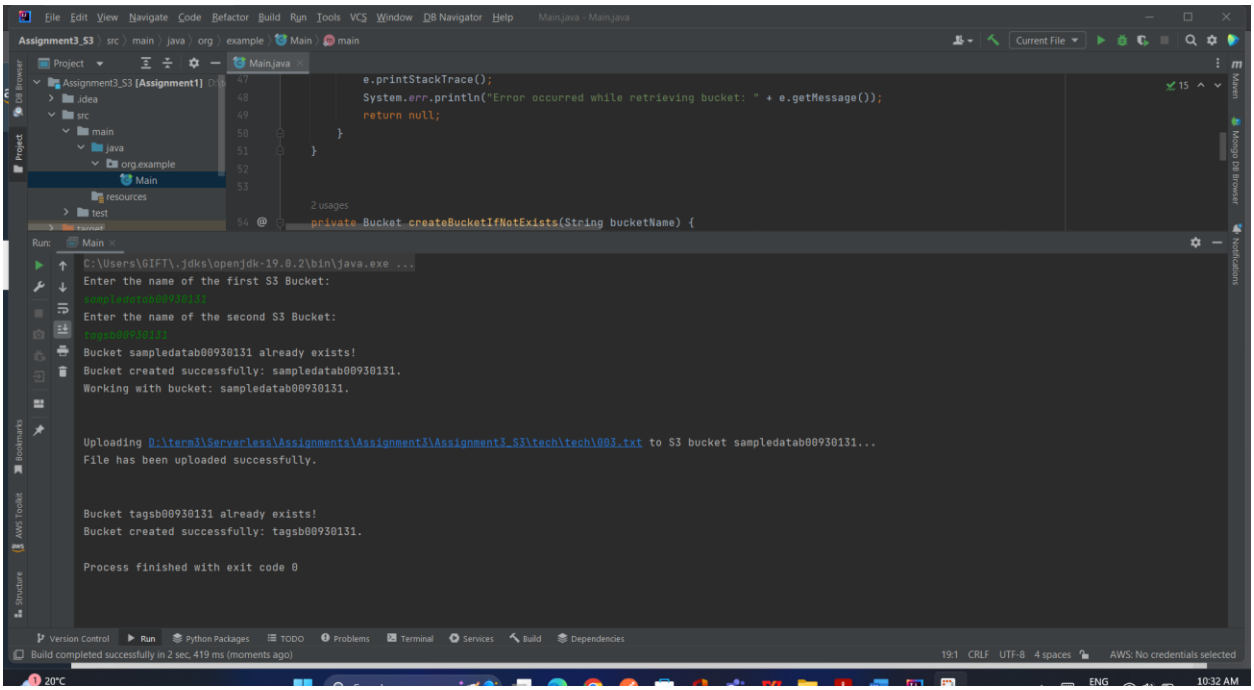

Figure 40 File named **003.txt** uploaded successfully to **sampledatab00930131** using Java SDK [5]

Figure 41 shows that file **003.txt** uploaded successfully to bucket **sampleb00930131**. When file upload to bucket **sampledatab0093031**, lambda **extractFeatures** will be trigger. Lambda will extract the name entities from the **003.txt file**, create another text file named **003ne.txt** in the bucket **tagsb00930131** and store extracted name entities in the **001ne.txt** file in json format. Figure 42 shows the execution logs of lambda fucntion **extractFeatures.**

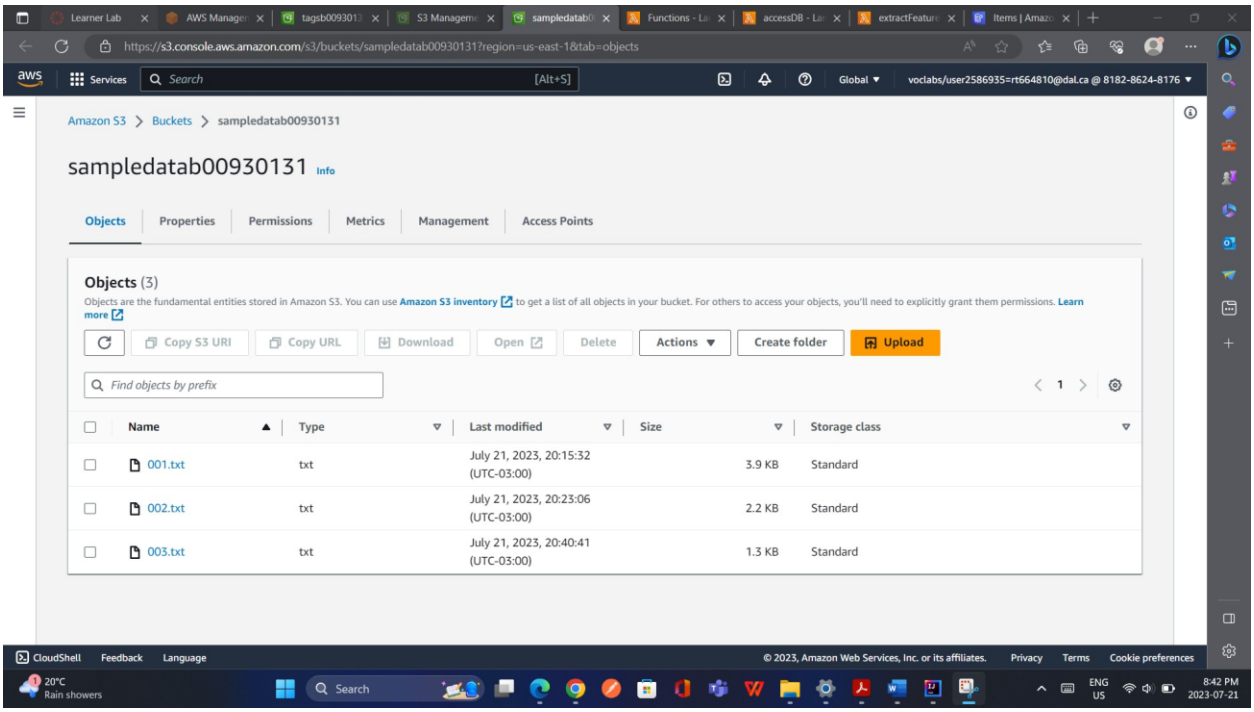

Figure 41 **003.txt** file uploaded to **sampeldatab00930131** successfully [2]

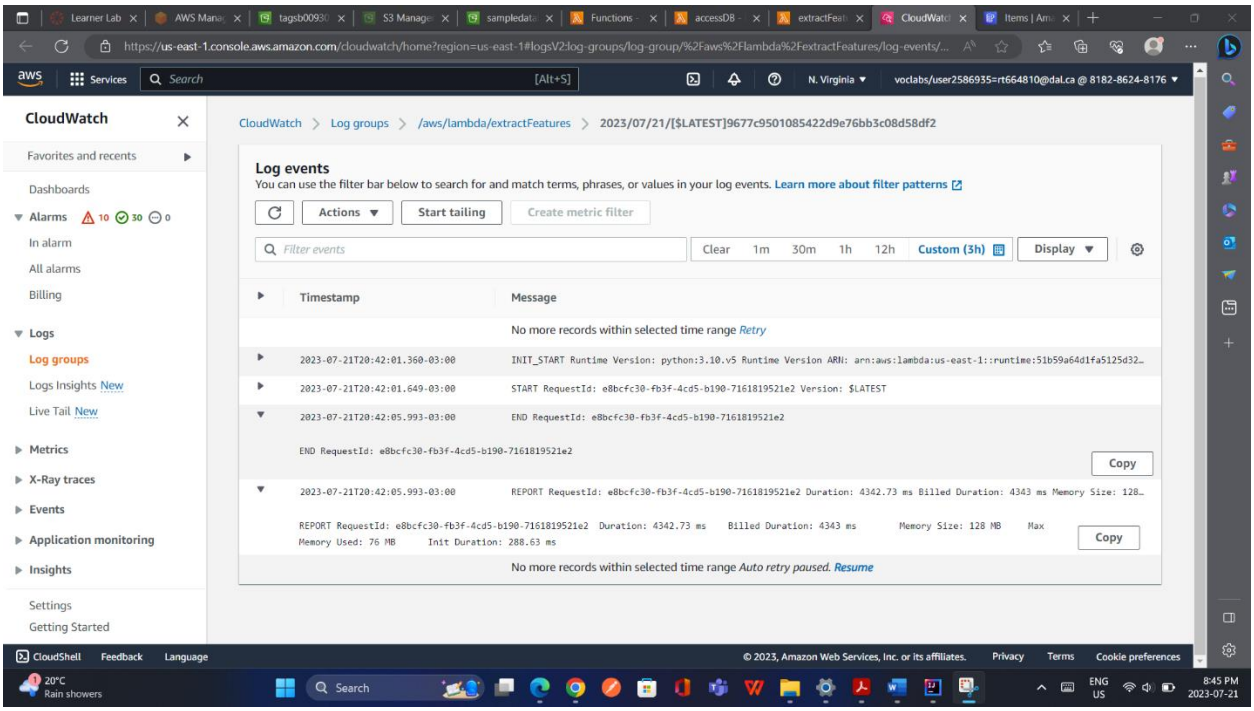

Figure 42 Execution logs of lambda **extractFeatures** to extract name entity of **003.txt** file from bucket **sampeldatab00930131** [4]

Figure 43 shows that file named **003ne.txt** created and uploaded to bucket **tagsb00930131** by lambda **extractFeatures**. When file will be available in bucket **tagsb00930131**, lambda **accessDB** will be trigger and store the name entities in DynamoDB **CSCI5410_Ass3_part2** table in key value pair. Figure 44 shows that lambda **accessDB** successfully uploaded key value pair in table **CSCI5410_Ass3_part2**. Figure 45 show the execution log of lambda **accessDB**.4

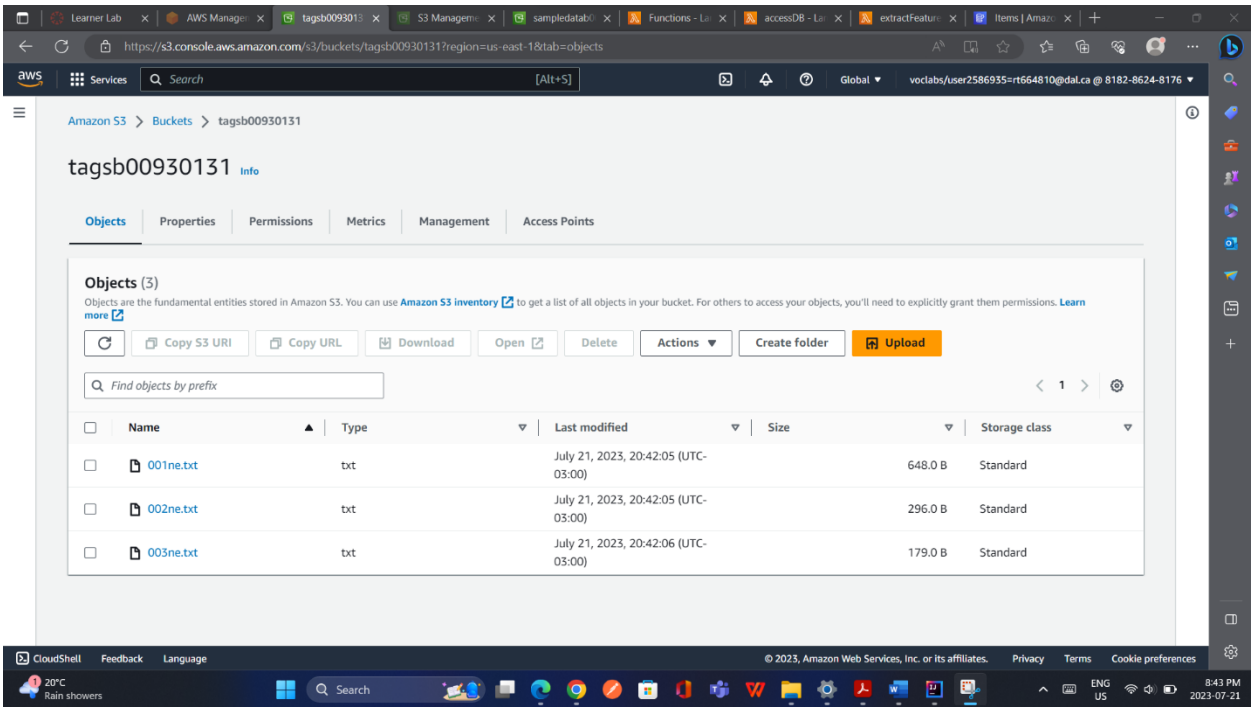

Figure 43 **003ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [2]

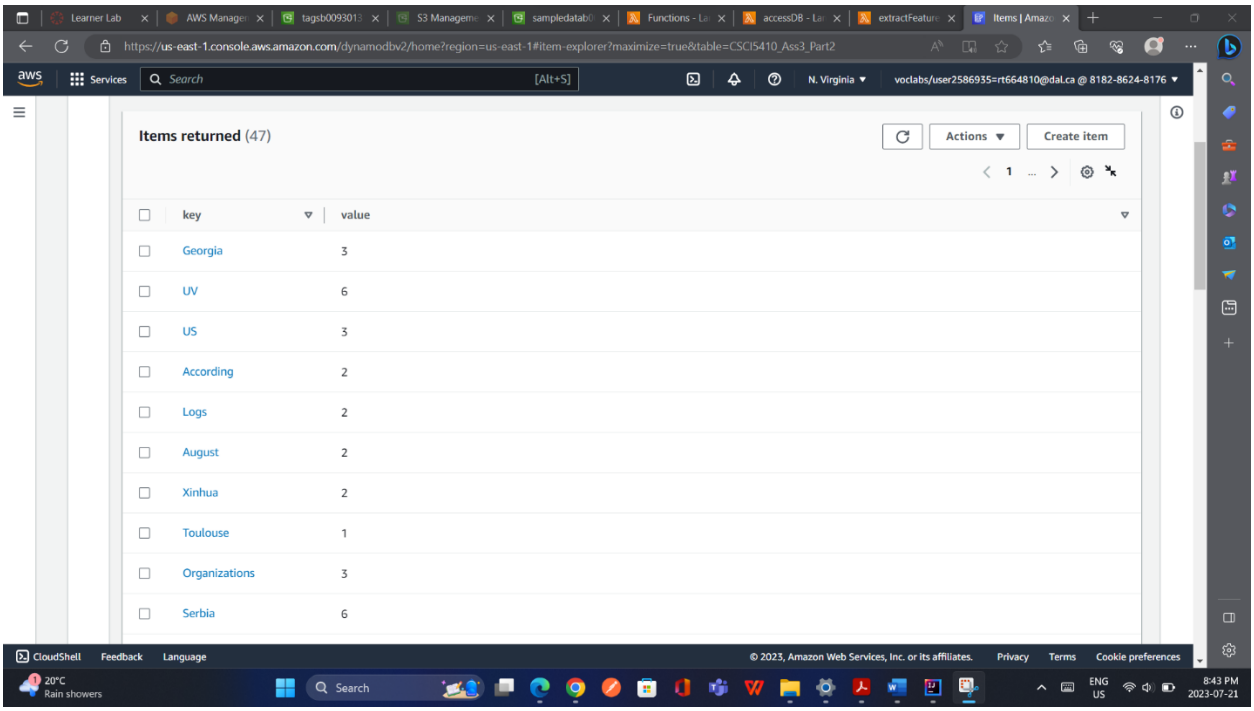

Figure 44 store key value from **003ne.txt file** to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1]

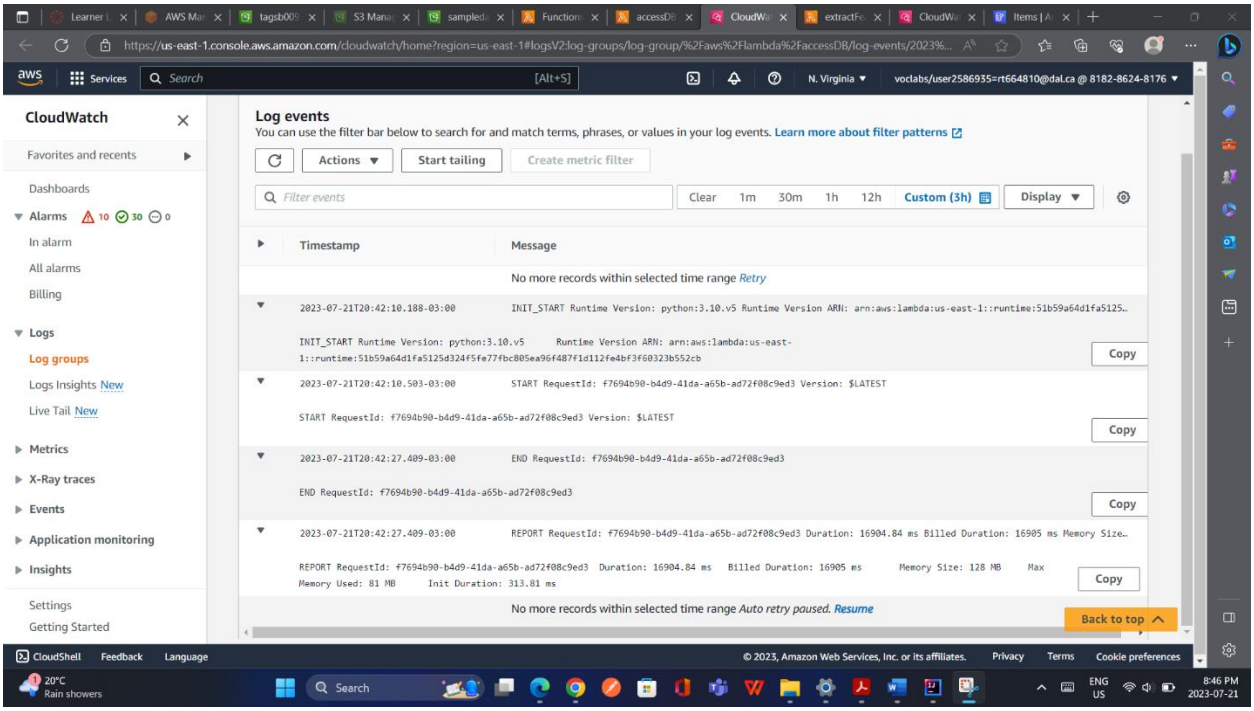

Figure 45 Execution logs of lambda **accessDB** to get the key value from **003ne.txt** file and store it to DynamoDB table [4]

Figure 46 shows that file named **004.txt** uploaded to bucket **sampledatab00930131** using Java SDK.

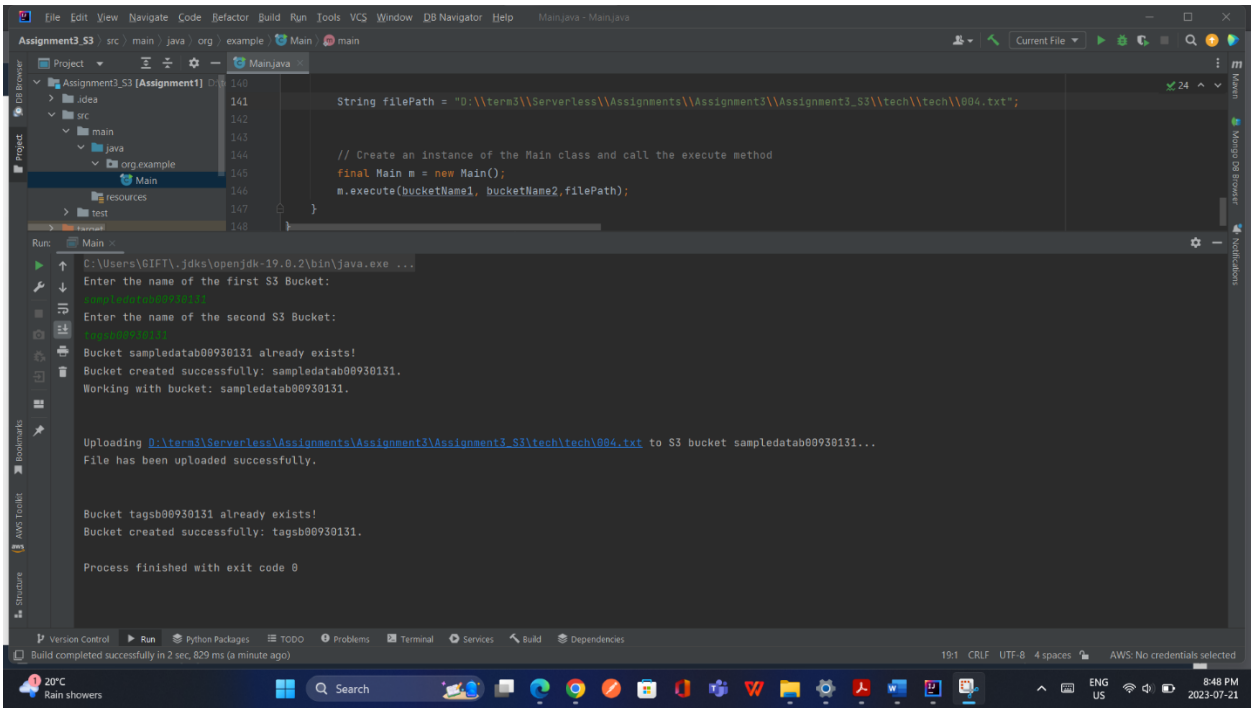

Figure 46 File named **004.txt** uploaded successfully to **sampledatab00930131** using Java SDK [5]

Figure 47 shows that file **003.txt** uploaded successfully to bucket **sampleb00930131**. When file upload to bucket **sampledatab0093031**, lambda **extractFeatures** will be trigger. Lambda will extract the name entities from the **003.txt file**, create another text file named **003ne.txt** in the bucket **tagsb00930131** and store extracted name entities in the **001ne.txt** file in json format. Figure 48 shows the execution logs of lambda fucntion **extractFeatures.**

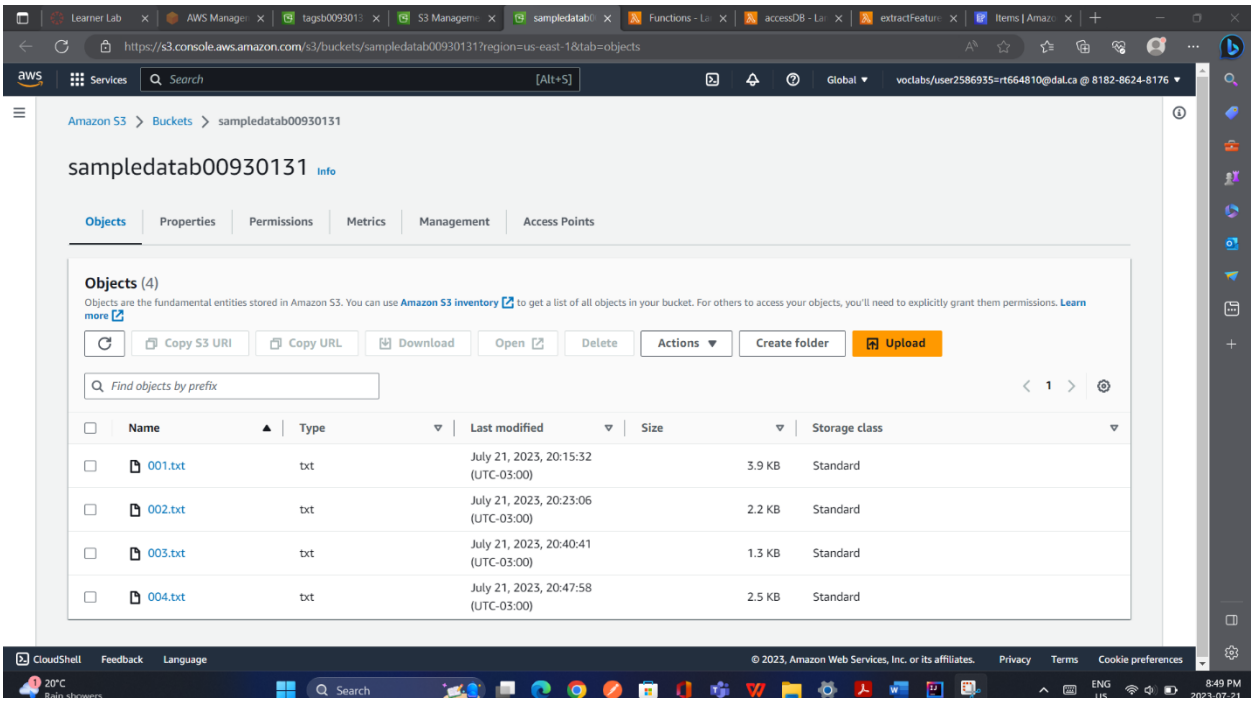

Figure 47 **004.txt** file uploaded to **sampeldatab00930131** successfully [2]

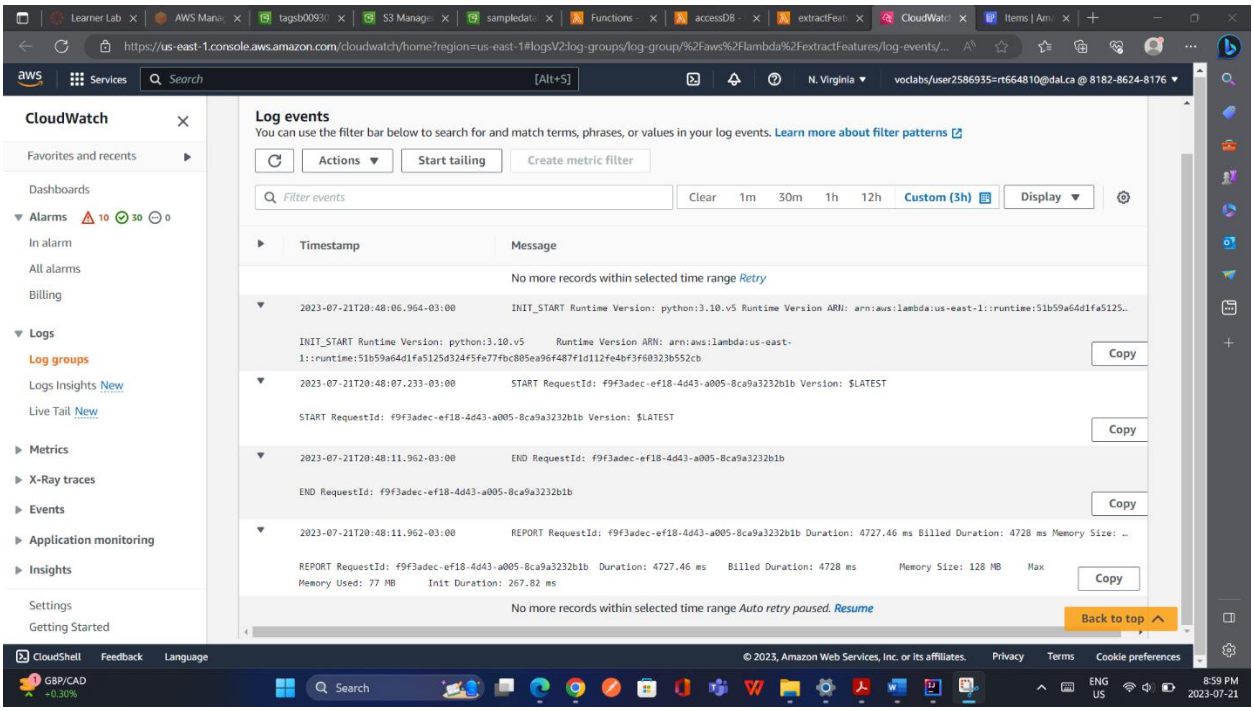

Figure 48 Execution logs of lambda **extractFeatures** to extract name entity of **004.txt** file from bucket **sampeldatab00930131** [4]

Figure 49 shows that file named **004ne.txt** created and uploaded to bucket **tagsb00930131** by lambda **extractFeatures**. When file will be available in bucket **tagsb00930131**, lambda **accessDB** will be trigger and store the name entities in DynamoDB **CSCI5410_Ass3_part2** table in key value pair. Figure 50 shows that lambda **accessDB** successfully uploaded key value pair in table **CSCI5410_Ass3_part2**. Figure 51 show the execution log of lambda **accessDB**.

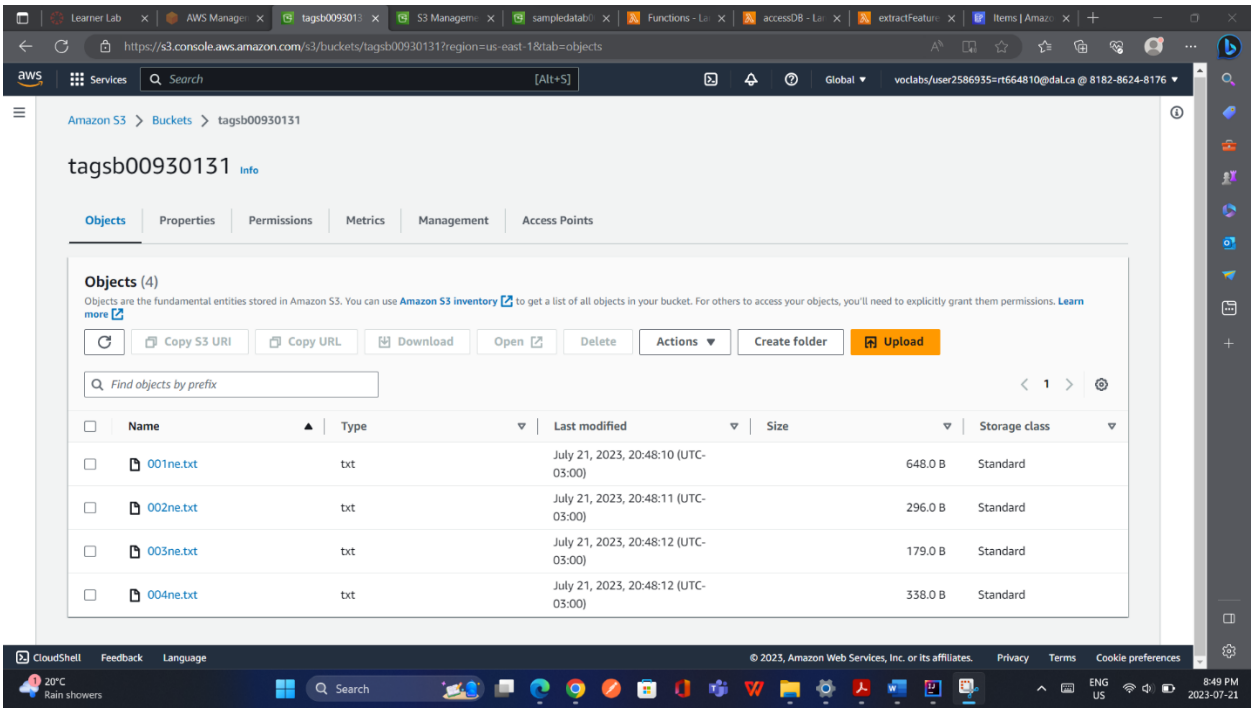

Figure 49 **004ne.txt** file uploaded by lambda fucntion **extractFeatures** to **tagsb00930131** successfully [2]

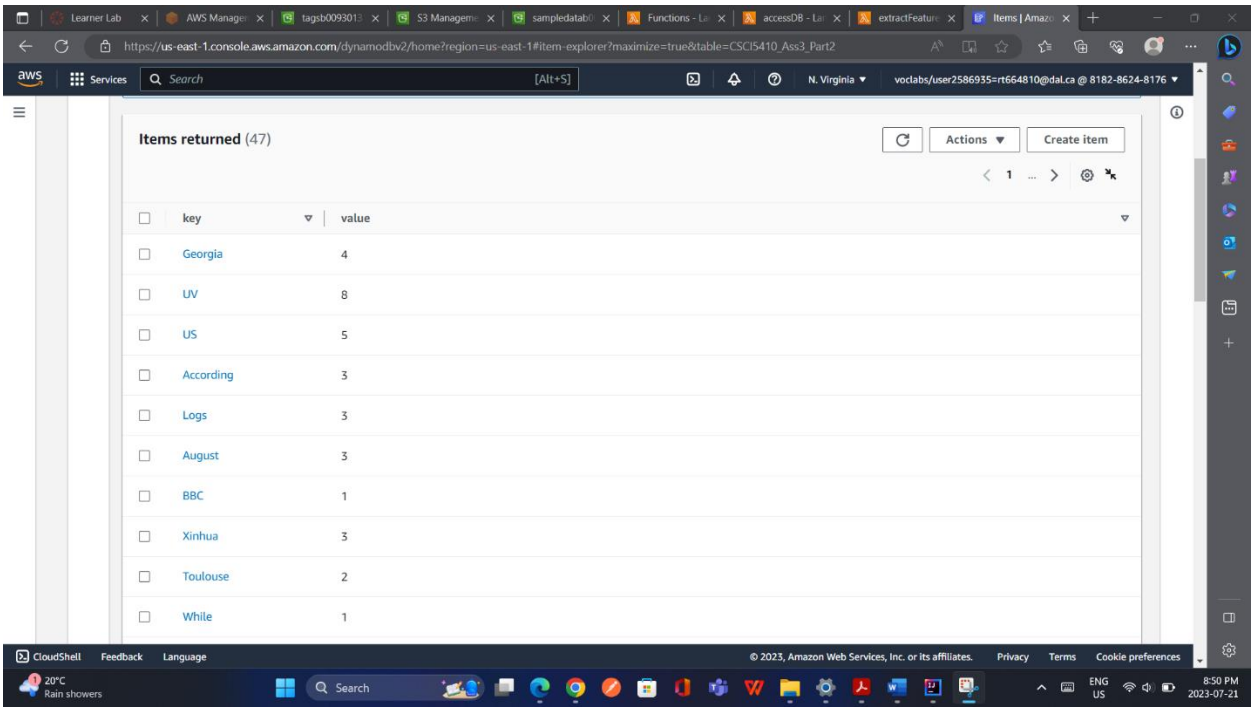

Figure 50 store key value from **004ne.txt file** to table **CSCI5410_Ass3_Part2** by lambda **accessDB** [1]

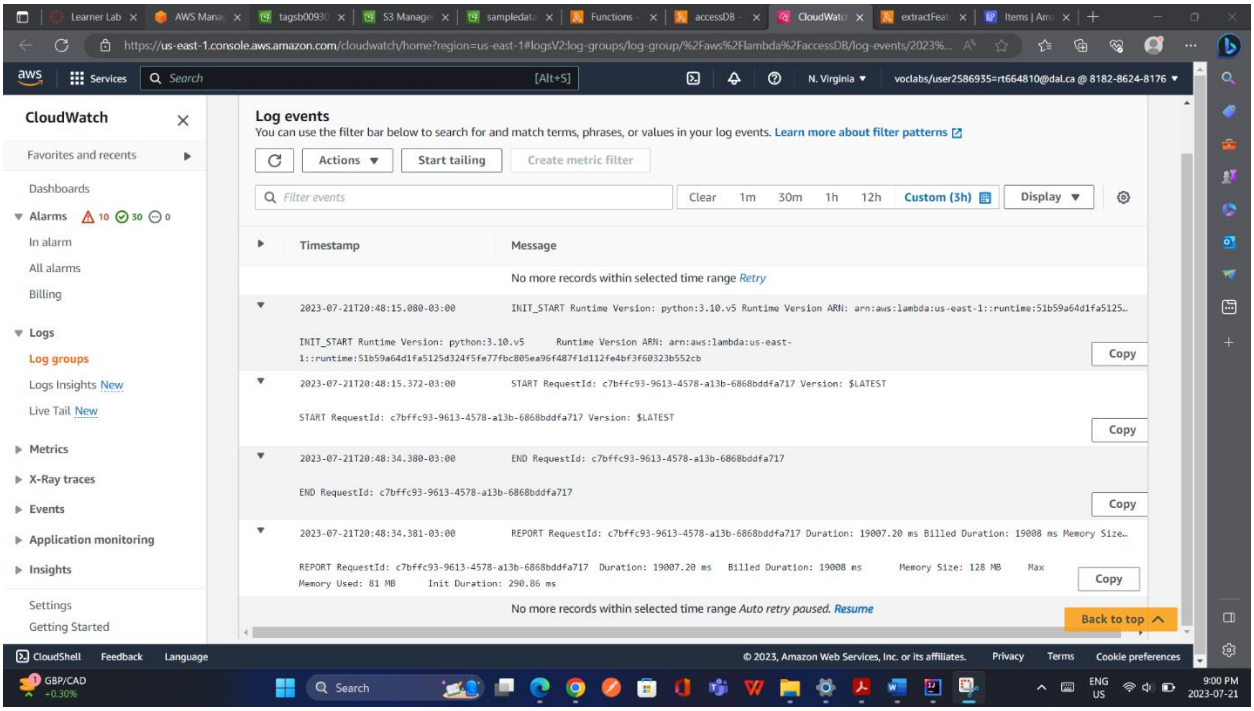

Figure 51 Execution logs of lambda **accessDB** to get the key value from **004ne.txt** file and store it to DynamoDB table [4]

Figures 52 and 53 show that all the text files are uploaded to bucket **sampledatab00930131**.

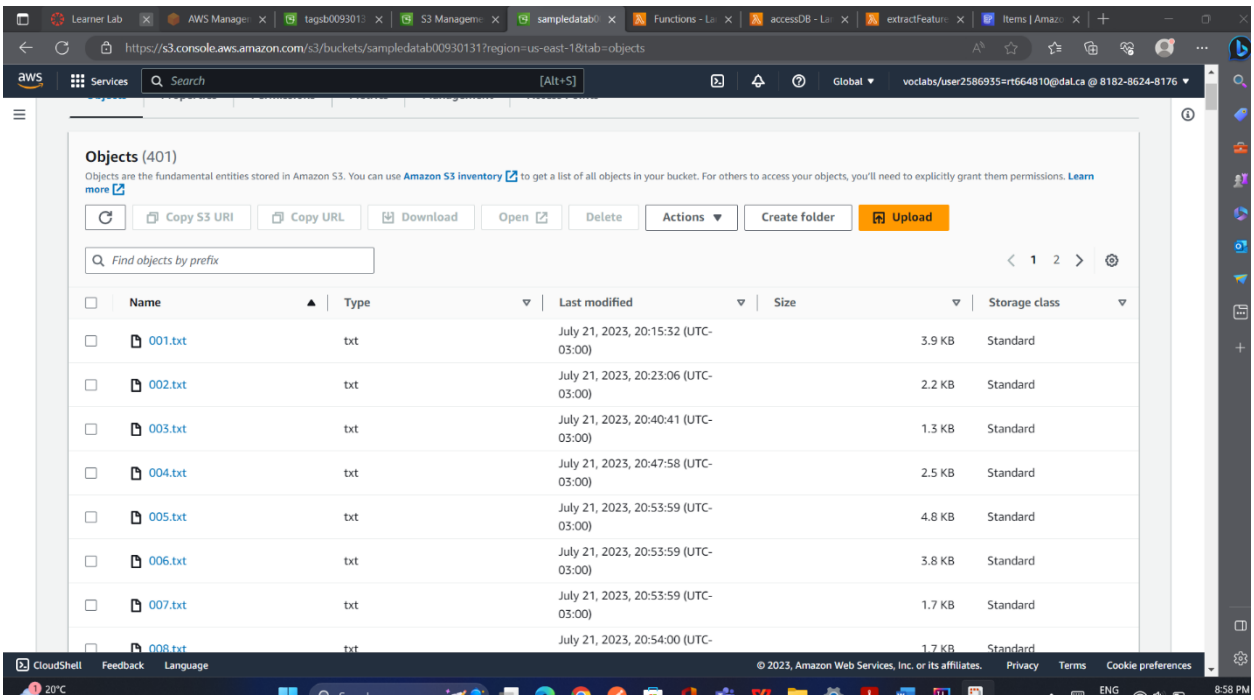

Figure 52 All files are uploaded to **samplebuacketb00930131** to extract the name entities [3]

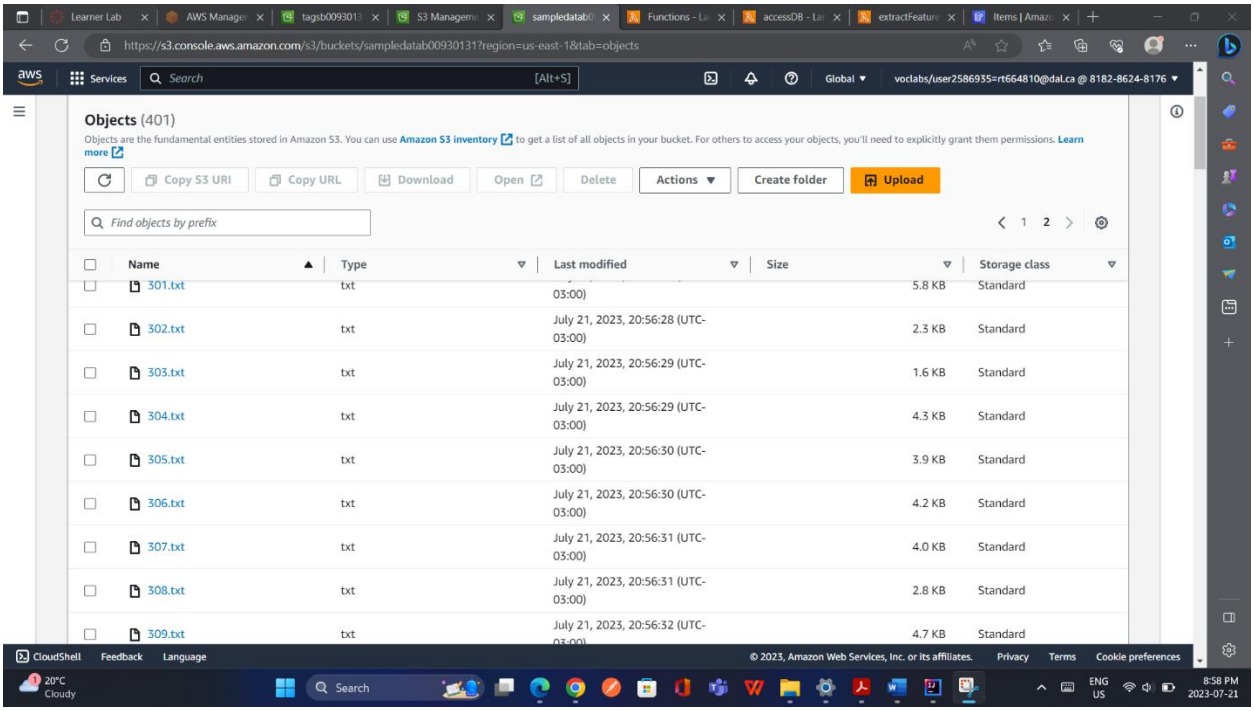

Figure 53 All files are uploaded to **samplebuacketb00930131** to extract the name entities [3]

Figures 54 and 55 show that all the file from **001ne.txt** to **400ne.txt** files are created and uploaded to bucket **tagsb00930131** by lambda **extractFeatures**.

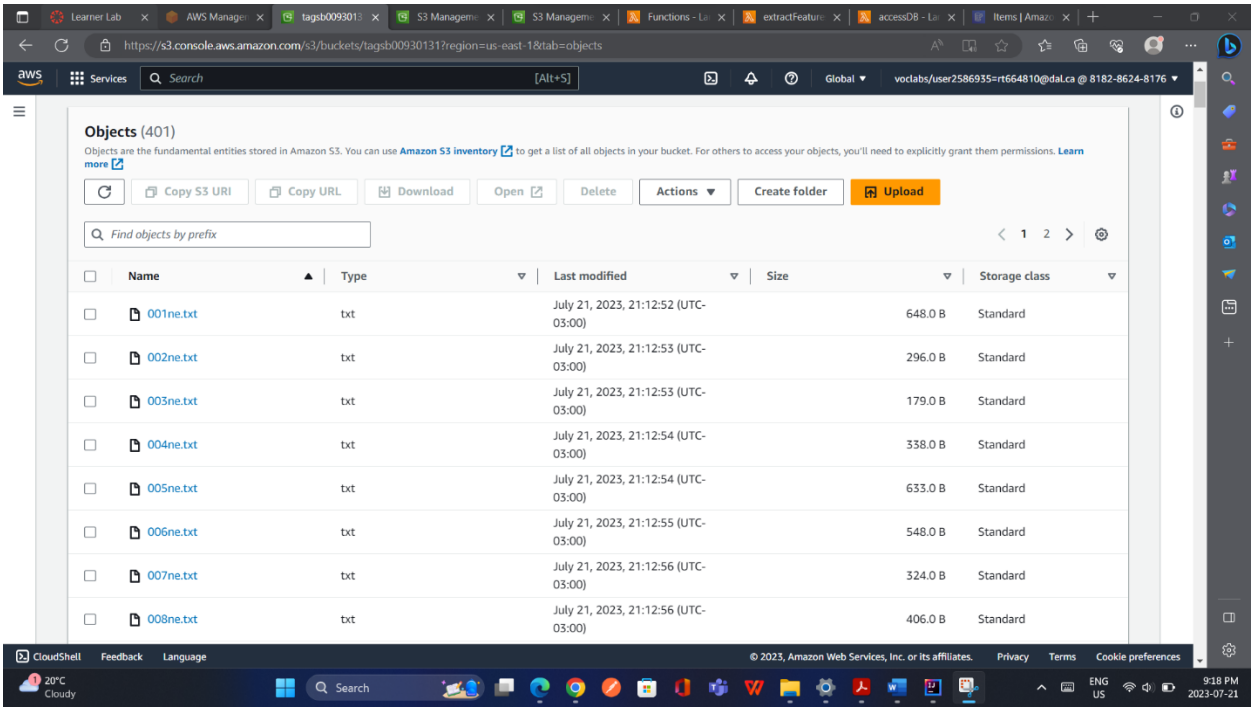

Figure 54 Lambda **accessDB** triggered, created all the files form **001ne.txt to 400ne.txt** and uploaded to **tagsb00930131** bucket [3]

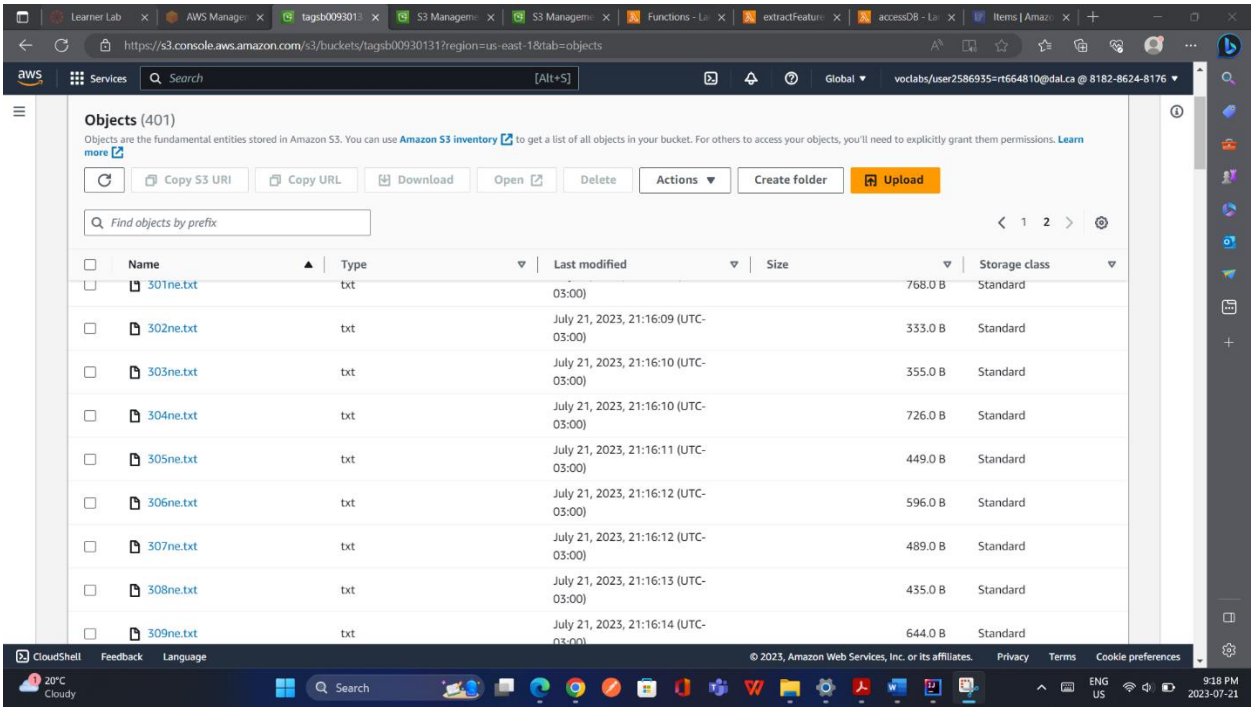

Figure 55 Lambda **accessDB** triggered, created all the files form **001ne.txt** to **400ne.txt** and uploaded to **tagsb00930131** bucket [3]

Figures 56,57 and 58 shows the DynamoDB table records which show key value pair of name entities.

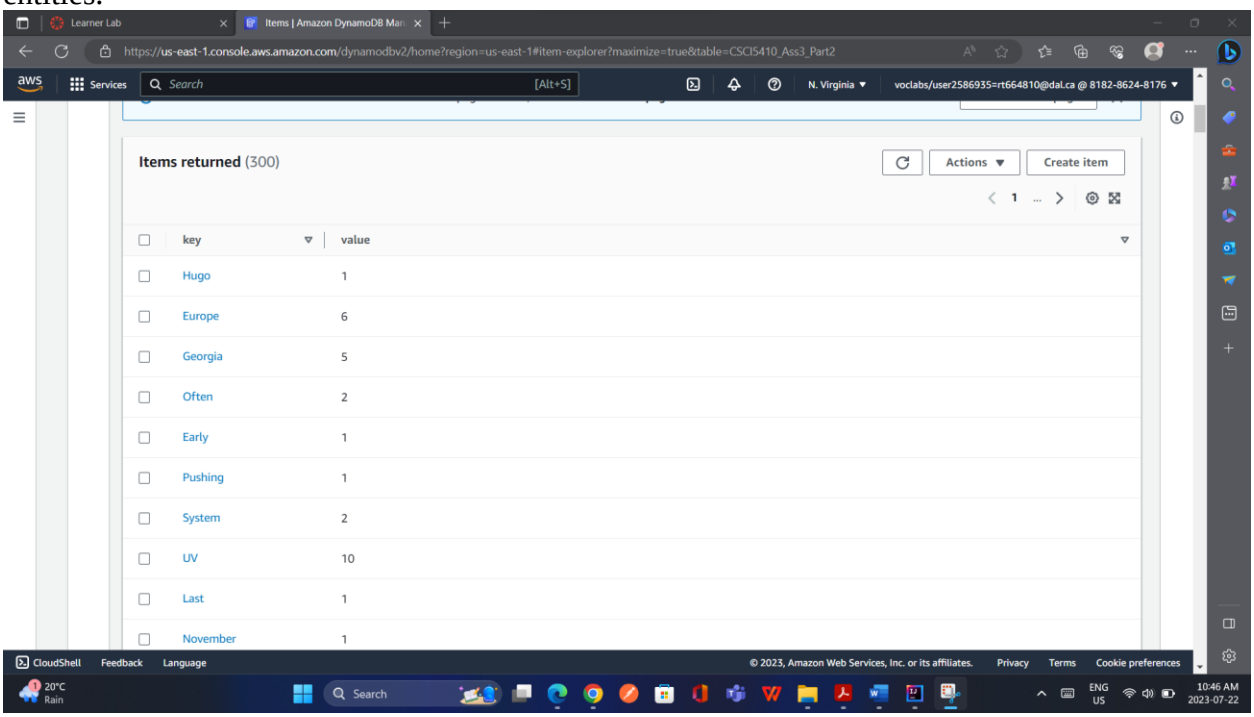

Figure 56 DynamoDB table with all key value stored using **accessDB** lambda [1]

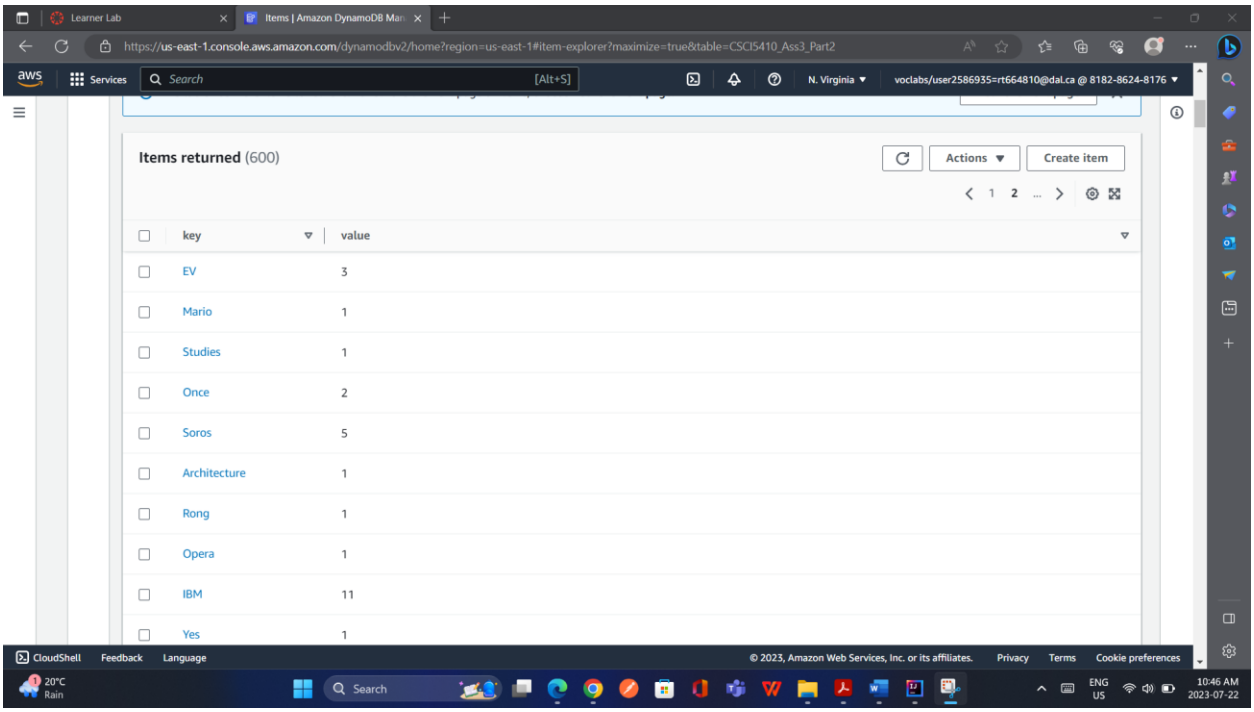

Figure 57 DynamoDB table with all key value stored using **accessDB** lambda [1]

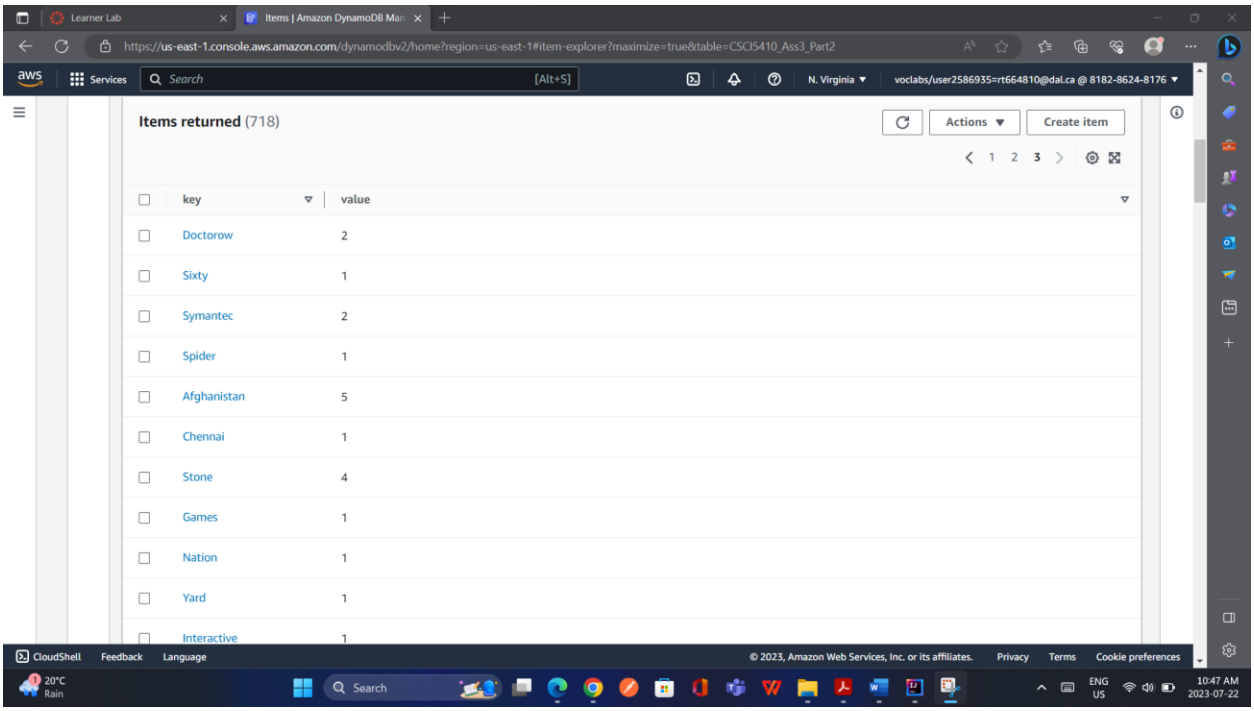

Figure 58 DynamoDB table with all key value stored using **accessDB** lambda [1]

Program Script

Java code to create S3 bucket and upload file in S3 bucket

Main.java [5]

```
package org.example;

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.nio.file.Paths;
import java.util.Scanner;

public final class Main {
    // Add your AWS credentials
    private static final String AWS_ACCESS_KEY = "YOUR AWS ACCESS KEY";
    private static final String AWS_SECRET_KEY = "YOUR AWS SECRET KEY";
    private static final String AWS_SESSION_TOKEN = "YOUR AWS SESSION TOKEN";
    private static final BasicSessionCredentials AWS_CREDENTIALS = new
BasicSessionCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY, AWS_SESSION_TOKEN);
    private final AmazonS3 amazonS3Client;
    private final Regions region = Regions.US_EAST_1;

    //Initialization of AWS S3 client
    public Main()
    {
        amazonS3Client = createAWSS3ClientBuilder();
    }

    public static void main(String[] args) {
        final Scanner sc = new Scanner(System.in);

        //Get the bucket names from console
        String bucketName1 = null;
        while (bucketName1 == null || bucketName1.trim().isEmpty()) {
            System.out.println("Enter the name of the first S3 Bucket:");
            bucketName1 = sc.nextLine();
        }

        String bucketName2 = null;
        while (bucketName2 == null || bucketName2.trim().isEmpty()) {
            System.out.println("Enter the name of the second S3 Bucket:");
            bucketName2 = sc.nextLine();
        }

        // Initialize the file path to upload the file to bucket
        String filePath =
"D:\\term3\\Serverless\\Assignments\\Assignment3\\Assignment3_S3\\tech\\tech\
```

```
\003.txt";


        final Main m = new Main();
        m.execute(bucketName1, bucketName2, filePath);
    }

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

    // Method getBucketIfExists is used to check that bucket is already
exists or not
    private Bucket getBucketIfExists(String bucketName) {
        try {
            return amazonS3Client.listBuckets().stream().filter(bucket ->
bucket.getName().equals(bucketName)).findFirst().orElse(null);
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Error occurred while retrieving bucket: " +
e.getMessage());
            return null;
        }
    }

    //Create a S3 bucket if bucket does not exist
    private Bucket createBucketIfNotExists(String bucketName) {
        try {
            if (amazonS3Client.doesBucketExistV2(bucketName)) {
                System.out.println("Bucket " + bucketName + " already
exists!");
                return getBucketIfExists(bucketName);
            } else {
                System.out.println("Creating bucket " + bucketName + "...");
                return amazonS3Client.createBucket(bucketName);
            }
        } catch (AmazonS3Exception e) {
            e.printStackTrace();
            System.err.println(e.getErrorMessage());
            return null;
        }
    }

    // Method uploadObject is used to upload file in the bucket
    private void uploadObject(String bucketName, String filePath) {
        File file = new File(filePath);
        String keyName = Paths.get(filePath).getFileName().toString();
        System.out.println("Uploading " + filePath + " to S3 bucket " +
bucketName + "...");
        try {
            amazonS3Client.putObject(bucketName, keyName, file);
            System.out.println("File has been uploaded successfully.");
            System.out.println();
            System.out.println();
        } catch (AmazonServiceException e) {
            e.printStackTrace();
```

```
            System.err.println(e.getMessage());
        }
    }

    public void execute(String bucketName1, String bucketName2, String
filePath) {
        try {
            // Create the first bucket or get the existing one
            Bucket s3Bucket1 = createBucketIfNotExists(bucketName1);
            if (s3Bucket1 != null) {
                System.out.println("Bucket created successfully: " +
s3Bucket1.getName() + ".");
                System.out.println("Working with bucket: " +
s3Bucket1.getName() + ".");
                System.out.println();
                System.out.println();

                uploadObject(s3Bucket1.getName(), filePath);
            } else {
                System.err.println("Error working with bucket: " +
bucketName1 + ".");
                return;
            }

            Bucket s3Bucket2 = createBucketIfNotExists(bucketName2);
            if (s3Bucket2 != null) {
                System.out.println("Bucket created successfully: " +
s3Bucket2.getName() + ".");


            } else {
                System.err.println("Error working with bucket: " +
bucketName2 + ".");
            }


        } catch (Exception e) {
            e.printStackTrace();
            System.err.println(e.getMessage());
        }
    }
}

/*
I referred below source to create DynamoDB table using java SDK as I reused
my assignment  1 code.
References:

[1] "Work with Amazon S3 - AWS SDK for Java 2.x," docs.aws.amazon.com.
[Online]. Available:
https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-
s3.html. [Accessed: 23 July 2023].

[2] R. Katrodiya, "CSCI 5410 Assignment 1." Summer Term, Dalhousie
University, [Online document], 2023. [Accessed: 23 July 2023].
 */
```

Java code to create table in DynamoDB

Main.java [5]

```
package org.example;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.*;
import com.amazonaws.services.dynamodbv2.util.TableUtils;

import java.util.List;
import java.util.Scanner;

public final class Main {
    // Add your AWS credentials
    private static final String AWS_ACCESS_KEY = "YOUR AWS ACCESS KEY";
    private static final String AWS_SECRET_KEY = "YOUR AWS SECRET KEY";
    private static final String AWS_SESSION_TOKEN = "YOUR AWS SESSION TOKEN";
    private static final BasicSessionCredentials AWS_CREDENTIALS = new
BasicSessionCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY, AWS_SESSION_TOKEN);
    private final AmazonDynamoDB dynamoDBClient;
    private final Regions region = Regions.US_EAST_1;

    // Initialize AWS DynamoDB client
    public Main() {

        dynamoDBClient = createDynamoDBClient();
    }

    public static void main(String[] args) {
        final Scanner sc = new Scanner(System.in);

        // Get the table and primary key attribute from console
        String tableName = null;
        while (tableName == null || tableName.trim().isEmpty()) {
            System.out.println("Enter the name of the DynamoDB table:");
            tableName = sc.nextLine();
        }

        String primaryKey = null;
        while (primaryKey == null || primaryKey.trim().isEmpty()) {
            System.out.println("Enter the primary key attribute name:");
            primaryKey = sc.nextLine();
        }

        final Main m = new Main();
        m.execute(tableName, primaryKey);
    }

    // Initialize Amazon DynamoDB client
    private AmazonDynamoDB createDynamoDBClient() {
```

```
        return AmazonDynamoDBClientBuilder.standard().withCredentials(new
AWSStaticCredentialsProvider(AWS_CREDENTIALS)).withEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration("dynamodb.us-east-1.amazonaws.com",
region.getName())).build();
    }

    // Method createDynamoDBTable is used to create table in Amazon Dynamodb
    private void createDynamoDBTable(String tableName, String primaryKey) {
        try {
            List<AttributeDefinition> attributeDefinitions = List.of(new
AttributeDefinition(primaryKey, ScalarAttributeType.S));

            List<KeySchemaElement> keySchema = List.of(new
KeySchemaElement(primaryKey, KeyType.HASH));

            ProvisionedThroughput provisionedThroughput = new
ProvisionedThroughput(1L, 1L);

            CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName).withAttributeDefinitions(attrib
uteDefinitions).withKeySchema(keySchema).withProvisionedThroughput(provisione
dThroughput);

            CreateTableResult createTableResult =
dynamoDBClient.createTable(createTableRequest);

            TableUtils.waitUntilActive(dynamoDBClient, tableName);

            System.out.println("DynamoDB table created successfully: " +
createTableResult.getTableDescription().getTableName());
        } catch (AmazonServiceException e) {
            e.printStackTrace();
            System.err.println("Error occurred while creating DynamoDB table:
" + e.getMessage());
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    public void execute(String tableName, String primaryKey) {
        try {
            createDynamoDBTable(tableName, primaryKey);
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println(e.getMessage());
        }
    }
}



/*
I referred below source to create DynamoDB table using java SDK as I reused
my assignment 1 code.
References:

[1] "Work with DynamoDB - AWS SDK for Java 2.x," docs.aws.amazon.com.
```

[Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-dynamodb.html. [Accessed: 23 July 2023].

[2] R. Katrodiya, "CSCI 5410 Assignment 1." Summer Term, Dalhousie University, [Online document], 2023. [Accessed: 23 July 2023].
 */

Lambda function code of extarctFeatures in python
lambda_function.py [3]

```
import boto3
import json
import re

def lambda_handler(event, context):

    # get the bucket name from event test input
    bucket_name = event['Records'][0]['s3']['bucket']['name']

    object_list = get_object_list(bucket_name)

    for file_info in object_list:
        file_key = file_info['Key']

        # This method is used to create S3 client
        s3_client = boto3.client('s3')

        # It will get the content from file of S3 bucket
        response = s3_client.get_object(Bucket=bucket_name, Key=file_key)
        file_content = response['Body'].read().decode('utf-8')

        # This mehtod is used to extract name entities from the file
        named_entities = extract_named_entities(file_content)

        # It will create json object of the extarcted entities
        json_object = {file_key + 'ne': named_entities}
        json_string = json.dumps(json_object)

        # It will cretae another file and store the json object in that file
        new_bucket_name = 'tagsb00930131'
        new_file_key = file_key[:-4] + 'ne.txt'

        # This method will upload the newly created file to bucket named
tagsb00930131
        s3_client.put_object(Body=json_string, Bucket=new_bucket_name,
Key=new_file_key)
```

```python
    return {
        'statusCode': 200,
        'body': 'Named entities extracted and saved as JSON files for all files
in the bucket.'
    }

def get_object_list(bucket_name):

    s3_client = boto3.client('s3')

    response = s3_client.list_objects_v2(Bucket=bucket_name)
    object_list = response['Contents'] if 'Contents' in response else []
    return object_list

# This method is used to extract name entities from the file
def extract_named_entities(file_content):

    words = re.findall(r'\b[A-Z][a-zA-Z]*\b', file_content)

    named_entities = {}
    for word in words:
        named_entities[word] = named_entities.get(word, 0) + 1

    return named_entities

# References:

# [1] “Building Lambda functions with Python - AWS Lambda,” docs.aws.amazon.com.
# [Online]. Available:  https://docs.aws.amazon.com/lambda/latest/dg/lambda-
python.html. [Accessed: 23 July 2023].
# [2] “Work with Amazon S3 - AWS SDK for Java 2.x,” docs.aws.amazon.com.
# [Online]. Available:   https://docs.aws.amazon.com/sdk-for-
java/latest/developer-guide/examples-s3.html. [Accessed: 23 July 2023].
```

Lambda function code of accessDB in python
lambda_function.py [3]

```
import boto3
import json

def lambda_handler(event, context):

    # Get the name of the S3 bucket from the test event data
    bucket_name = event['Records'][0]['s3']['bucket']['name']

    object_list = get_object_list(bucket_name)

    for file_info in object_list:
        file_key = file_info['Key']

        # Initialize Amazon S3 client
        s3_client = boto3.client('s3')

        # This is mehtod id used to get the content from the S3 bucket
        response = s3_client.get_object(Bucket=bucket_name, Key=file_key)
        file_content = response['Body'].read().decode('utf-8')

        named_entities_json = json.loads(file_content)

        # This method is used to update the DynamoDB table
        update_dynamodb(named_entities_json)

    return {
        'statusCode': 200,
        'body': 'DynamoDB table updated.'
    }

def get_object_list(bucket_name):

    # Initialize Amazon S3 client
    s3_client = boto3.client('s3')

    response = s3_client.list_objects_v2(Bucket=bucket_name)

    object_list = response['Contents'] if 'Contents' in response else []
    return object_list

def update_dynamodb(named_entities_json):
    # This method will create DynamoDB resource
```

```
    dynamodb = boto3.resource('dynamodb')

    #It will get the DynamoDB name from tables of DynamoDB
    table = dynamodb.Table('CSCI5410_Ass3_Part2')

    for key, value in named_entities_json.items():
        for entity, entity_value in value.items():
            # IT will chekc that entity is already exist or not
            response = table.get_item(
                Key={
                    'key': entity
                }
            )
            item = response.get('Item')

            # If the entity exists, it will update value
            if item:
                table.update_item(
                    Key={
                        'key': entity
                    },
                    UpdateExpression='SET #val = #val + :inc',
                    ExpressionAttributeNames={
                        '#val': 'value'
                    },
                    ExpressionAttributeValues={
                        ':inc': entity_value
                    }
                )
            # If  entity does not exist, it will  create a new item with its
value
            else:
                table.put_item(
                    Item={
                        'key': entity,
                        'value': entity_value
                    }
                )


# References:

# [1] “Building Lambda functions with Python - AWS Lambda,” docs.aws.amazon.com.
# [Online]. Available:  https://docs.aws.amazon.com/lambda/latest/dg/lambda-
python.html. [Accessed: 23 July 2023].
```

```
# [2] “Work with DynamoDB - AWS SDK for Java 2.x,” docs.aws.amazon.com.
# [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-
guide/examples-dynamodb.html. [Accessed: 23 July 2023].
```

Test Cases

Figures 59, 60,61,63,64,65 and 66 show the test cases to test the lambda fucntion which extract entities form 003.txt file of bucket **sampledatab00930131**, created another file named 003ne.txt and uploaded that file to another bucket named tagsb00930131. When second lambda **accessDB** tested, it calculates the key value pair from the 003ne.txt file and store in DynamoDB.

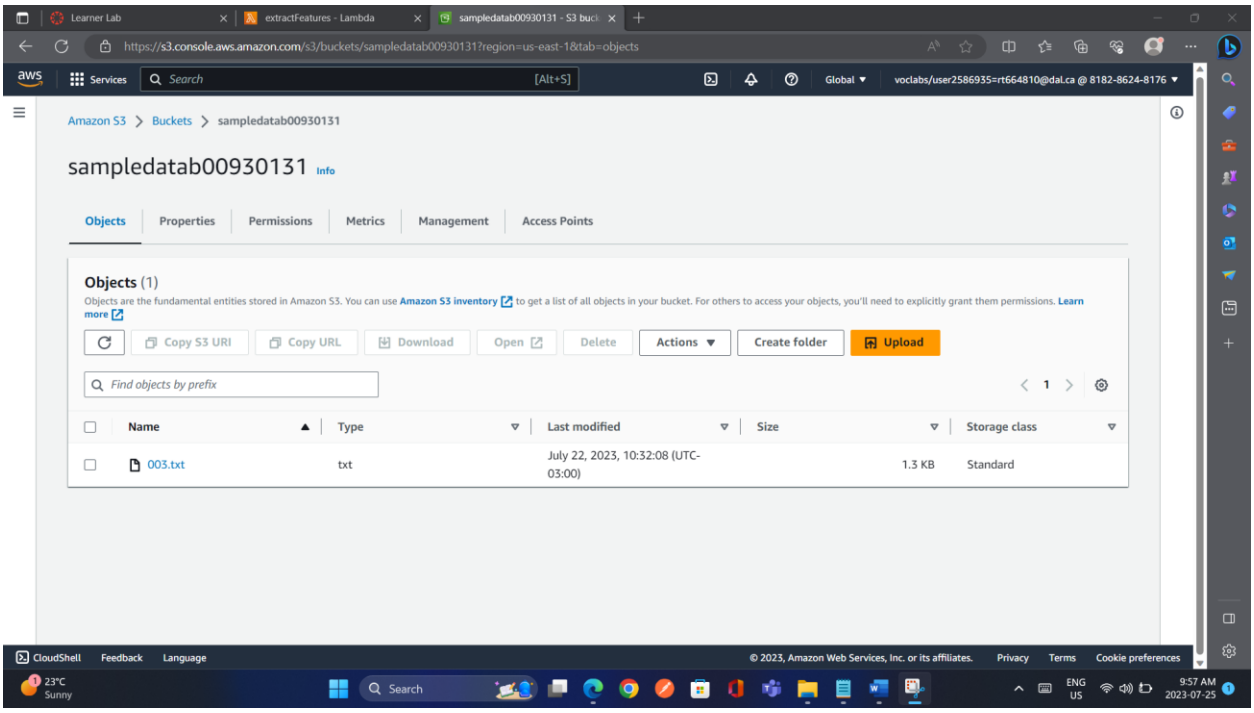

Figure 59 **File 003.txt** uploaded to **sampledatab00930131** bucket [2]

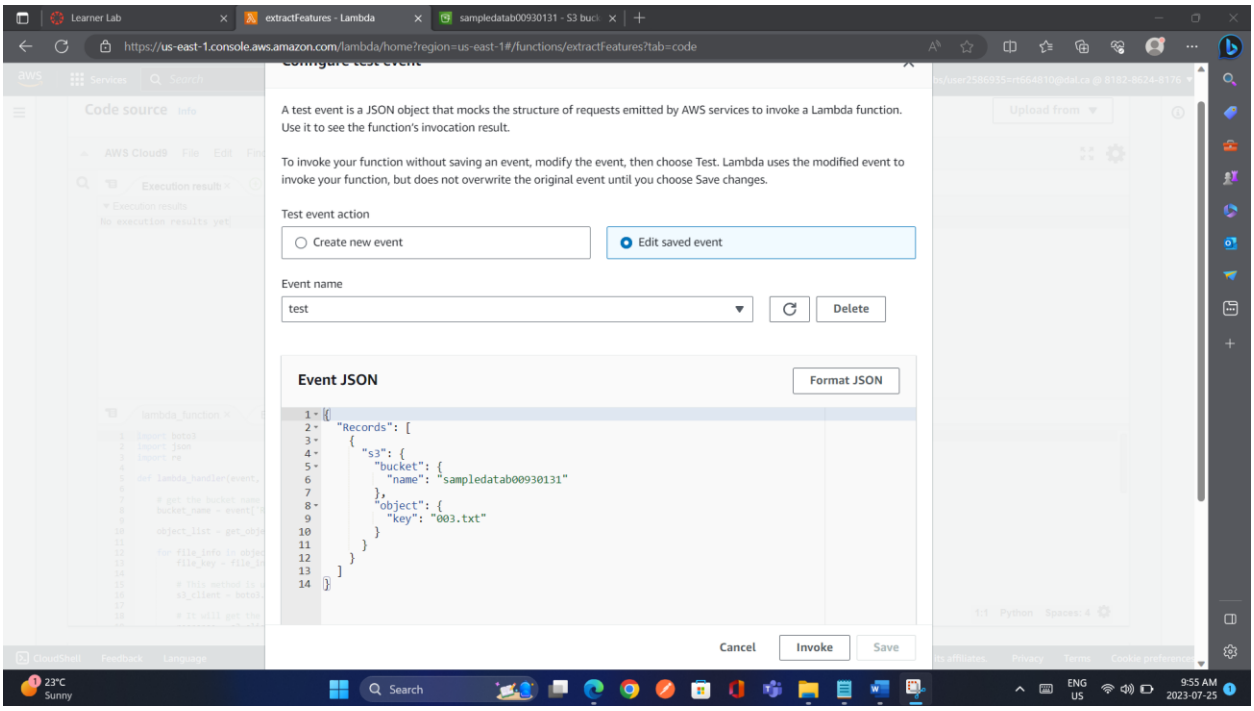

Figure 60 test event configured to test the lambda **extractFeatures [3]**

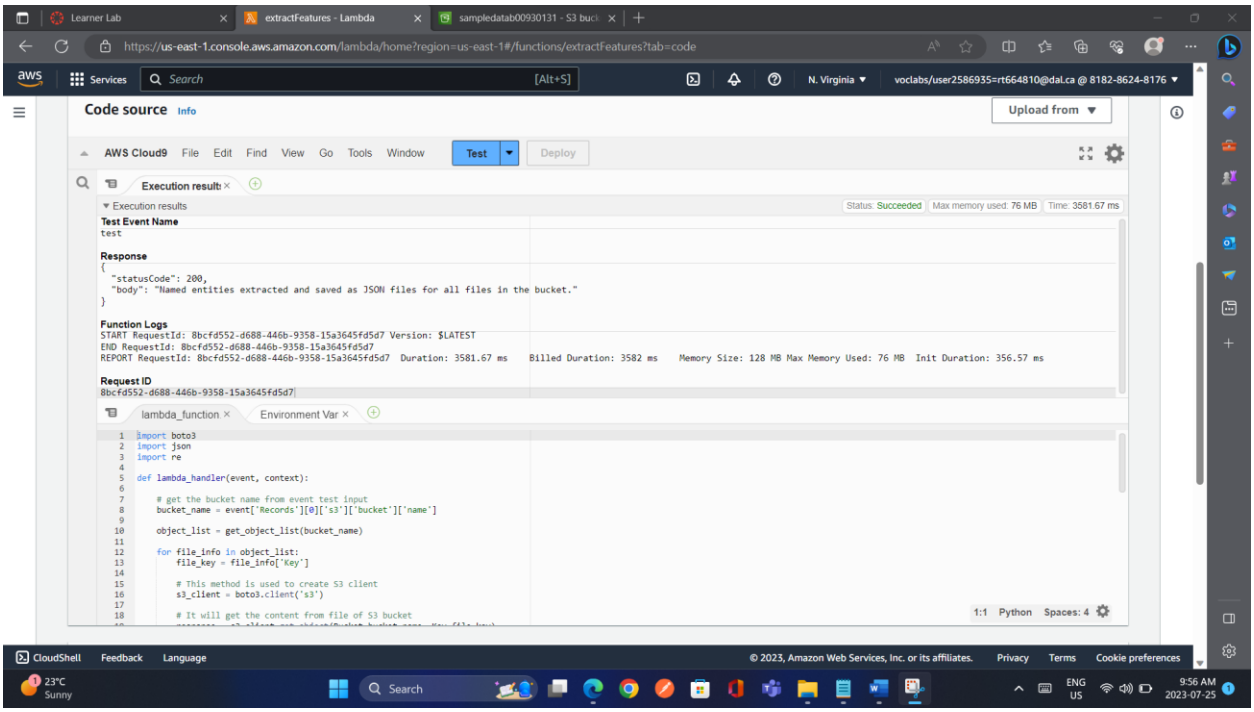

Figure 61 lambda **extarctFeatures** tested with 200 status code [3]

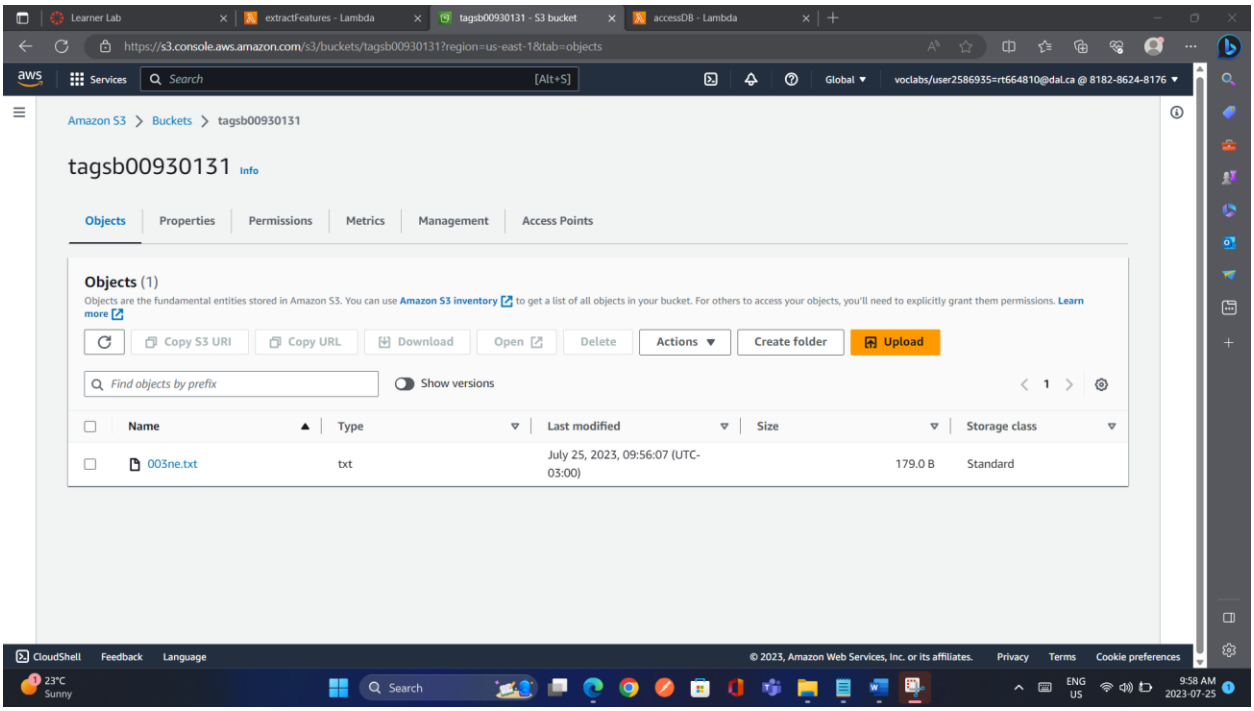

Figure 62 On successfully execution of lambda, **003ne.txt** file is created and uploaded to bucket **tagsb00930131** [2]

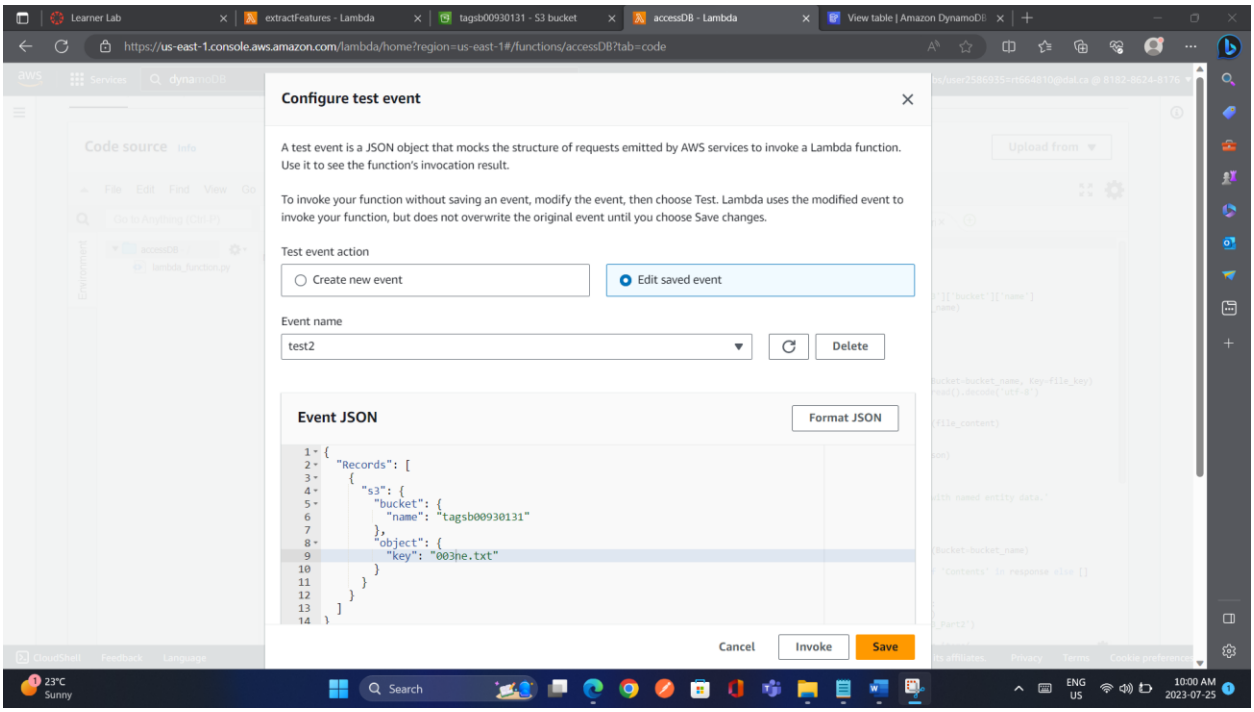

Figure 63 test event configured for second lambda **accessDB** [3]

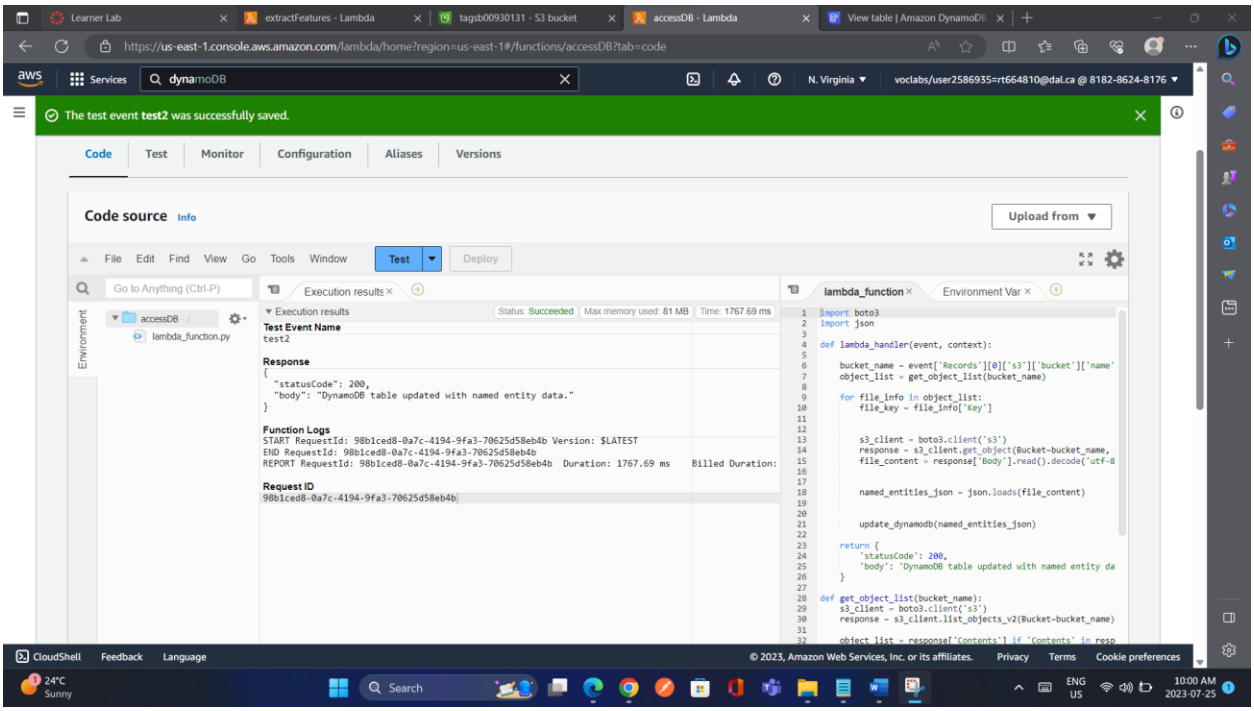

Figure 64 lambda **accessDB** tested with 200 status code [3]

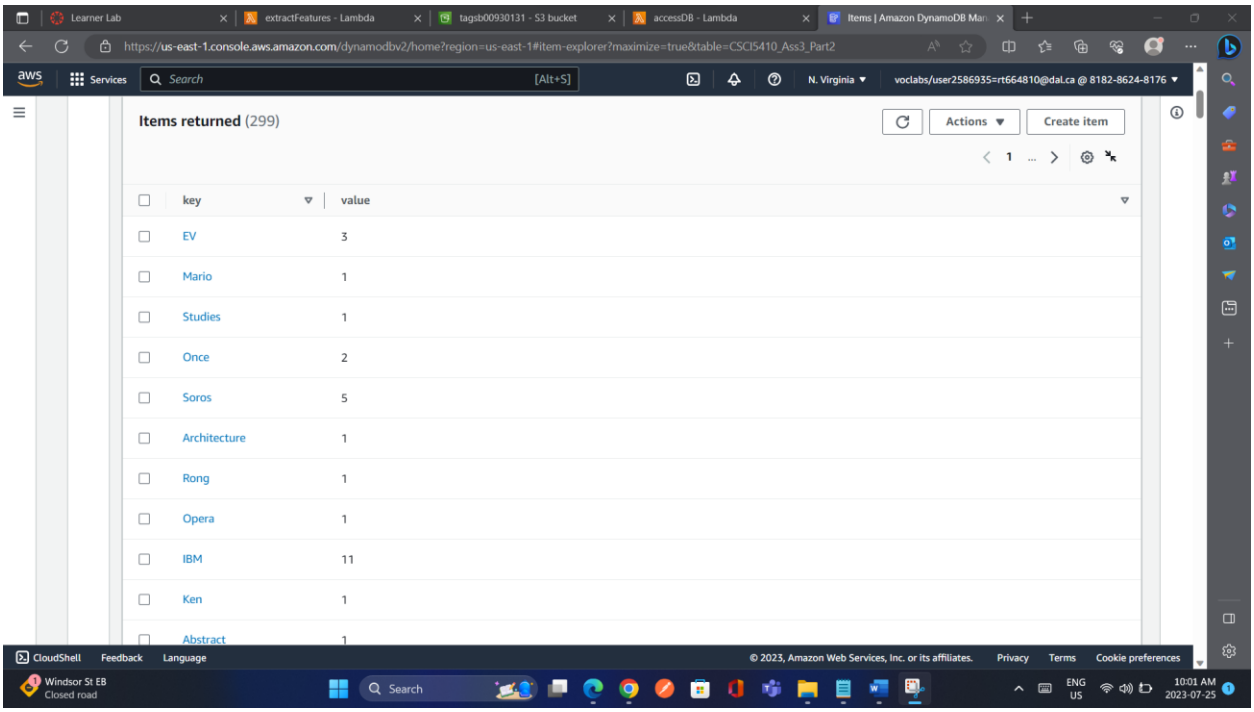

Figure 65 On successful execution of lambda **accessDB**, key value pair is stored in **DynamoDB** [1]

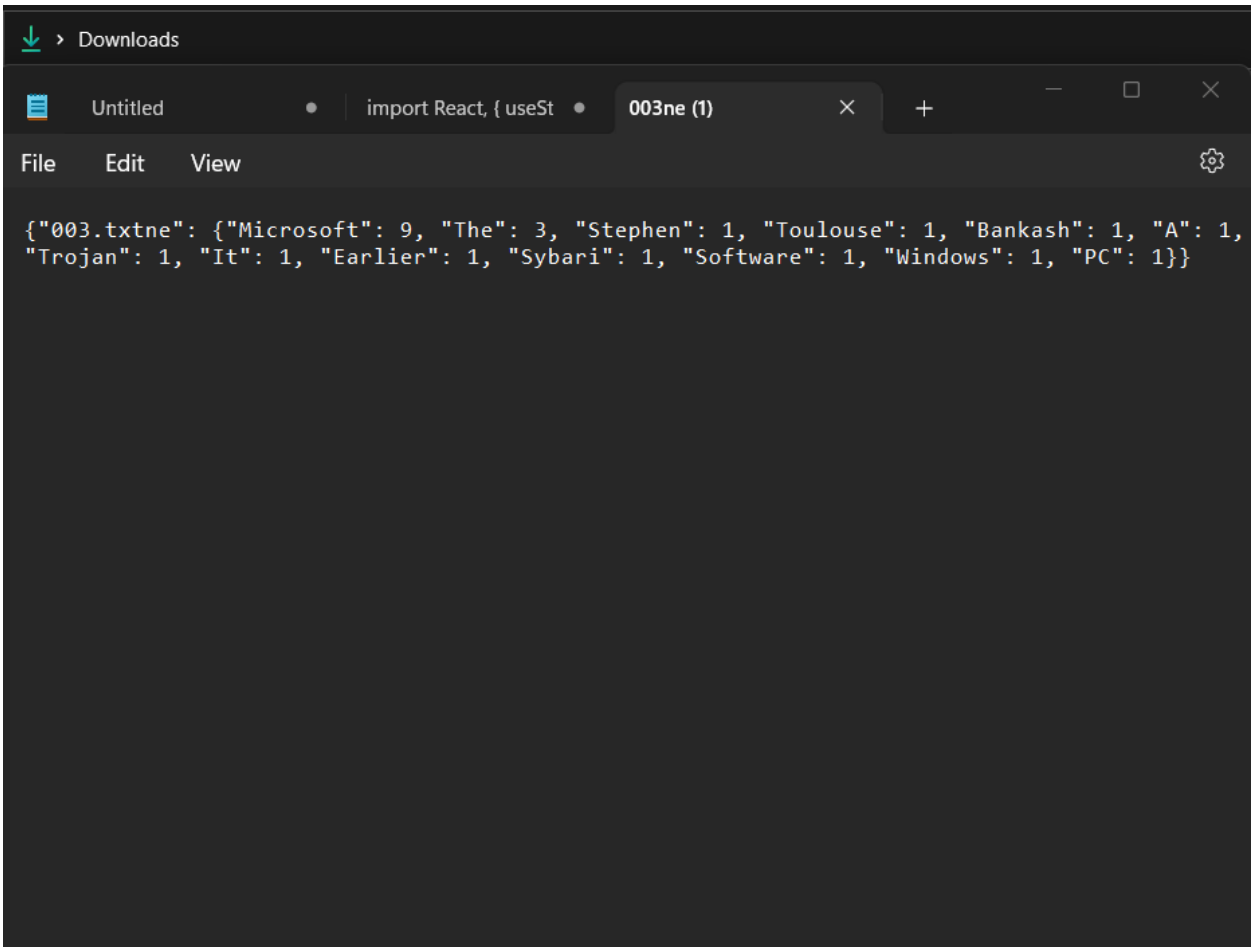

Figure 66 Output file named 003be.txt

References

[1] AWS, “Amazon DynamoDB - Overview,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed: 23 July 2023].

[2] AWS, “Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service,” Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 23 July 2023].

[3] Amazon, “AWS Lambda – Serverless Compute - Amazon Web Services,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 23 July 2023].

[4] AWS, “Amazon CloudWatch - Application and Infrastructure Monitoring,” Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/cloudwatch/. [Accessed: 23 July 2023].

[5] JetBrains, “IntelliJ IDEA,” JetBrains, 2019. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed: 23 July 2023].

[6] “Work with Amazon S3 - AWS SDK for Java 2.x,” docs.aws.amazon.com. [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-s3.html. [Accessed: 23 July 2023].

[7] “Work with DynamoDB - AWS SDK for Java 2.x,” docs.aws.amazon.com. [Online]. Available: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-dynamodb.html. [Accessed: 23 July 2023].

[8] “Building Lambda functions with Python - AWS Lambda,” docs.aws.amazon.com. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html. [Accessed: 23 July 2023].