

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

- 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)

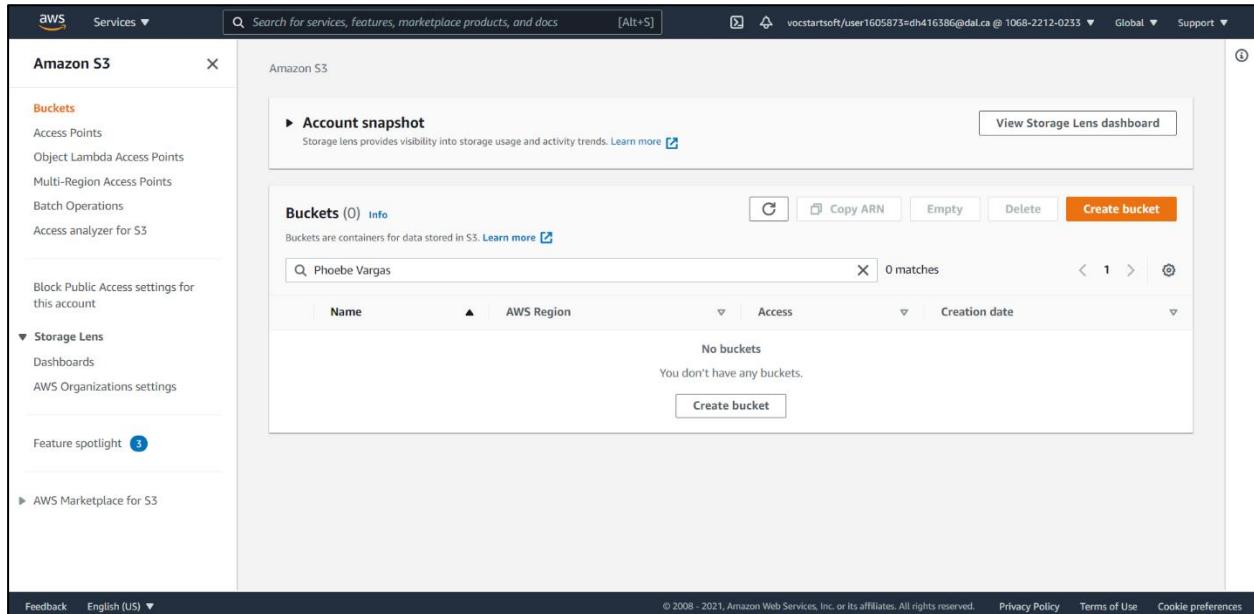


Figure 1 - Empty Amazon S3 (i.e., No buckets) [1]

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.

```

public void execute() {
    try {
        final AmazonS3 awsS3ClientBuilder = createAWS3ClientBuilder();
        if (awsS3ClientBuilder != null) {
            final Bucket firstBucket = createBucketIfNotExists(awsS3ClientBuilder, bucketName: "first-b00857606");
            final Bucket secondBucket = createBucketIfNotExists(awsS3ClientBuilder, bucketName: "second-b00857606");
        } else {
            System.err.println("Error creating AWS S3 client builder instance!");
        }
    } catch (final Exception e) {
        e.printStackTrace();
        System.err.println(e.getMessage());
    }
}

```

Run: EventDrivenServerlessApplicationTest

E:\Java\jdk-14.0.1\bin\java.exe ...
Creating bucket first-b00857606...
Bucket first-b00857606 created successfully!
Creating bucket second-b00857606...
Bucket second-b00857606 created successfully!

Process finished with exit code 0

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)

Name	AWS Region	Access	Creation date
first-b00857606	US East (N. Virginia) us-east-1	Objects can be public	October 20, 2021, 12:06:48 (UTC-03:00)
second-b00857606	US East (N. Virginia) us-east-1	Objects can be public	October 20, 2021, 12:06:52 (UTC-03:00)

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).

The screenshot shows the AWS IAM Roles page. On the left, the navigation menu is visible with 'Identity and Access Management (IAM)' selected. In the center, the 'Summary' tab for the 'AWSLambdaS3FullAccess' role is active. The role ARN is listed as 'arn:aws:iam::106822120233:role/AWSLambdaS3FullAccess'. The 'Role description' is 'Allows Lambda functions to call AWS S3 services. Full access to AWS Lambda and AWS S3.' Below this, 'Instance Profile ARNs' and 'Path' are listed as '/'. The 'Creation time' is '2021-10-21 09:38 ADT' and 'Last activity' is '2021-10-21 17:14 ADT (Yesterday)'. The 'Maximum session duration' is '1 hour'. The 'Permissions' tab is selected, showing three policies applied: 'AmazonS3FullAccess', 'AWSLambdaBasicExecutionRole', and 'AWSLambda_FullAccess', all of which are AWS managed policies. A message at the bottom indicates a permission denial for the user performing the action.

Policy name	Policy type
AmazonS3FullAccess	AWS managed policy
AWSLambdaBasicExecutionRole	AWS managed policy
AWSLambda_FullAccess	AWS managed policy

You need permissions
You do not have the permission required to perform this operation. Ask your administrator to add permissions.
User: arn:aws:sts::106822120233:assumed-role/vocstartsoft/user1605873-dh416386@dal.ca is not authorized to perform: access-analyzer>ListPolicyGenerations on resource: arn:aws:access-analyzer:us-east-1:106822120233:* with an explicit deny

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).

The screenshot shows the AWS Identity and Access Management (IAM) service interface. On the left, the navigation menu is visible with the 'Roles' option selected. The main content area is titled 'Summary' for the role 'AWSLambdaS3DynamoDBFullAccess'. Key details shown include:

- Role ARN:** arn:aws:iam::106822120233:role/AWSLambdaS3DynamoDBFullAccess
- Role description:** Allows Lambda functions to call AWS S3 services and DynamoDB services. Full access to AWS Lambda, AWS S3 and DynamoDB services.
- Instance Profile ARNs:** /
- Path:** /
- Creation time:** 2021-10-22 13:35 ADT
- Last activity:** Not accessed in the tracking period
- Maximum session duration:** 1 hour

The 'Permissions' tab is selected, showing the applied policies:

Policy name	Policy type
AmazonS3FullAccess	AWS managed policy
AmazonDynamoDBFullAccess	AWS managed policy
AWSLambdaBasicExecutionRole	AWS managed policy
AWSLambda_FullAccess	AWS managed policy

A red box highlights a message at the bottom: "You need permissions You do not have the permission required to perform this operation. Ask your administrator to add permissions. User: arn:aws:sts::106822120233:assumed-role/vocstartsoft/user1605873=dh416386@dal.ca is not authorized to perform: access-analyzer>ListPolicyGenerations on resource: arn:aws:access-analyzer:us-east-1:106822120233:* with an explicit deny".

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.

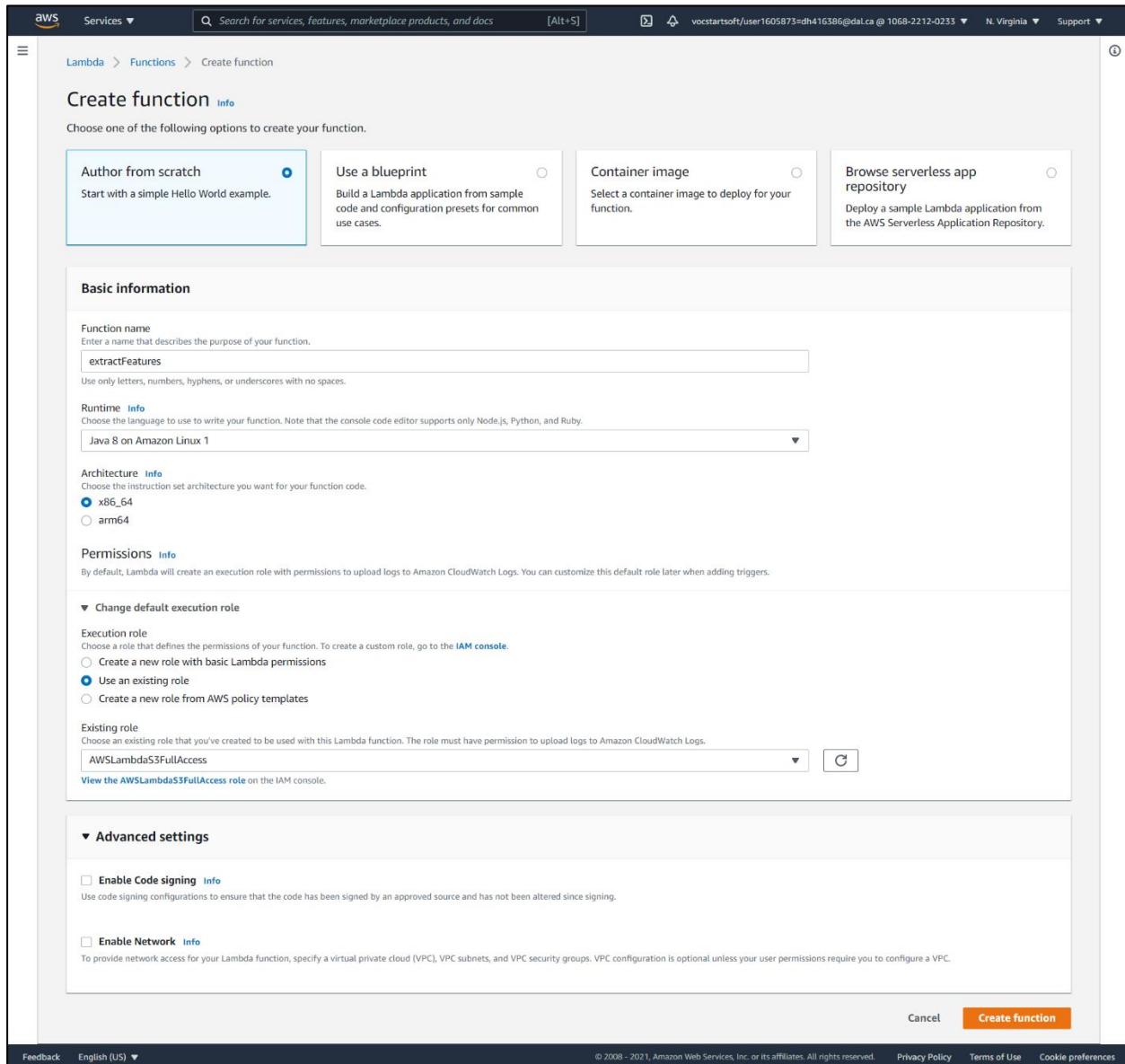


Figure 6 - Creation of "extractFeatures" lambda function [3]

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”.

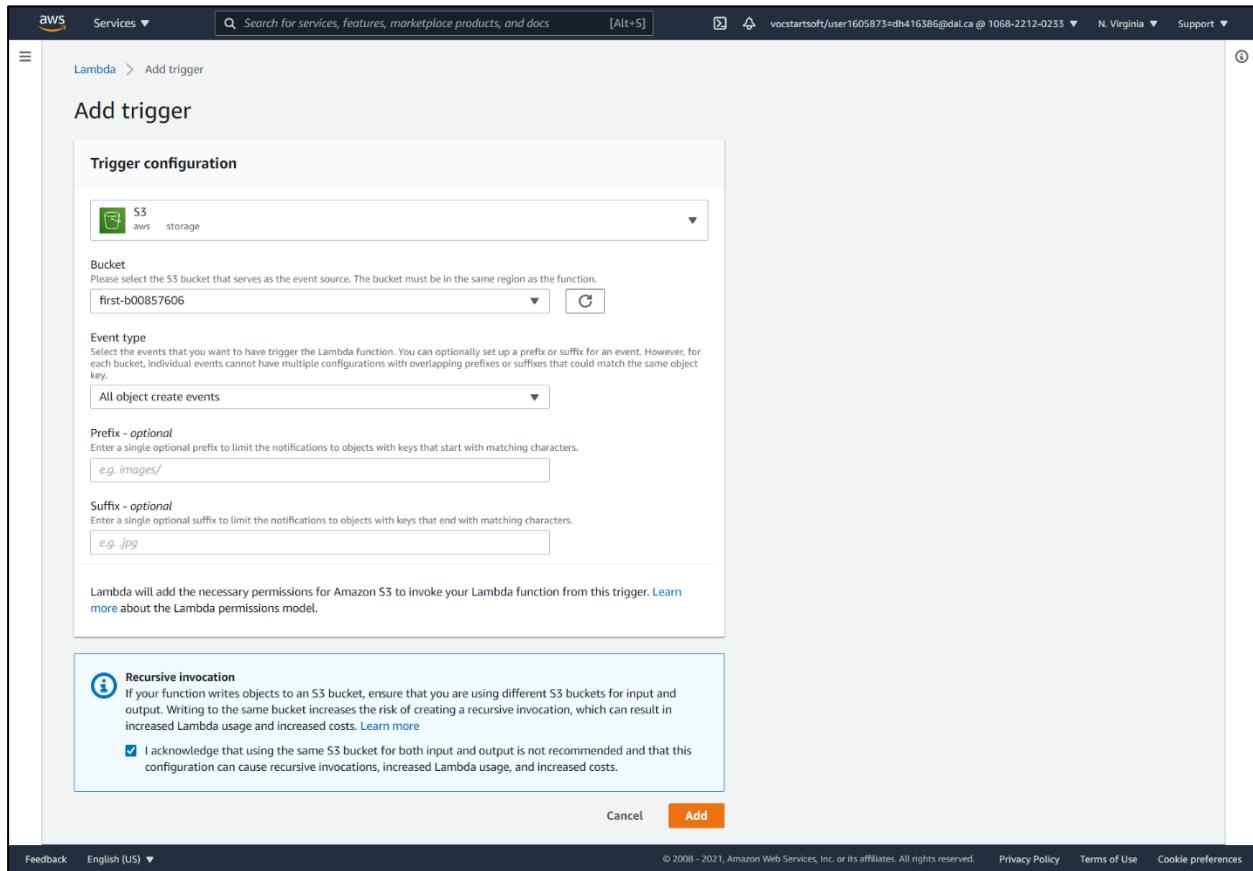


Figure 7 - Creation of trigger on lambda “extractFeatures” attached to bucket “first-b00857606” [3]

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

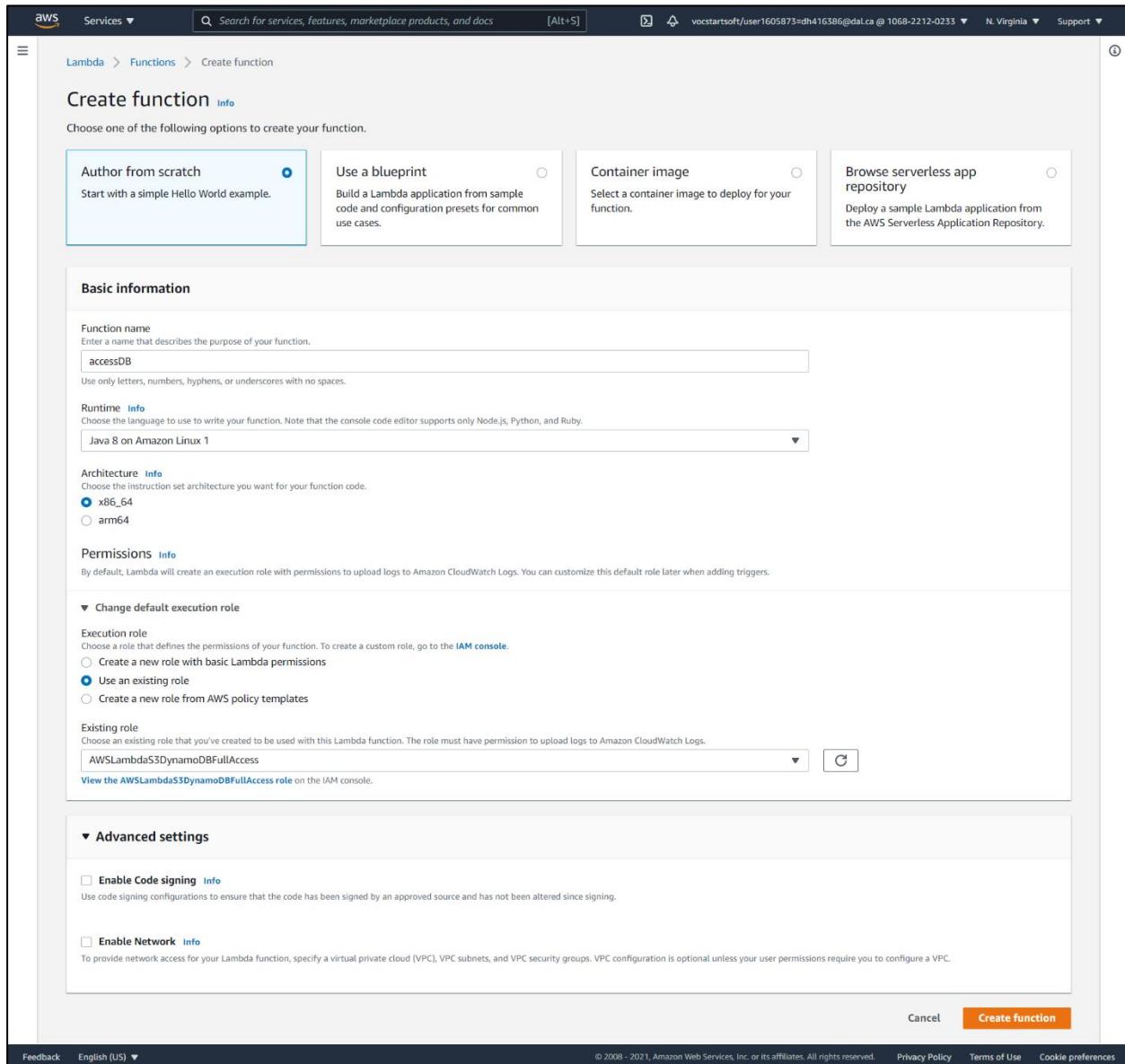


Figure 8 - Creation of "accessDB" lambda function [3]

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”.

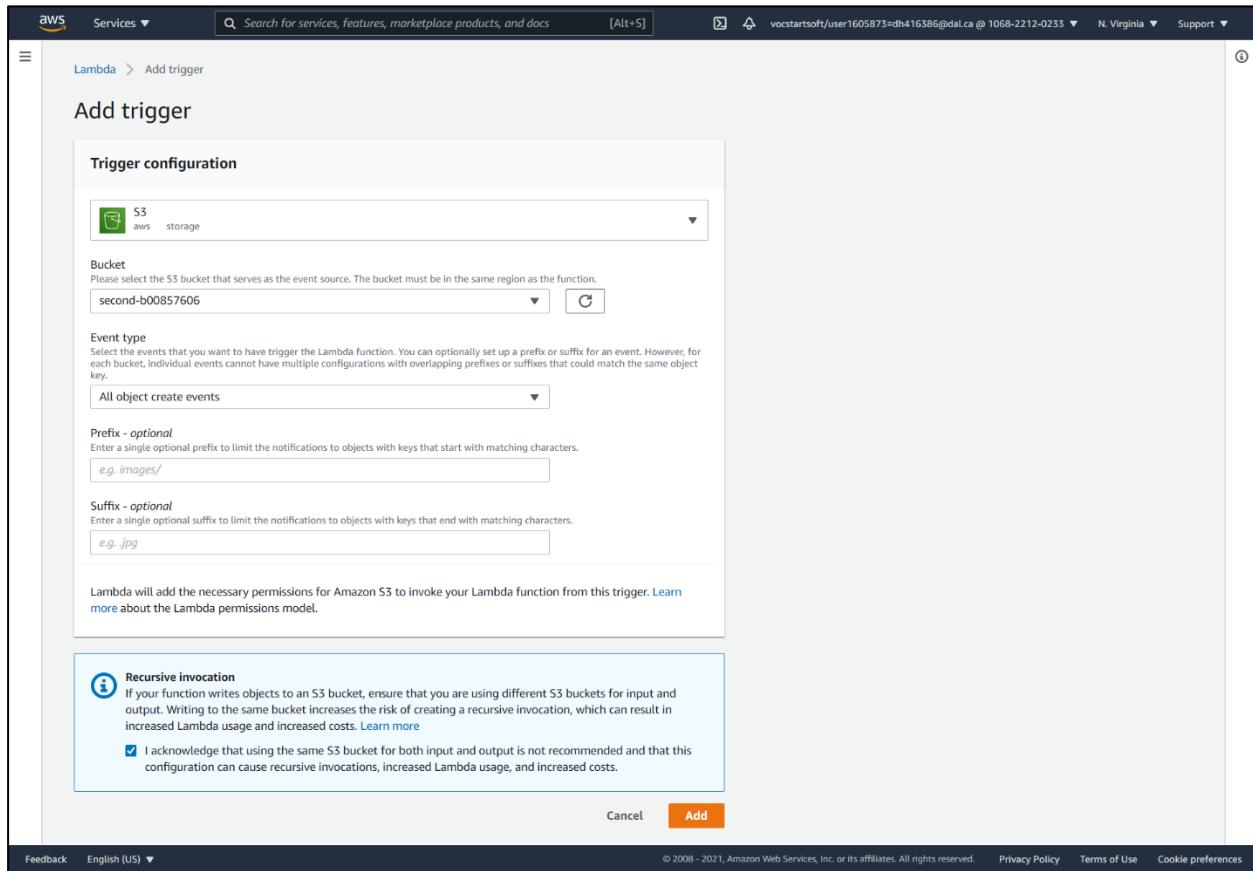


Figure 9 - Creation of trigger on lambda “accessDB” attached to bucket “second-b00857606” [3]

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.

The screenshot shows the AWS Lambda function dashboard for "extractFeatures".

- Function Overview:**
 - Name:** extractFeatures
 - Description:** -
 - Last modified:** 1 hour ago
 - Function ARN:** arn:aws:lambda:us-east-1:106822120233:function:extractFeatures
- Code Source:** The deployment package is too large to enable inline code editing.
- Runtime Settings:**
 - Runtime:** Java 8 on Amazon Linux 1
 - Handler:** assignment3.lambda.ExtractFeaturesLambda::handleRequest
 - Architecture:** x86_64
- Layers:** There is no data to display.

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.

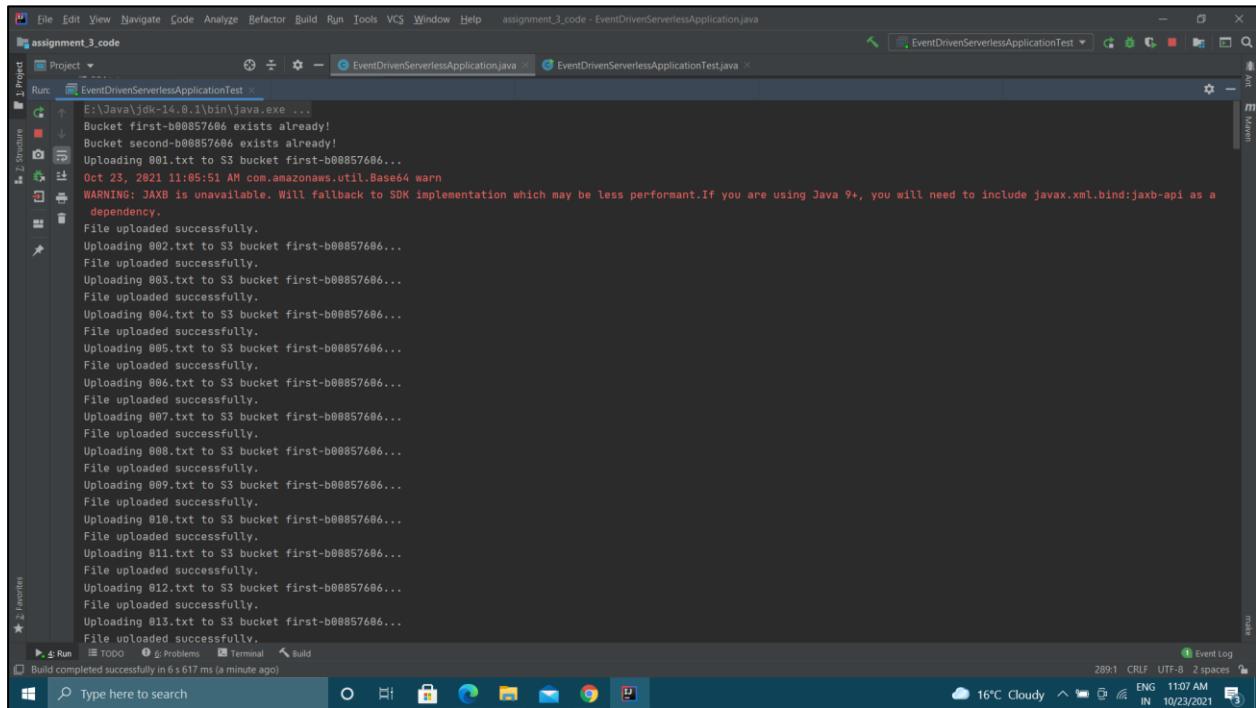
The screenshot shows the AWS Lambda function dashboard for the 'accessDB' function. The top navigation bar includes 'Services', a search bar, and account information ('vocstartsoft/user1605873=dh41G386@dal.ca @ 1068-2212-0233'). The main content area is titled 'accessDB' and contains the following sections:

- Function overview**: Shows the function name 'accessDB', a status icon, and a 'Layers' section indicating '(0)'. It also lists triggers: 'S3' and '+ Add trigger'. Buttons for 'Throttle', 'Copy ARN', and 'Actions' are at the top right.
- Code source**: A note states 'The deployment package of your Lambda function "accessDB" is too large to enable inline code editing. However, you can still invoke your function.' An 'Upload from' button is available.
- Code properties**: Displays package size (10.6 MB), SHA256 hash (I9FD3haYVZUSYCnTlbx+Yrx9jxotURbSbnlHFenChv8), and last modified date (October 23, 2021, 09:35 AM ADT).
- Runtime settings**: Shows runtime ('Java 8 on Amazon Linux 1'), handler ('assignment3.lambda.AccessDBLambda::handleRequest'), and architecture ('x86_64'). An 'Edit' button is present.
- Layers**: A table with columns 'Merge order', 'Name', 'Layer version', 'Compatible runtimes', 'Compatible architectures', and 'Version ARN'. A note says 'There is no data to display.'

At the bottom, there are links for 'Feedback', language ('English (US)'), and various AWS service links.

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).



The screenshot shows an IDE interface with the project 'assignment_3_code' open. The 'Run' tab is selected, displaying the command 'E:\Java\jdk-14.0.1\bin\java.exe ...'. The output window shows log messages from the application's execution:

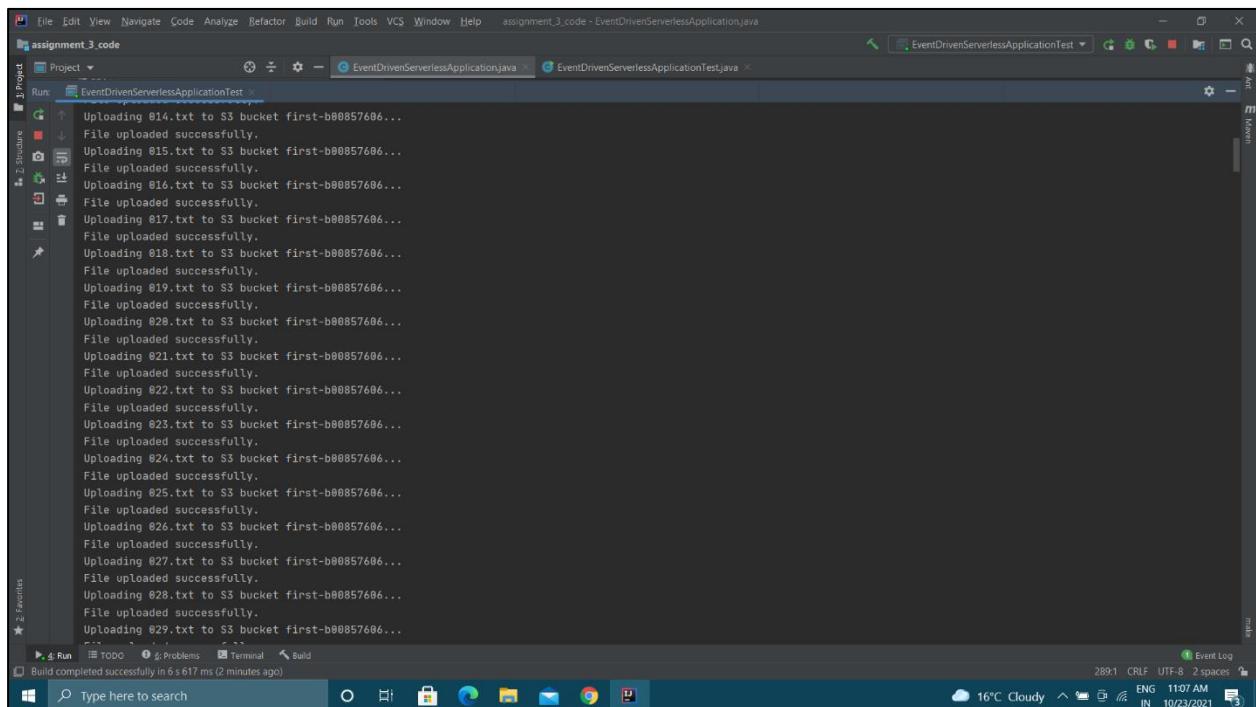
```

Bucket first-b00857606 exists already!
Bucket second-b00857606 exists already!
Uploading #01.txt to S3 bucket first-b00857606...
Oct 23, 2021 11:05:51 AM com.amazonaws.util.Base64 warn
WARNING: JAXB is unavailable. Will fallback to SDK implementation which may be less performant.If you are using Java 9+, you will need to include javax.xml.bind:jaxb-api as a dependency.
File uploaded successfully.
Uploading #02.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #03.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #04.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #05.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #06.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #07.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #08.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #09.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #10.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #11.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #12.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #13.txt to S3 bucket first-b00857606...
File uploaded successfully.

```

The status bar at the bottom indicates 'Build completed successfully in 6 s 617 ms (a minute ago)'. The system tray shows the date and time as 10/23/2021.

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



This screenshot shows the same IDE interface as Figure 12, with the project 'assignment_3_code' open and the 'Run' tab selected. The output window continues the log of file uploads:

```

Uploading #14.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #15.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #16.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #17.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #18.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #19.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #20.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #21.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #22.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #23.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #24.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #25.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #26.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #27.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #28.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading #29.txt to S3 bucket first-b00857606...

```

The status bar at the bottom indicates 'Build completed successfully in 6 s 617 ms (2 minutes ago)'. The system tray shows the date and time as 10/23/2021.

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

The screenshot shows an IDE interface with a terminal window displaying log messages. The messages indicate the successful upload of numerous files (388.txt through 401.txt) to an S3 bucket named 'first-b00857606'. The log output is as follows:

```

File uploaded successfully.
Uploading 388.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 389.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 390.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 391.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 392.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 393.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 394.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 395.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 396.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 397.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 398.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 399.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 400.txt to S3 bucket first-b00857606...
File uploaded successfully.
Uploading 401.txt to S3 bucket first-b00857606...
File uploaded successfully.

Process finished with exit code 0

```

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.

The screenshot shows the AWS S3 console interface. On the left, the sidebar lists 'Buckets', 'Storage Lens', and 'AWS Marketplace for S3'. The main area shows the 'first-b00857606' bucket details. Under the 'Objects' tab, there is a table listing 401 objects. The table includes columns for Name, Type, Last modified, Size, and Storage class. All objects are of type 'txt' and are stored in the 'Standard' storage class. The 'Last modified' column shows dates from October 23, 2021, at various times between 11:05:53 and 11:05:55 UTC-03:00.

Name	Type	Last modified	Size	Storage class
001.txt	txt	October 23, 2021, 11:05:53 (UTC-03:00)	3.9 KB	Standard
002.txt	txt	October 23, 2021, 11:05:53 (UTC-03:00)	2.2 KB	Standard
003.txt	txt	October 23, 2021, 11:05:53 (UTC-03:00)	1.3 KB	Standard
004.txt	txt	October 23, 2021, 11:05:54 (UTC-03:00)	2.5 KB	Standard
005.txt	txt	October 23, 2021, 11:05:54 (UTC-03:00)	4.8 KB	Standard
006.txt	txt	October 23, 2021, 11:05:55 (UTC-03:00)	3.8 KB	Standard
007.txt	txt	October 23, 2021, 11:05:55 (UTC-03:00)	1.7 KB	Standard
008.txt	txt	October 23, 2021, 11:05:55 (UTC-03:00)	1.7 KB	Standard

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

Objects (401)

Name	Type	Last modified	Size	Storage class
289.txt	txt	October 23, 2021, 11:07:57 (UTC-03:00)	2.4 KB	Standard
290.txt	txt	October 23, 2021, 11:07:58 (UTC-03:00)	1.9 KB	Standard
291.txt	txt	October 23, 2021, 11:07:58 (UTC-03:00)	4.1 KB	Standard
292.txt	txt	October 23, 2021, 11:07:59 (UTC-03:00)	2.4 KB	Standard
293.txt	txt	October 23, 2021, 11:07:59 (UTC-03:00)	4.0 KB	Standard
294.txt	txt	October 23, 2021, 11:07:59 (UTC-03:00)	4.5 KB	Standard
295.txt	txt	October 23, 2021, 11:08:00 (UTC-03:00)	3.7 KB	Standard
296.txt	txt	October 23, 2021, 11:08:00 (UTC-03:00)	3.8 KB	Standard
297.txt	txt	October 23, 2021, 11:08:01 (UTC-03:00)	2.4 KB	Standard
298.txt	txt	October 23, 2021, 11:08:01 (UTC-03:00)	2.5 KB	Standard
299.txt	txt	October 23, 2021, 11:08:01 (UTC-03:00)	4.2 KB	Standard
300.txt	txt	October 23, 2021, 11:08:02 (UTC-03:00)	4.6 KB	Standard

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

Objects (401)

Name	Type	Last modified	Size	Storage class
390.txt	txt	October 23, 2021, 11:08:44 (UTC-03:00)	2.1 KB	Standard
391.txt	txt	October 23, 2021, 11:08:44 (UTC-03:00)	4.8 KB	Standard
392.txt	txt	October 23, 2021, 11:08:45 (UTC-03:00)	2.0 KB	Standard
393.txt	txt	October 23, 2021, 11:08:45 (UTC-03:00)	2.8 KB	Standard
394.txt	txt	October 23, 2021, 11:08:45 (UTC-03:00)	4.8 KB	Standard
395.txt	txt	October 23, 2021, 11:08:46 (UTC-03:00)	4.8 KB	Standard
396.txt	txt	October 23, 2021, 11:08:47 (UTC-03:00)	5.9 KB	Standard
397.txt	txt	October 23, 2021, 11:08:47 (UTC-03:00)	2.5 KB	Standard
398.txt	txt	October 23, 2021, 11:08:47 (UTC-03:00)	2.2 KB	Standard
399.txt	txt	October 23, 2021, 11:08:48 (UTC-03:00)	6.1 KB	Standard
400.txt	txt	October 23, 2021, 11:08:49 (UTC-03:00)	2.3 KB	Standard
401.txt	txt	October 23, 2021, 11:08:49 (UTC-03:00)	15.8 KB	Standard

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.

Log stream	Last event time
2021/10/23/[\$LATEST]5433f27225cf4cf8ce5d5195e47919	2021-10-23 11:07:34 (UTC-03:00)
2021/10/23/[\$LATEST]74fa36a47fd4e83a4192837074dd6e9	2021-10-23 11:06:07 (UTC-03:00)
2021/10/23/[\$LATEST]07d6e59102046a5a6ad208c7ce39d85	2021-10-23 11:06:06 (UTC-03:00)
2021/10/23/[\$LATEST]8e31d1417e66463698368ad94d1a0e97	2021-10-23 11:06:05 (UTC-03:00)
2021/10/23/[\$LATEST]63fb1281b724cd49a002ce5afde2b13	2021-10-23 11:06:05 (UTC-03:00)
2021/10/23/[\$LATEST]92d4b5631573402b362f7b043021ae	2021-10-23 11:06:05 (UTC-03:00)
2021/10/23/[\$LATEST]033930cd26614148ad9d8ef448b5dab2	2021-10-23 11:06:05 (UTC-03:00)
2021/10/23/[\$LATEST]ba1fdf12cb7e44d6bd72bbebe6fd914c	2021-10-23 11:06:05 (UTC-03:00)
2021/10/23/[\$LATEST]5c0108b76da146fe62a2b5c9b58758	2021-10-23 11:06:04 (UTC-03:00)
2021/10/23/[\$LATEST]8828aa556a384ea7963bc9a495d27656	2021-10-23 11:06:03 (UTC-03:00)
2021/10/23/[\$LATEST]55bd066951724773b1c844e1d50b6c65	2021-10-23 11:06:03 (UTC-03:00)
2021/10/23/[\$LATEST]3434cc772315496599358e01cf4b1fc	2021-10-23 11:06:03 (UTC-03:00)
2021/10/23/[\$LATEST]6b0847a5784ec4ce5e3591c8cf6a6b00b	2021-10-23 11:06:02 (UTC-03:00)
2021/10/23/[\$LATEST]7d0a5e9119f64ccb63518df44a1b89	2021-10-23 11:06:02 (UTC-03:00)
2021/10/23/[\$LATEST]33b19e5cbc441a181a623936f4b7efd	2021-10-23 11:06:02 (UTC-03:00)
2021/10/23/[\$LATEST]e9fe6a07ff014564b02145b945a8f70e	2021-10-23 11:06:01 (UTC-03:00)
2021/10/23/[\$LATEST]094ddaf11690848b58b55ee847852dbaa	2021-10-23 11:06:01 (UTC-03:00)
2021/10/23/[\$LATEST]b19a5c4b5d27483ab0653729c8c5e980	2021-10-23 11:06:01 (UTC-03:00)
2021/10/23/[\$LATEST]0a1dfb3b554146a193798464396fb2	2021-10-23 11:06:00 (UTC-03:00)
2021/10/23/[\$LATEST]a4ffa7fd3cf447b90f675ef63c938f5	2021-10-23 11:05:59 (UTC-03:00)
2021/10/23/[\$LATEST]4x92d7e97b4f4bd04a4d1db7d96e03	2021-10-23 11:05:59 (UTC-03:00)
2021/10/23/[\$LATEST]2b2f92ae2140b5a15079ac21613be4	2021-10-23 11:05:58 (UTC-03:00)
2021/10/23/[\$LATEST]5ebc18d4082b44a4b79b879d2ebe1d1f	2021-10-23 11:05:58 (UTC-03:00)
2021/10/23/[\$LATEST]ea9fd5cc6adb41d6872346fdbf4e43f	2021-10-23 11:05:58 (UTC-03:00)
2021/10/23/[\$LATEST]d0f6fa20b4f42a78dcde551b25b117c	2021-10-23 11:05:57 (UTC-03:00)
2021/10/23/[\$LATEST]9e4d132b04b4946d8397cd548d145f	2021-10-23 11:05:56 (UTC-03:00)
2021/10/23/[\$LATEST]226f9bc256043c4b21971df3ce8444f	2021-10-23 11:05:56 (UTC-03:00)
2021/10/23/[\$LATEST]4fa0a67b7d8244689441903060a4c23a	2021-10-23 11:05:56 (UTC-03:00)
2021/10/23/[\$LATEST]e4359c84bf27466894d5678c80686656	2021-10-23 11:05:56 (UTC-03:00)
2021/10/23/[\$LATEST]61d274775ede4729886fd79df4824a63	2021-10-23 11:05:55 (UTC-03:00)
2021/10/23/[\$LATEST]3b78a625f8a04ae29761c2381944f0af	2021-10-23 11:05:55 (UTC-03:00)
2021/10/23/[\$LATEST]8fa653659f59466b8f96e82e5414203a	2021-10-23 11:05:55 (UTC-03:00)
2021/10/23/[\$LATEST]47fd70eee05c4a7e9d7932318596e07f	2021-10-23 11:05:54 (UTC-03:00)

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”)

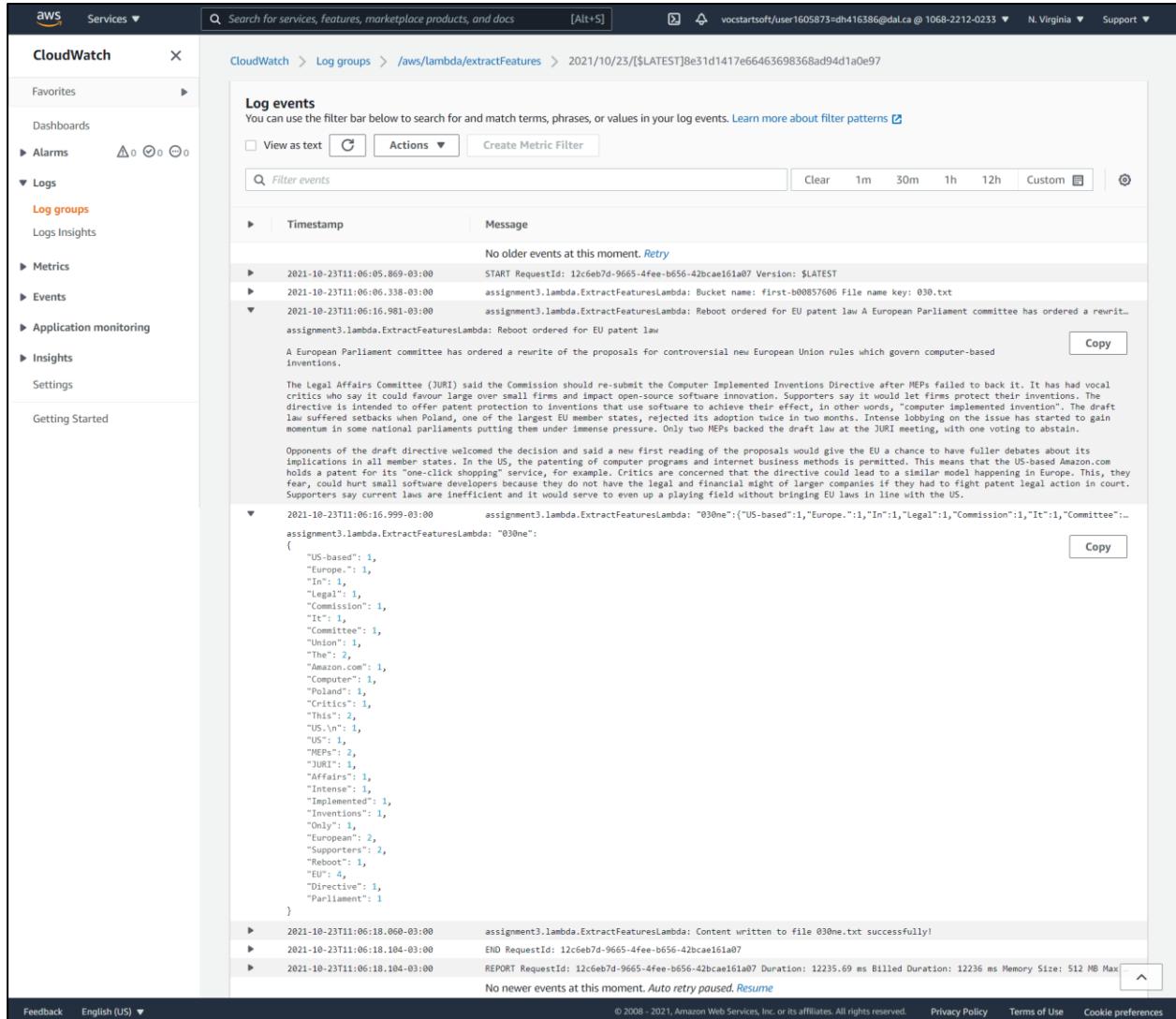
The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar is titled "CloudWatch" and includes sections for Favorites, Dashboards, Alarms, Logs (selected), Log groups, Metrics, Events, Application monitoring, Insights, Settings, and Getting Started. The main content area shows a breadcrumb navigation path: CloudWatch > Log groups > /aws/lambda/extractFeatures > 2021/10/23/[LATEST]4fa0a67b7d8244689441903060a4c23a. Below this, a section titled "Log events" contains a search bar, filter buttons (View as text, Actions, Create Metric Filter), and time range controls (Clear, 1m, 30m, 1h, 12h, Custom). The log entries table has columns for "Timestamp" and "Message". The first entry is timestamped 2021-10-23T11:05:56.399-03:00 and shows the Lambda function starting a request. Subsequent entries show the function processing the request, reaching Wi-fi farmers in Peru, and writing content to file 006ne.txt. The final entry is timestamped 2021-10-23T11:06:08.375-03:00 and shows the REPORT message indicating successful completion.

Timestamp	Message
2021-10-23T11:05:56.399-03:00	START RequestId: f79302e6-c5bd-4fdd-b1fc-c6ad826751e0 Version: \$LATEST
2021-10-23T11:05:56.852-03:00	assignment3.lambda.ExtractFeaturesLambda: Bucket name: first-b00857606 File name key: 006ne.txt
2021-10-23T11:06:07.436-03:00	assignment3.lambda.ExtractFeaturesLambda: Wi-fi web reaches farmers in Peru A network of community computer centres, linked by ...
2021-10-23T11:06:07.437-03:00	assignment3.lambda.ExtractFeaturesLambda: "006ne": {"Peru":1,"Irrigation":1,"September_The":1,"BBC":1,"Cepes":2,"Wi-fi":1,"Nr":...}
2021-10-23T11:06:08.335-03:00	assignment3.lambda.ExtractFeaturesLambda: Content written to file 006ne.txt successfully!
2021-10-23T11:06:08.375-03:00	END RequestId: f79302e6-c5bd-4fdd-b1fc-c6ad826751e0
2021-10-23T11:06:08.375-03:00	REPORT RequestId: f79302e6-c5bd-4fdd-b1fc-c6ad826751e0 Duration: 11974.98 ms Billed Duration: 11975 ms Memory Size: 512 MB Max ...

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.



The screenshot shows the AWS CloudWatch interface with the 'Logs' section selected. The log group path is /aws/lambda/extractFeatures. The log stream is 2021/10/23/[SLATEST]8e31d1417e66463698368ad94d1a0e97. The log events table has columns for 'Timestamp' and 'Message'. The first event is a START message. Subsequent events show the lambda function processing a file named '030.txt' and extracting named entities. The extracted entities are listed in a JSON object within the 'Message' field of each event. The entities include 'US-based', 'Europe', 'In', 'Legal', 'Commission', 'It', 'Committee', 'Union', 'The', 'Amazon.com', 'Poland', 'Critics', 'This', 'US', 'n', 'US', 'MEPs', 'JURI', 'Affairs', 'Intense', 'Implemented', 'Inventions', 'Only', 'European', 'Supporters', 'Reboot', 'EU', 'Directive', and 'Parliament'. The last event is a REPORT message indicating success.

```

2021-10-23T11:06:05.869-03:00 START RequestId: 12c6eb7d-9665-4fee-b656-42bcae161a07 Version: $LATEST
2021-10-23T11:06:06.338-03:00 assignment3.lambda.ExtractFeaturesLambda: Bucket name: first-b00857606 File name key: 030.txt
2021-10-23T11:06:16.981-03:00 assignment3.lambda.ExtractFeaturesLambda: Reboot ordered for EU patent law A European Parliament committee has ordered a rewrite.
assignment3.lambda.ExtractFeaturesLambda: Reboot ordered for EU patent law A European Parliament committee has ordered a rewrite.
A European Parliament committee has ordered a rewrite of the proposals for controversial new European Union rules which govern computer-based inventions.
The Legal Affairs Committee (JURI) said the Commission should re-submit the Computer Implemented Inventions Directive after MEPs failed to back it. It has had vocal critics who say it could favour large over small firms and impact open-source software innovation. Supporters say it would let firms protect their inventions. The directive is intended to offer patent protection to inventions that use software to achieve their effect, in other words, "computer implemented invention". The draft law suffered setbacks when Poland, one of the largest EU member states, rejected its adoption twice in two months. Intense lobbying on the issue has started to gain momentum in some national parliaments putting them under immense pressure. Only two MEPs backed the draft law at the JURI meeting, with one voting to abstain.
Opponents of the draft directive welcomed the decision and said a new first reading of the proposals would give the EU a chance to have fuller debates about its implications in all member states. In the US, the patenting of computer programs and internet business methods is permitted. This means that the US-based Amazon.com holds a patent for its "one-click shopping" service, for example. Critics are concerned that the directive could lead to a similar model happening in Europe. This, they fear, could hurt small software developers because they do not have the legal and financial might of larger companies if they had to fight patent legal action in court. Supporters say current laws are inefficient and it would serve to even up a playing field without bringing EU laws in line with the US.
2021-10-23T11:06:16.999-03:00 assignment3.lambda.ExtractFeaturesLambda: "030ne": {"US-based": 1, "Europe": 1, "In": 1, "Legal": 1, "Commission": 1, "It": 1, "Committee": 1, "Union": 1, "The": 2, "Amazon.com": 1, "Commission": 1, "Poland": 1, "Critics": 1, "This": 2, "US": 1, "n": 1, "US": 1, "MEPs": 2, "JURI": 1, "Affairs": 1, "Intense": 1, "Implemented": 1, "Inventions": 1, "Only": 1, "European": 2, "Supporters": 2, "Reboot": 1, "EU": 4, "Directive": 1, "Parliament": 1}
2021-10-23T11:06:18.060-03:00 assignment3.lambda.ExtractFeaturesLambda: Content written to file 030ne.txt successfully!
2021-10-23T11:06:18.104-03:00 END RequestId: 12c6eb7d-9665-4fee-b656-42bcae161a07
2021-10-23T11:06:18.104-03:00 REPORT RequestId: 12c6eb7d-9665-4fee-b656-42bcae161a07 Duration: 12235.69 ms Billed Duration: 12236 ms Memory Size: 512 MB Max:

```

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.

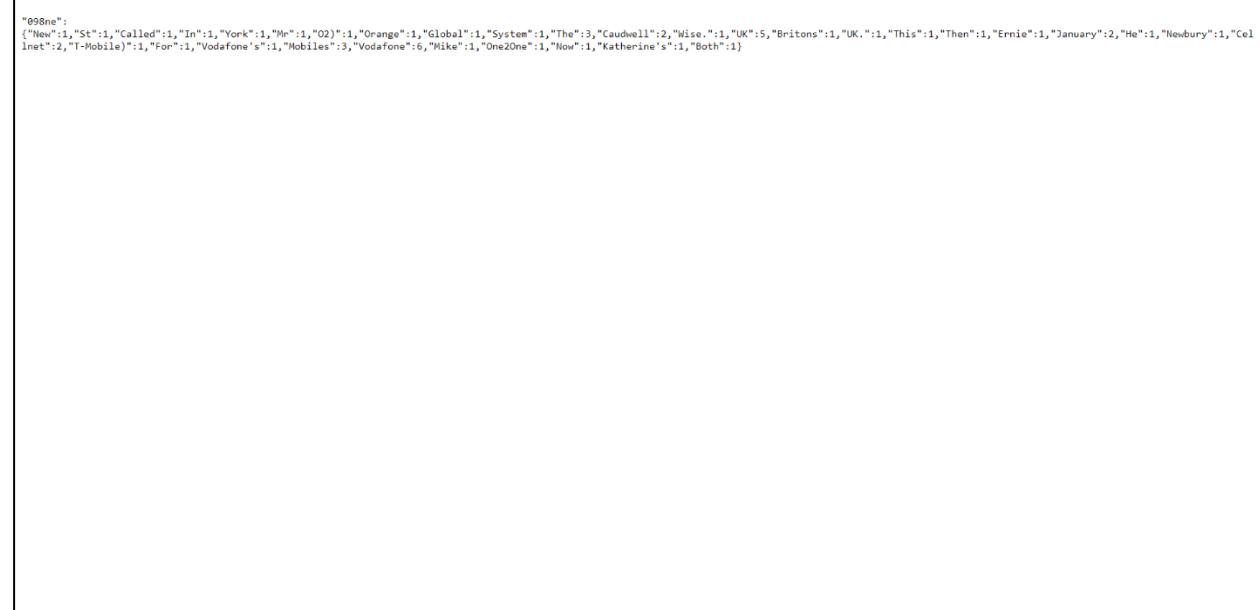


Figure 21 - JSON array content of a random file

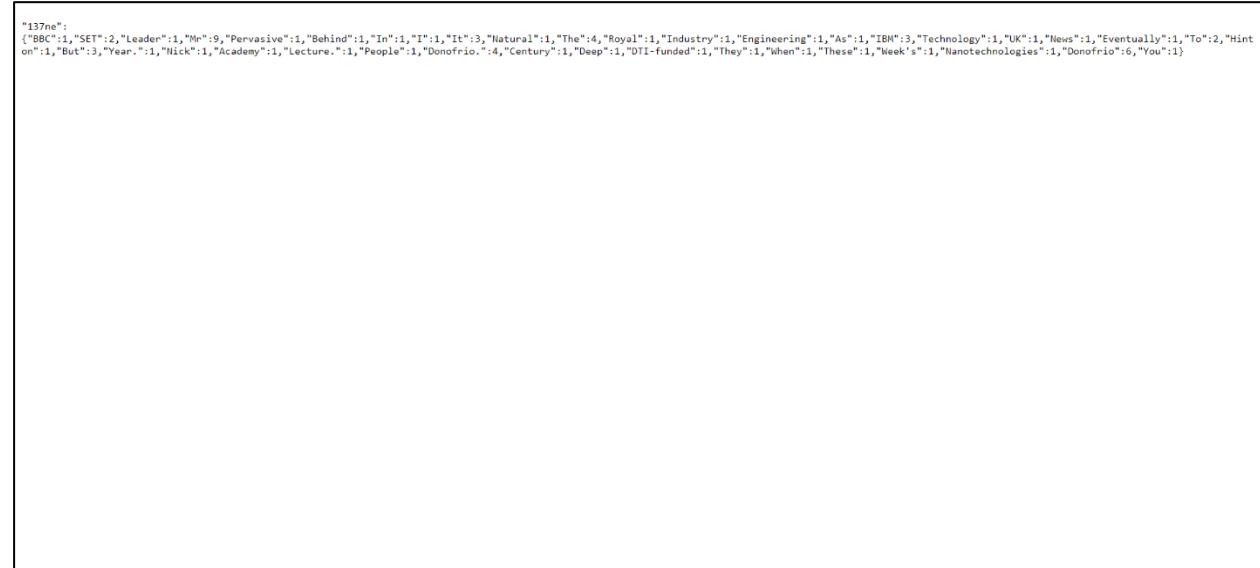


Figure 22 - JSON array content of a random file

f. This file will be saved as 001ne.txt in a new bucket - SecondB00xxxxxxxx.

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

Name	Type	Last modified	Size	Storage class
001ne.txt	txt	October 23, 2021, 11:06:05 (UTC-03:00)	553.0 B	Standard
002ne.txt	txt	October 23, 2021, 11:06:08 (UTC-03:00)	235.0 B	Standard
003ne.txt	txt	October 23, 2021, 11:06:07 (UTC-03:00)	154.0 B	Standard
005ne.txt	txt	October 23, 2021, 11:06:08 (UTC-03:00)	545.0 B	Standard
006ne.txt	txt	October 23, 2021, 11:06:08 (UTC-03:00)	504.0 B	Standard
007ne.txt	txt	October 23, 2021, 11:06:09 (UTC-03:00)	267.0 B	Standard

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

Name	Type	Last modified	Size	Storage class
291ne.txt	txt	October 23, 2021, 11:08:00 (UTC-03:00)	614.0 B	Standard
292ne.txt	txt	October 23, 2021, 11:08:00 (UTC-03:00)	402.0 B	Standard
293ne.txt	txt	October 23, 2021, 11:08:00 (UTC-03:00)	596.0 B	Standard
294ne.txt	txt	October 23, 2021, 11:08:01 (UTC-03:00)	850.0 B	Standard
295ne.txt	txt	October 23, 2021, 11:08:01 (UTC-03:00)	546.0 B	Standard
296ne.txt	txt	October 23, 2021, 11:08:02 (UTC-03:00)	585.0 B	Standard
297ne.txt	txt	October 23, 2021, 11:08:02 (UTC-03:00)	260.0 B	Standard
298ne.txt	txt	October 23, 2021, 11:08:03 (UTC-03:00)	286.0 B	Standard
299ne.txt	txt	October 23, 2021, 11:08:04 (UTC-03:00)	385.0 B	Standard
300ne.txt	txt	October 23, 2021, 11:08:04 (UTC-03:00)	474.0 B	Standard
301ne.txt	txt	October 23, 2021, 11:08:04 (UTC-03:00)	612.0 B	Standard
302ne.txt	txt	October 23, 2021, 11:08:04 (UTC-03:00)	260.0 B	Standard

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

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with options like 'Buckets', 'Storage Lens', 'Feature spotlight', and 'AWS Marketplace for S3'. The main area is titled 'Objects (397)' and contains a table of uploaded files. The table has columns for 'Name', 'Type', 'Last modified', 'Size', and 'Storage class'. All files are named '39xne.txt' (where x is a digit from 8 down to 0) and are of type 'txt', modified on October 23, 2021, at various times between 11:08:45 and 11:08:50 UTC-03:00, and are 213.0 B to 974.0 B in size, all stored in 'Standard' storage class.

	Name	Type	Last modified	Size	Storage class
	389ne.txt	txt	October 23, 2021, 11:08:45 (UTC-03:00)	469.0 B	Standard
	390ne.txt	txt	October 23, 2021, 11:08:45 (UTC-03:00)	413.0 B	Standard
	391ne.txt	txt	October 23, 2021, 11:08:45 (UTC-03:00)	706.0 B	Standard
	392ne.txt	txt	October 23, 2021, 11:08:46 (UTC-03:00)	275.0 B	Standard
	393ne.txt	txt	October 23, 2021, 11:08:47 (UTC-03:00)	252.0 B	Standard
	394ne.txt	txt	October 23, 2021, 11:08:47 (UTC-03:00)	559.0 B	Standard
	395ne.txt	txt	October 23, 2021, 11:08:48 (UTC-03:00)	691.0 B	Standard
	396ne.txt	txt	October 23, 2021, 11:08:48 (UTC-03:00)	974.0 B	Standard
	397ne.txt	txt	October 23, 2021, 11:08:49 (UTC-03:00)	267.0 B	Standard
	398ne.txt	txt	October 23, 2021, 11:08:49 (UTC-03:00)	207.0 B	Standard
	399ne.txt	txt	October 23, 2021, 11:08:50 (UTC-03:00)	881.0 B	Standard
	400ne.txt	txt	October 23, 2021, 11:08:50 (UTC-03:00)	213.0 B	Standard

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.

```

2021-10-23T11:07:12.799Z 7a2b3ef8-a751-4449-9704-e17dcd1cd9c Task timed out after 15.02 seconds
2021-10-23T11:07:13.298Z :00 START RequestId: a8326754-3547-4e42-b9ab-bd26c1ef7e4d Version: $LATEST
2021-10-23T11:07:13.714Z :00 assignment3.lambda.AccessDB: Bucket name: second-b00857606 File name key: 182ne.txt
2021-10-23T11:07:24.315Z :00 assignment3.lambda.AccessDB: "182ne": {"Rock": 1, "London-based": 1, "Ms": 1, "People": 1, "L": 1, "GSM": 1, "Nok11a": 1, "The": 2, "Henderson": 2, "Belinda": 1, "Some": 1, "UK": 1, "And": 1, "London": 1, "Tuesday": 1, "Rooster": 1}
2021-10-23T11:07:24.333Z :00 assignment3.lambda.AccessDB: "Rock":1,"London-based":1,"Ms":1,"People":1,"It":1,"GSM":1,"Mobile":1,"The":2,"Henderson":2...
2021-10-23T11:07:24.354Z :00 assignment3.lambda.AccessDB: Map Size - 16
2021-10-23T11:07:26.037Z :00 assignment3.lambda.AccessDB: Table already exists: EntityFrequencyTable (Service: AmazonDynamoDBv2; Status Code: 400; Error Type: ResourceInUseException)
2021-10-23T11:07:28.314Z :00 END RequestId: a8326754-3547-4e42-b9ab-bd26c1ef7e4d
2021-10-23T11:07:28.314Z :00 REPORT RequestId: a8326754-3547-4e42-b9ab-bd26c1ef7e4d Duration: 15015.12 ms Billed Duration: 15000 ms Memory Size: 512 MB
2021-10-23T14:07:28.313Z a8326754-3547-4e42-b9ab-bd26c1ef7e4d Task timed out after 15.02 seconds
2021-10-23T11:07:28.986Z :00 START RequestId: 23effb0-9b8f-44b6-9979-b3b4bb412340 Version: $LATEST
2021-10-23T11:07:29.416Z :00 assignment3.lambda.AccessDB: Bucket name: second-b00857606 File name key: 218ne.txt
2021-10-23T11:07:39.596Z :00 assignment3.lambda.AccessDB: "218ne": {"All":1,"Somewhat":1,"Slicing":1,"Time":3,"It":1,"Persia Still":1,"The":5,"Prince":3,...}
2021-10-23T11:07:39.633Z :00 assignment3.lambda.AccessDB: "All":1,"Somewhat":1,"Slicing":1,"Time":3,"It":1,"Persia Still":1,"The":5,"Prince":3,"As":1,"C...
2021-10-23T11:07:39.634Z :00 assignment3.lambda.AccessDB: Map Size - 27

```

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]

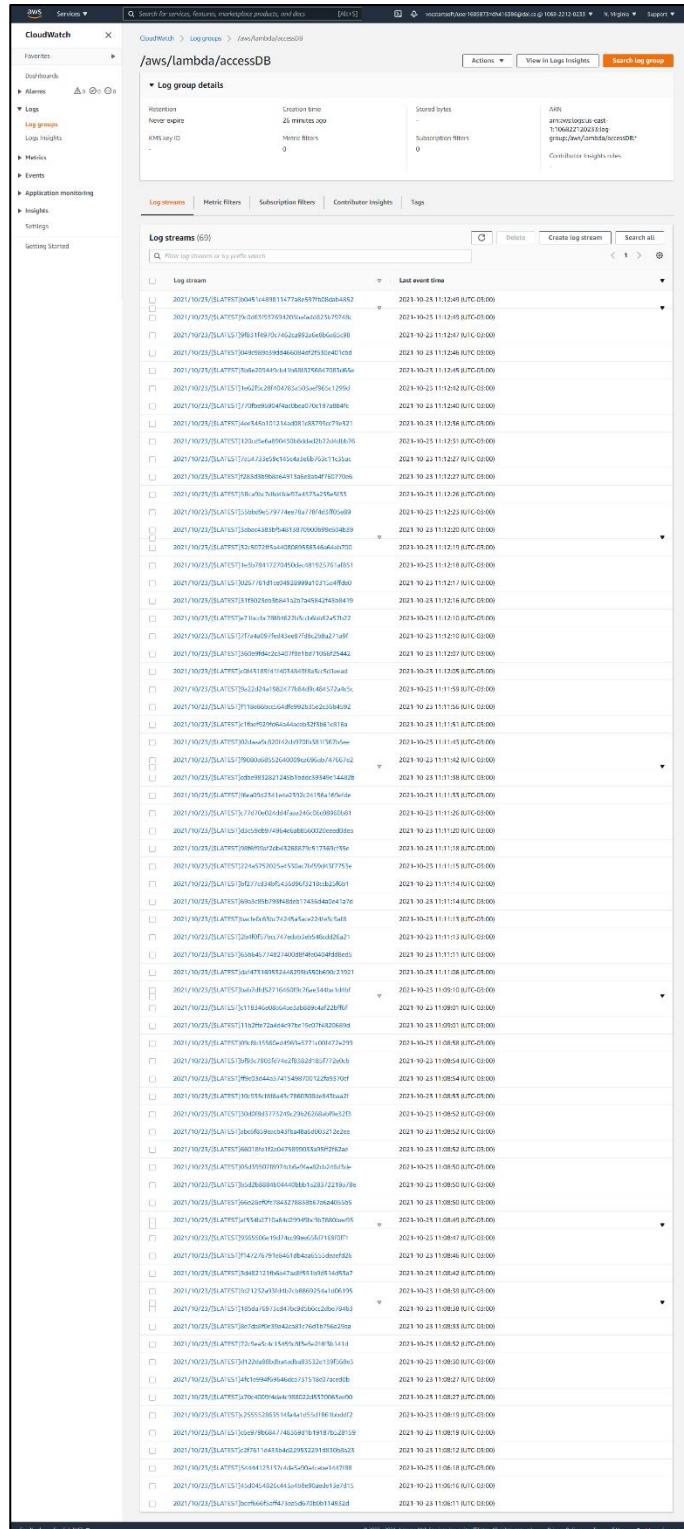
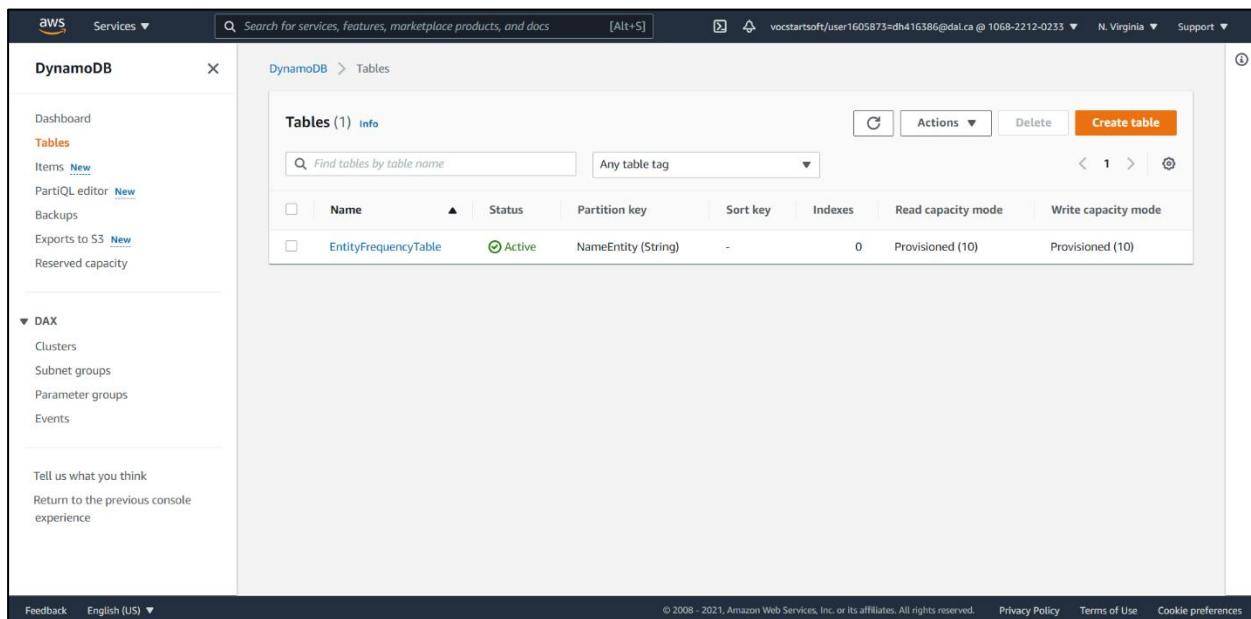


Figure 28 - Triggered “accessDB” lambda function logs when files are uploaded to bucket “second-b00857606” (contd.) [4]

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).

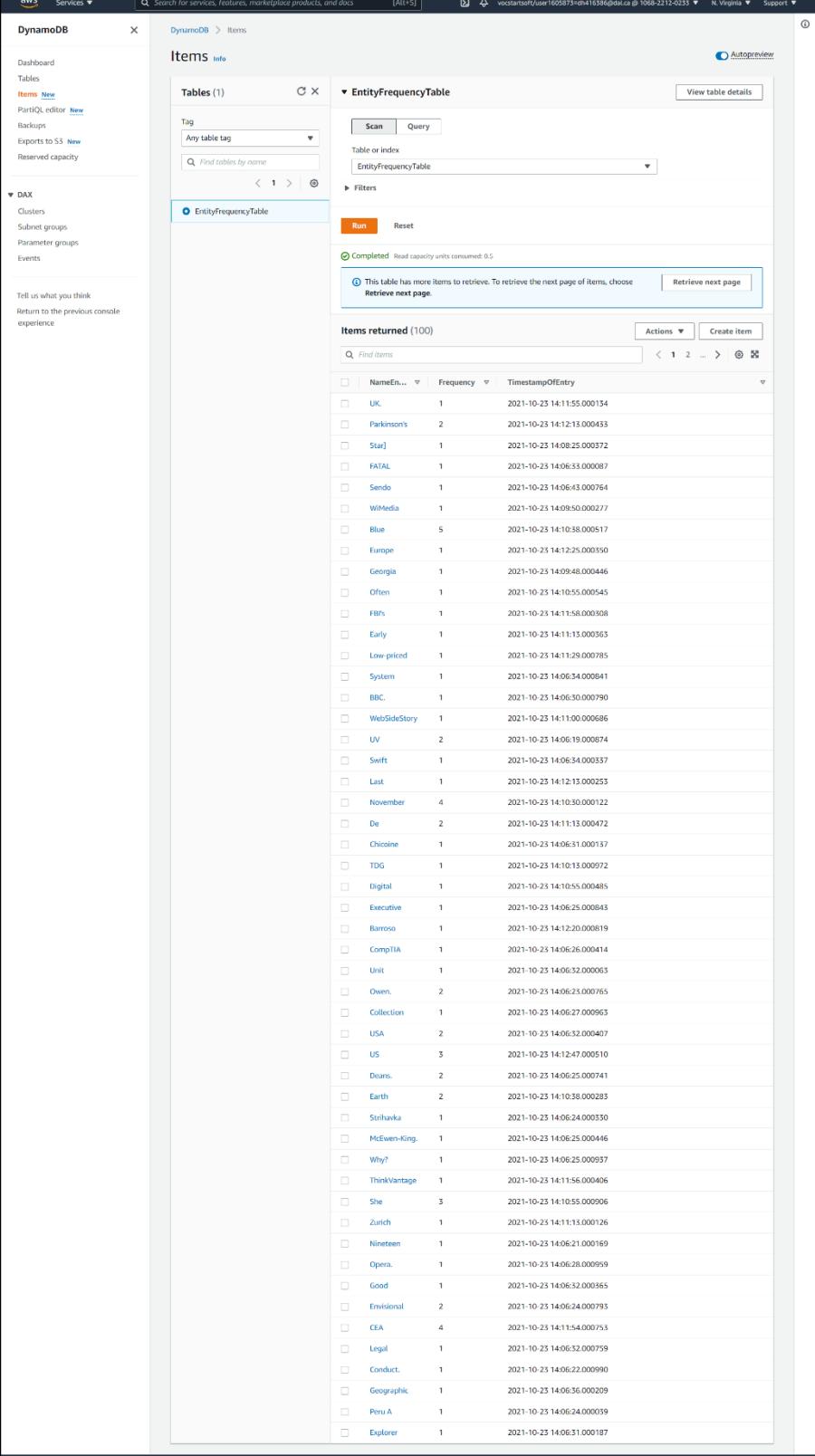
Figures 29, 27 and 28 display logs in AWS CloudWatch when files are uploaded to the “second-b00857606” bucket.



The screenshot shows the AWS DynamoDB console interface. On the left, there's a sidebar with options like Dashboard, Tables (which is selected), PartQL Editor, Backups, Exports to S3, and Reserved capacity. Below that is a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. At the bottom of the sidebar, there are links for 'Tell us what you think', 'Return to the previous console experience', 'Feedback', and language selection ('English (US)').

The main area is titled 'Tables (1) info'. It shows a table with one entry: EntityFrequencyTable. The table has columns: Name, Status, Partition key, Sort key, Indexes, Read capacity mode, and Write capacity mode. The table row shows EntityFrequencyTable, Active, NameEntity (String), -, 0, Provisioned (10), and Provisioned (10). There are buttons for Actions (with options like Delete and Create table), a search bar for 'Find tables by table name', and a dropdown for 'Any table tag'.

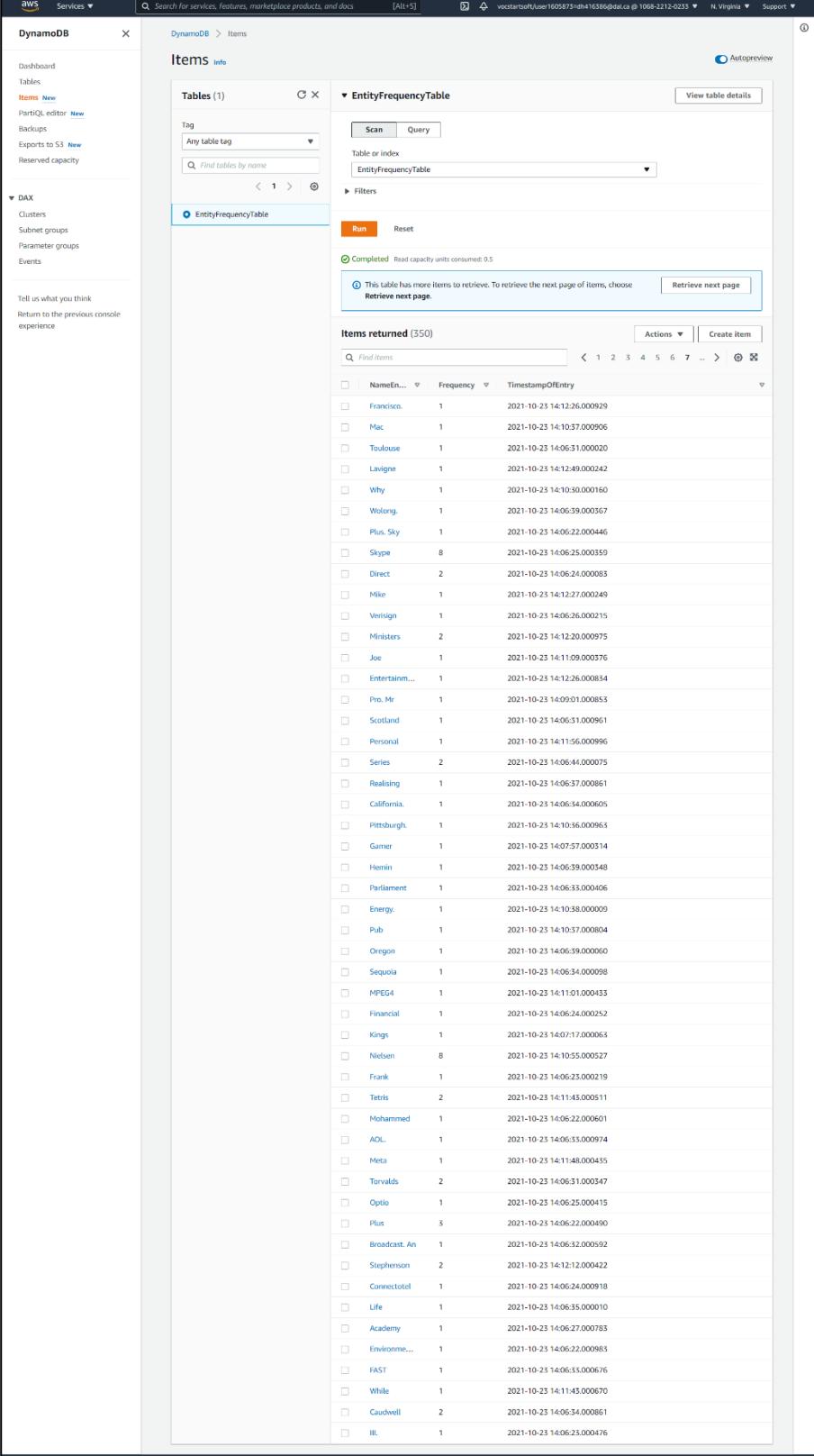
Figure 29 – Table “EntityFrequencyTable” in DynamoDB [5]



The screenshot shows the AWS DynamoDB Items page for the EntityFrequencyTable. The table has one item per row, with columns for NameEntity, Frequency, and TimestampOfEntry. The data is as follows:

NameEntity	Frequency	TimestampOfEntry
UK.	1	2021-10-23 14:11:55.000154
Parkinson's	2	2021-10-23 14:12:13.000433
Star]	1	2021-10-23 14:08:25.000372
FATAL	1	2021-10-23 14:06:33.000087
Sendo	1	2021-10-23 14:06:43.000764
WtMedia	1	2021-10-23 14:09:50.000277
Blue	5	2021-10-23 14:10:58.000517
Europe	1	2021-10-23 14:12:25.000150
Georgia	1	2021-10-23 14:09:48.000446
Often	1	2021-10-23 14:10:55.000545
FIRs	1	2021-10-23 14:11:58.000508
Early	1	2021-10-23 14:11:15.000563
Low-priced	1	2021-10-23 14:11:29.000785
System	1	2021-10-23 14:06:54.000841
BBC.	1	2021-10-23 14:06:50.000790
WebsideStory	1	2021-10-23 14:11:00.000696
UV	2	2021-10-23 14:06:19.000874
Swift	1	2021-10-23 14:06:34.000337
Last	1	2021-10-23 14:12:13.000253
November	4	2021-10-23 14:10:30.000122
De	2	2021-10-23 14:11:13.000472
Chicoine	1	2021-10-23 14:06:51.000137
TDG	1	2021-10-23 14:10:13.000972
Digital	1	2021-10-23 14:10:55.000485
Executive	1	2021-10-23 14:06:25.000843
Barroso	1	2021-10-23 14:12:20.000819
ComptIA	1	2021-10-23 14:06:26.000414
Unit	1	2021-10-23 14:06:52.000063
Owen.	2	2021-10-23 14:06:23.000765
Collection	1	2021-10-23 14:06:27.000963
USA	2	2021-10-23 14:06:52.000407
US	3	2021-10-23 14:12:47.000510
Deans.	2	2021-10-23 14:06:25.000741
Earth	2	2021-10-23 14:10:58.000283
Strifanka	1	2021-10-23 14:06:24.000330
McIver-King.	1	2021-10-23 14:06:25.000446
Why?	1	2021-10-23 14:06:25.000937
ThinkVantage	1	2021-10-23 14:11:56.000406
She	3	2021-10-23 14:10:55.000906
Zurich	1	2021-10-23 14:11:13.000126
Nineteen	1	2021-10-23 14:06:21.000169
Opera.	1	2021-10-23 14:06:28.000959
Good	1	2021-10-23 14:06:52.000365
Envisional	2	2021-10-23 14:06:24.000793
CEA	4	2021-10-23 14:11:54.000753
Legal	1	2021-10-23 14:06:52.000759
Conduct.	1	2021-10-23 14:06:22.000990
Geographic	1	2021-10-23 14:06:36.000209
Peru A	1	2021-10-23 14:06:24.000039
Explorer	1	2021-10-23 14:06:31.000187

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



The screenshot shows the AWS DynamoDB Items page for the EntityFrequencyTable. The table has three columns: NameEntity, Frequency, and TimestampOfEntry. The data is as follows:

NameEntity	Frequency	TimestampOfEntry
Francisco.	1	2021-10-23 14:12:26.000929
Mac	1	2021-10-23 14:10:57.000906
Toulouse	1	2021-10-23 14:06:51.000020
Lavigne	1	2021-10-23 14:12:49.000242
Why	1	2021-10-23 14:10:30.000160
Wulong.	1	2021-10-23 14:06:59.000367
Plus_Sky	1	2021-10-23 14:06:22.000446
Skype	8	2021-10-23 14:06:25.000159
Direct	2	2021-10-23 14:06:24.000083
Mike	1	2021-10-23 14:12:27.000249
Verisign	1	2021-10-23 14:06:26.000215
Ministers	2	2021-10-23 14:12:20.000975
Joe	1	2021-10-23 14:11:09.000376
Entertainm...	1	2021-10-23 14:12:26.000834
Pro_Mr	1	2021-10-23 14:09:01.000853
Scotland	1	2021-10-23 14:06:51.000961
Personal	1	2021-10-23 14:11:56.000996
Series	2	2021-10-23 14:06:44.000075
Realising	1	2021-10-23 14:06:37.000061
California.	1	2021-10-23 14:06:34.000605
Pittsburgh	1	2021-10-23 14:10:56.000965
Gamer	1	2021-10-23 14:07:57.000314
Hennin	1	2021-10-23 14:06:59.000348
Parliament	1	2021-10-23 14:06:33.000406
Energy.	1	2021-10-23 14:10:58.000009
Pub	1	2021-10-23 14:10:57.000804
Oregon	1	2021-10-23 14:06:39.000060
Sequoia	1	2021-10-23 14:06:34.000098
MPEG4	1	2021-10-23 14:11:01.000433
Financial	1	2021-10-23 14:06:24.000252
Kings	1	2021-10-23 14:07:17.000063
Nielsen	8	2021-10-23 14:10:55.000527
Frank	1	2021-10-23 14:06:23.000219
Tetris	2	2021-10-23 14:11:43.000511
Mohammed	1	2021-10-23 14:06:22.000601
AOI..	1	2021-10-23 14:06:53.000974
Meta	1	2021-10-23 14:11:48.000035
Torvalds	2	2021-10-23 14:06:31.000547
Optio	1	2021-10-23 14:06:25.000415
Plus	3	2021-10-23 14:06:22.000490
Broadcast_An	1	2021-10-23 14:06:32.000592
Stephenson	2	2021-10-23 14:12:12.000422
Connectotel	1	2021-10-23 14:06:24.000918
Life	1	2021-10-23 14:06:35.000010
Academy	1	2021-10-23 14:06:27.000783
Environme...	1	2021-10-23 14:06:22.000983
FAST	1	2021-10-23 14:06:33.000676
While	1	2021-10-23 14:11:43.000670
Curdwell	2	2021-10-23 14:06:34.000861
Ill.	1	2021-10-23 14:06:23.000476

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

Program/Scripts

- 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.getValue());
            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.SSSSS")  
                .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.