

PART-B

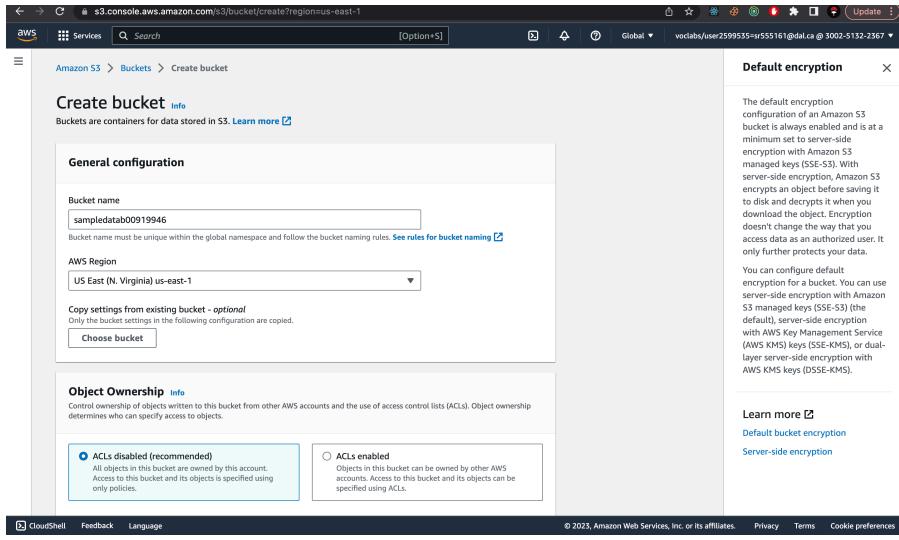


Fig. 1: Create a sampledatab00919946 bucket as mentioned in docs.

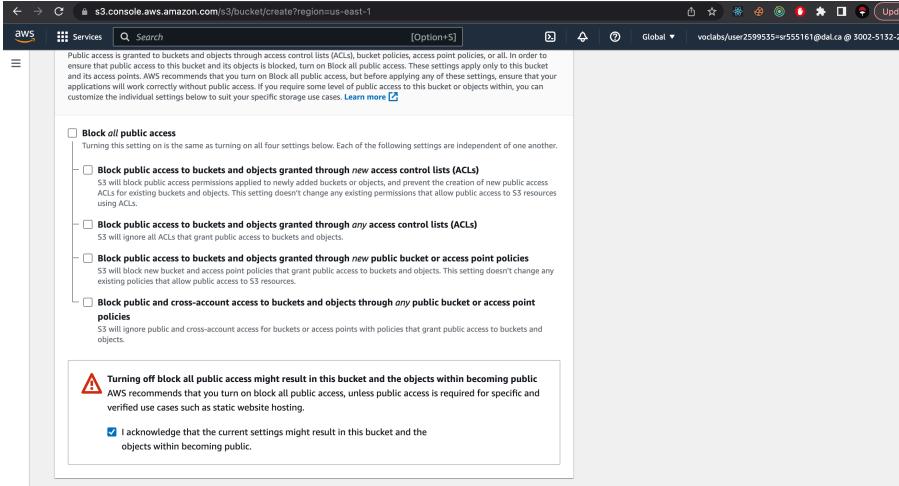


Fig. 2: Removing the public access block.

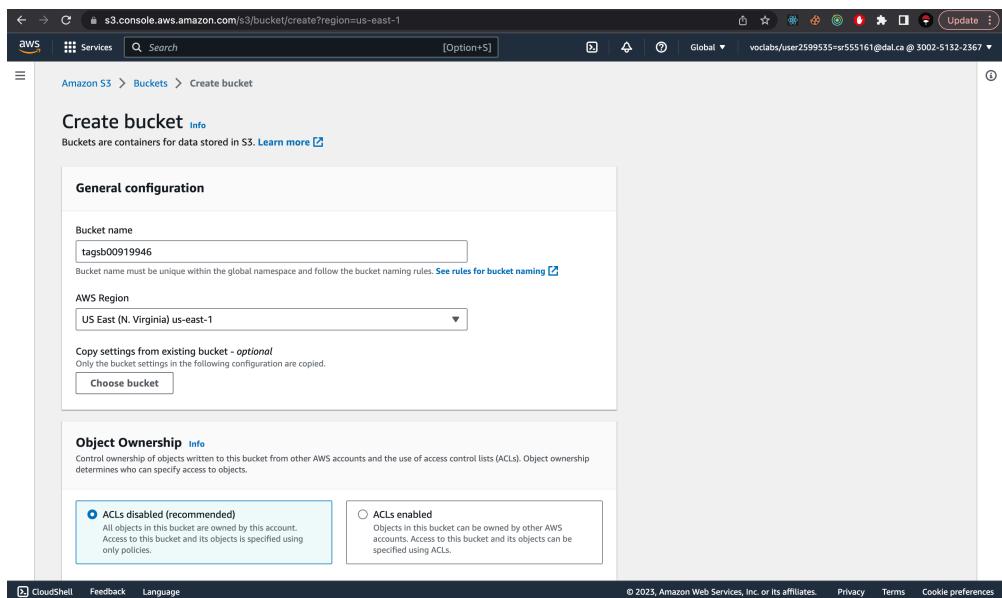


Fig. 3: Creating another bucket tagsb00919946.

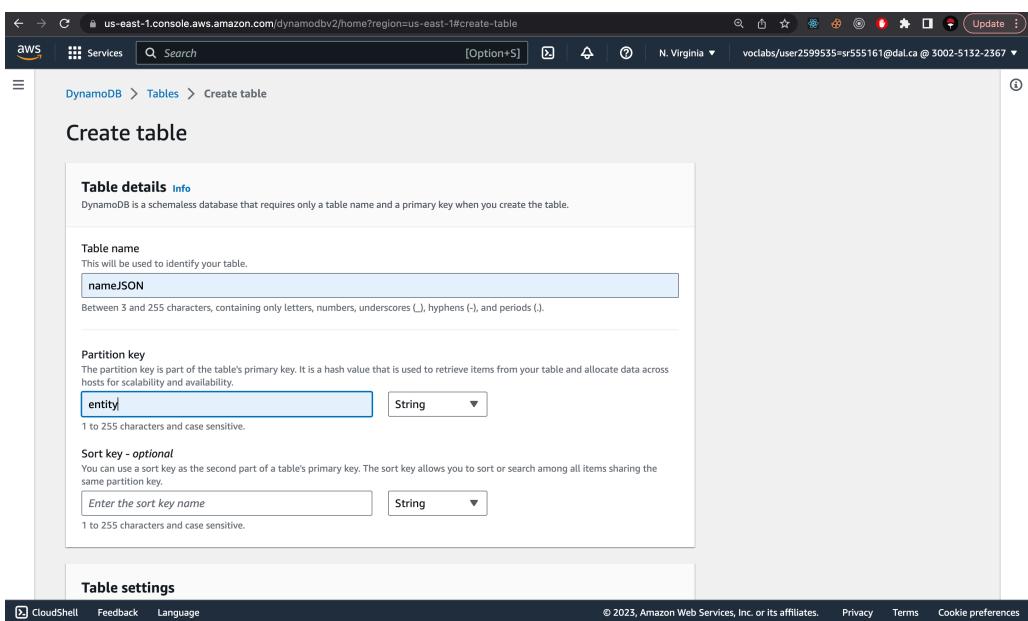


Fig. 4: Creating a DynamoDB table nameJSON to store the data from tags bucket..

The screenshot shows the AWS DynamoDB console interface. On the left, there's a sidebar with options like Dashboard, Tables, PartiQL editor, Backups, Exports to S3, Imports from S3, Reserved capacity, Settings, DAX Clusters, Subnet groups, Parameter groups, and Events. The main area is titled 'Creating the nameJSON table. It will be available for use shortly.' and shows a table list titled 'Tables (4) Info'. The table 'nameJSON' is listed with the status 'Creating', partition key 'entity (S)', sort key '-', and other details. A 'Create table' button is visible at the top right of the table list.

Fig. 5: nameJSON table created.

The screenshot shows the AWS Lambda console interface. The top navigation bar includes CloudShell, Feedback, Language, and links for 2023, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences. The main area is titled 'Create function' and shows three options: 'Author from scratch', 'Use a blueprint', and 'Container image'. The 'Author from scratch' option is selected. Below this, there are sections for 'Basic information', 'Runtime', 'Architecture', and 'Permissions'. In the 'Basic information' section, the function name is 'extractFeatures'. Under 'Runtime', 'Node.js 16.x' is selected. Under 'Architecture', 'x86_64' is selected. The 'Permissions' section notes that Lambda will create an execution role with CloudWatch Logs permissions. At the bottom, there are links for CloudShell, Feedback, Language, and the same footer links as the top bar.

Fig. 6: extractFeatures Lambda function as per the requirement.

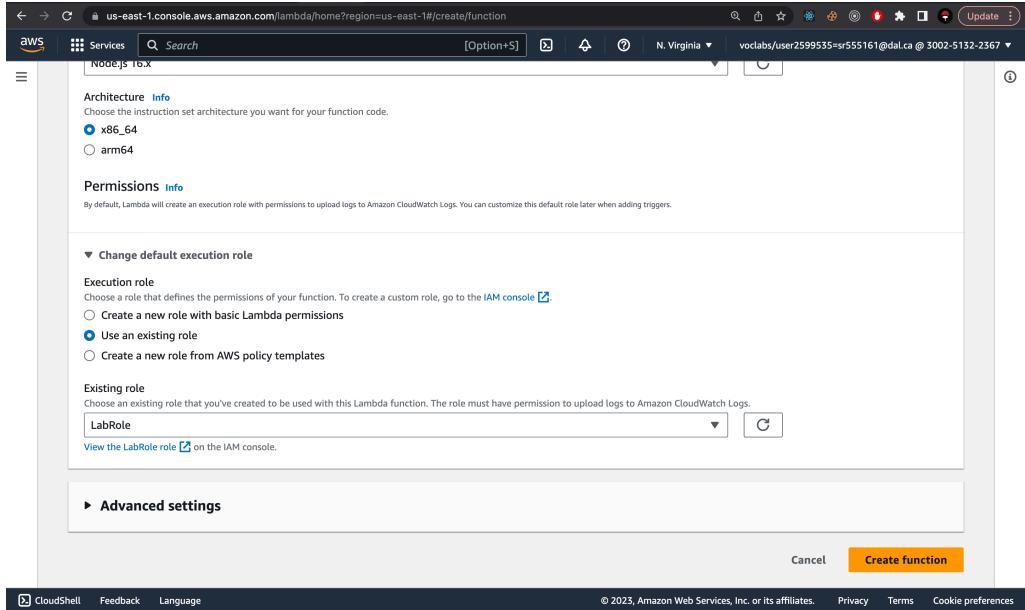


Fig. 7: selecting the IAM role as LabRole.

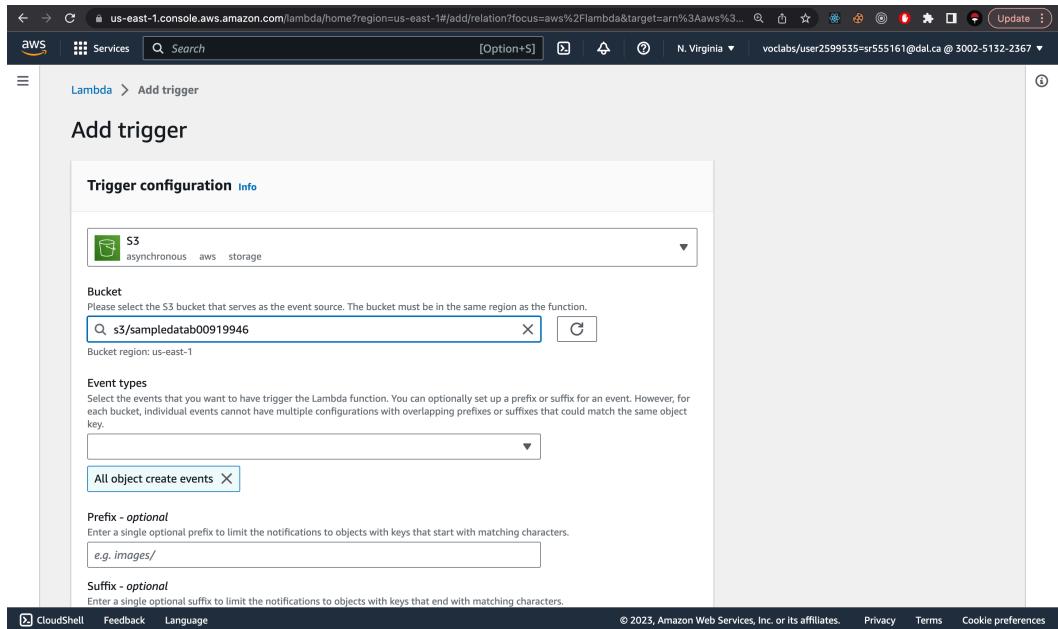


Fig. 8: Adding trigger as S3 to the extractFeatures Lambda function as per the requirement.

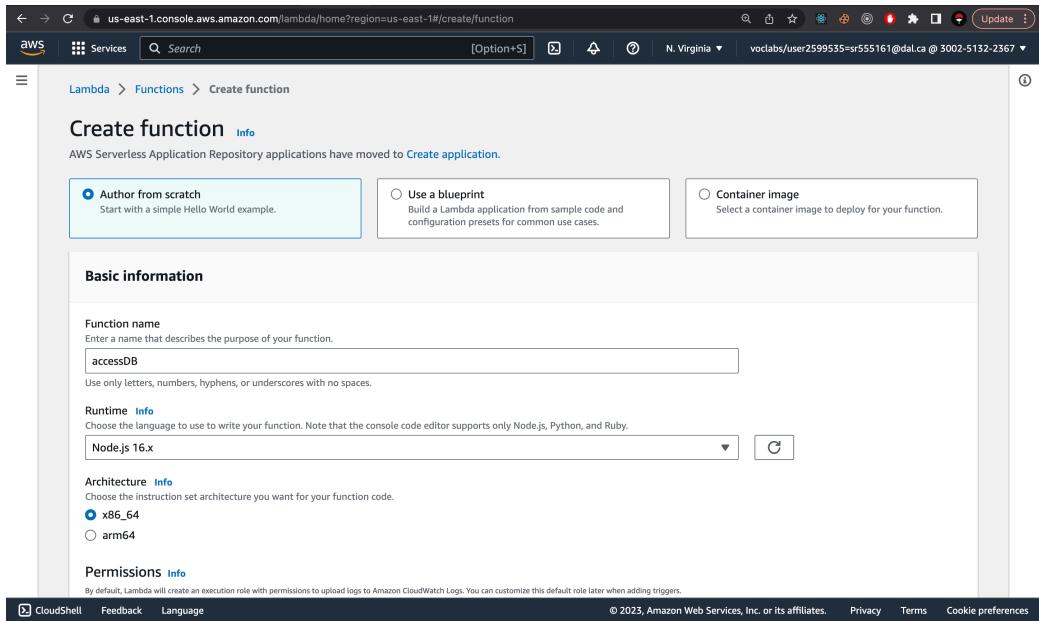


Fig. 9: accessDB Lambda function as per the requirement.

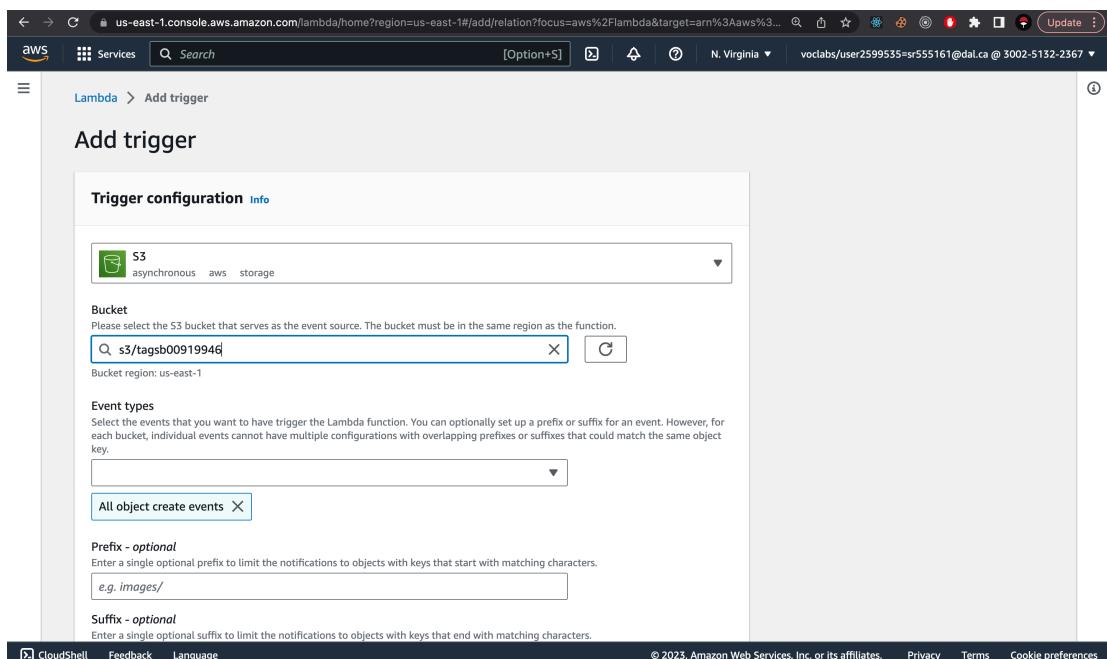


Fig. 10: Adding tags bucket trigger to the accessDB Lambda function as per the requirement.

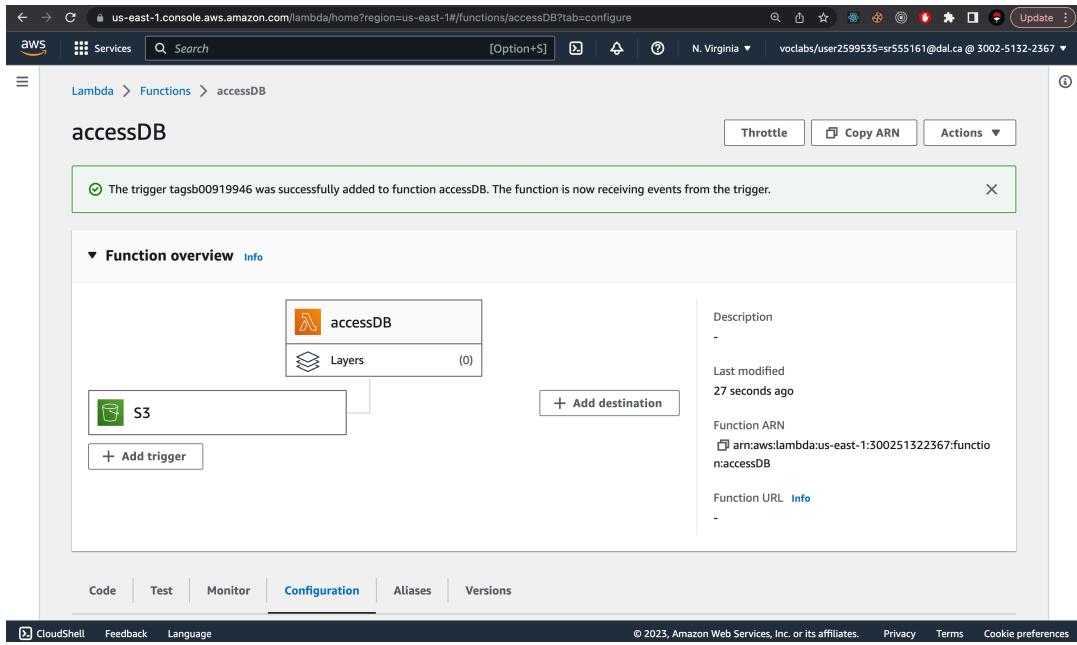


Fig. 11: accessDB S3 trigger.

```

app.js
1 const AWS = require('aws-sdk');
2 const fs = require('fs');
3
4 // AWS credentials and region
5 const AWS_ACCESS_KEY_ID = 'ASIAIUL2C7577QDR5RFB';
6 const AWS_SECRET_ACCESS_KEY = 'VQdV7BWH410YK7HMcTvbtzJMKb1a8ZrymD';
7 const AWS_REGION = 'us-east-1';
8 const AWS_SESSION_TOKEN = '';
9
10 // Bucket and folder names
11 const BUCKET_NAME = 'sampledata00919946';
12 const SOURCE_FOLDER = 'Tech';
13
14 // Configure AWS credentials and region
15 AWS.config.update({
16   accessKeyId: AWS_ACCESS_KEY_ID,
17   secretAccessKey: AWS_SECRET_ACCESS_KEY,
18   region: AWS_REGION,
19   sessionToken: AWS_SESSION_TOKEN
20 });
21
22 // Function to upload a file to S3 bucket
23 async function uploadToS3(bucket, filePath, key) {
24   const s3 = new AWS.S3();
25   const fileStream = fs.createReadStream(filePath);
26
27   const params = {
28     Bucket: bucket,
29     Key: key,
30     Body: fileStream
31   };
32
33   return new Promise((resolve, reject) => {
34     s3.upload(params, function(err, data) {
35       if (err) reject(err);
36       else resolve(data);
37     });
38   });
39 }
40
41 // Main function to upload all files in Tech folder
42 const main = async () => {
43   const techDir = './Tech';
44   const files = fs.readdirSync(techDir);
45
46   for (const file of files) {
47     const filePath = `${techDir}/${file}`;
48     const key = file;
49
50     await uploadToS3(BUCKET_NAME, filePath, key);
51   }
52 }
53
54 main().catch((err) => {
55   console.error(`An error occurred: ${err}`);
56 });

```

Fig. 12: Javascript code written to put the tech folder files in sampleDataB00919946 S3 bucket with 100 milliseconds.

The screenshot shows the AWS Lambda console interface. The URL is us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/extractFeatures?tab=code. The page title is "extractFeatures - Lambda Functions". The top navigation bar includes "Services", "Search", "[Option+S]", "N. Virginia", "voclabs/user2599535=r555161@dal.ca @ 3002-5132-2367", and "Update". Below the title, there are tabs for "Code", "Test", "Monitor", "Configuration", "Aliases", and "Versions". The "Code" tab is selected. The main area displays the "Code source" section with "Info" and "Test" tabs. The "Test" tab is selected. The code editor shows the "index.js" file with the following content:

```

1 const AWS = require('aws-sdk');
2 const s3 = new AWS.S3();
3
4 exports.handler = async (event) => {
5   try {
6     const { bucket, object } = event.Records[0].s3;
7     const fileContent = await getFileContent(bucket.name, object.key);
8     const namedEntities = performNER(fileContent);
9     const jsonResult = createJSONObject(namedEntities, object.key.split('.')[0]);
10    await saveJSONObjectToBucket('tagsb00919946', object.key + '.json', jsonResult);
11  }
12  return {
13    statusCode: 200,
14    body: 'Named entity extraction and saving to new bucket completed successfully.'
15  };
16 } catch (error) {
17   console.error(`Error:${error}`);
18   return {
19     statusCode: 500,
20     body: 'Error processing the file and saving to the new bucket.'
21   };
22 }
23 };
24 };

```

The bottom of the screen includes "CloudShell", "Feedback", "Language", and copyright information: "© 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences".

Fig. 13: extractFeatures Lambda function.

The screenshot shows the AWS S3 console interface. The URL is s3.console.aws.amazon.com/s3/buckets/tagsb00919946?region=us-east-1&tab=permissions. The page title is "tagsb00919946 - Buckets". The left sidebar has sections for "Buckets", "Access Points", "Storage Lens", "AWS Organizations settings", "Feature spotlight", and "AWS Marketplace for S3". The main content area shows the "Permissions" tab selected under "tagsb00919946 - Info". The "Permissions overview" section shows "Objects can be public". The "Block public access (bucket settings)" section is expanded, showing "Block all public access" is set to "Off". The "Bucket policy" section is partially visible at the bottom.

Fig. 14: Setting permissions for the tagsB00919946 S3 Bucket.

```

1  {
2   "Id": "Policy1690504951257",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "stmt1690504949817",
7       "Action": "s3:*",
8       "Effect": "Allow",
9       "Resource": "arn:aws:s3:::tagsb00919946/*",
10      "Principal": "*"
11    }
12  ]
13 }

```

Fig. 15: Setting policy for the tagsB00919946 S3 Bucket.

```

1  {
2   "Id": "Policy169050002382",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "stmt169050001349",
7       "Action": "s3:*",
8       "Effect": "Allow",
9       "Resource": "arn:aws:s3:::sampledata00919946/*",
10      "Principal": "*"
11    }
12  ]
13 }

```

Fig. 16: Setting the policy for the sampleB00919946 S3 Bucket.

The screenshot shows a code editor interface with several tabs open. The main tab displays a file named `index.js` which contains code for uploading files to an AWS S3 bucket. The code uses the AWS SDK for Node.js and includes configuration for AWS credentials and region, as well as a function to upload files from a local path to the S3 bucket. Below the code editor, there is a terminal window showing the output of the command `node index.js`, which lists the successful upload of numerous files (e.g., `001.txt`, `002.txt`, etc.) to the `s3Bucket`.

```
const AWS = require('aws-sdk');
const fs = require('fs');

// AWS credentials and region
const AWS_ACCESS_KEY_ID = 'ASIAUL2C75TWHY7KNGC';
const AWS_SECRET_ACCESS_KEY = '0CAh-xPRcEv1s5wsvnHluzb9LzfzI41kYyK5j';
const AWS_REGION = 'us-east-1';
const AWS_SESSION_TOKEN = 'FwoGZXIyMEdBL///////////E4e02z5TMQlWJYYhrfWLAACbf23hQv/0ZLN0h0jeeA50jF0L55Af4DxPV5KF14lop#MKtP3hNugEl0cz2d+1/sperM5536fV5gINCEz05mJ0fhfC6dC';

// Bucket and folder names
const BUCKET_NAME = 'sampletab00919946';
const SOURCE_FOLDER = 'Tech';

// Configure AWS credentials and region
AWS.config.update({
    accessKeyId: AWS_ACCESS_KEY_ID,
    secretAccessKey: AWS_SECRET_ACCESS_KEY,
    region: AWS_REGION,
    sessionToken: AWS_SESSION_TOKEN
});

// Function to upload a file to S3 bucket
async function uploadToS3(bucket, filePath, key) {
    const s3 = new AWS.S3();
    const fileStream = fs.createReadStream(filePath);

    const params = {
        Bucket: bucket,
        Key: key,
        Body: fileStream
    };
}

for (let i = 1; i < 100; i++) {
    uploadToS3(BUCKET_NAME, `./${SOURCE_FOLDER}/0${i}.txt`, `0${i}.txt`);
}
```

TERMINAL

```
Uploading 001.txt to s3 bucket
Uploading 002.txt to s3 bucket
Uploading 003.txt to s3 bucket
Uploading 004.txt to s3 bucket
Uploading 005.txt to s3 bucket
Uploading 006.txt to s3 bucket
Uploading 007.txt to s3 bucket
Uploading 008.txt to s3 bucket
Uploading 009.txt to s3 bucket
Uploading 010.txt to s3 bucket
Uploading 011.txt to s3 bucket
Uploading 012.txt to s3 bucket
Uploading 013.txt to s3 bucket
Uploading 014.txt to s3 bucket
Uploading 015.txt to s3 bucket
Uploading 016.txt to s3 bucket
Uploading 017.txt to s3 bucket
Uploading 018.txt to s3 bucket
Uploading 019.txt to s3 bucket
Uploading 020.txt to s3 bucket
Uploading 021.txt to s3 bucket
Uploading 022.txt to s3 bucket
Uploading 023.txt to s3 bucket
Uploading 024.txt to s3 bucket
Uploading 025.txt to s3 bucket
Uploading 026.txt to s3 bucket
Uploading 027.txt to s3 bucket
Uploading 028.txt to s3 bucket
Uploading 029.txt to s3 bucket
Uploading 030.txt to s3 bucket
Uploading 031.txt to s3 bucket
Uploading 032.txt to s3 bucket
```

Fig. 17: Files getting stored in sampledatab00919946 S3 Bucket.

The screenshot shows the Amazon DynamoDB console interface. The left sidebar includes links for Dashboard, Tables, Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Reserved capacity, Settings, Clusters, Subnet groups, Parameter groups, and Events. The main area displays the 'MockTriviaTeams' table with the following details:

- Table name: MockTriviaTeams
- Region: N. Virginia
- Last updated: 2023-08-15 10:45:00
- ItemCount: 3,019
- Read capacity units consumed: 0.5
- Write capacity units consumed: 0.0

The table data is as follows:

entity (String)	count
Brazilians	1
Section	1
Rutkowski	1
Suppliers	1
Latest	1
Dating	1

Fig. 18: Named Entities stored in DynamoDB after the triggers of Lambda from S3 Bucket.

tagsb00919946 Info

Publicly accessible

Objects (401)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Name	Type	Last modified	Size	Storage class
001.txt.json	json	July 27, 2023, 21:53:29 (UTC-03:00)	548.0 B	Standard
002.txt.json	json	July 27, 2023, 21:53:29 (UTC-03:00)	246.0 B	Standard
003.txt.json	json	July 27, 2023, 21:53:30 (UTC-03:00)	143.0 B	Standard
004.txt.json	json	July 27, 2023, 21:53:30 (UTC-03:00)	278.0 B	Standard
005.txt.json	json	July 27, 2023, 21:53:29 (UTC-03:00)	529.0 B	Standard

Fig. 19: 401 JSON files stored in tagsb00919946 S3 Bucket.

sampledatab00919946 Info

Publicly accessible

Objects (401)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Name	Type	Last modified	Size	Storage class
001.txt	txt	July 27, 2023, 21:53:26 (UTC-03:00)	3.9 KB	Standard
002.txt	txt	July 27, 2023, 21:53:26 (UTC-03:00)	2.2 KB	Standard
003.txt	txt	July 27, 2023, 21:53:27 (UTC-03:00)	1.3 KB	Standard
004.txt	txt	July 27, 2023, 21:53:27 (UTC-03:00)	2.5 KB	Standard
005.txt	txt	July 27, 2023, 21:53:27 (UTC-03:00)	4.8 KB	Standard

Fig. 20: As it is file from tech folder stored in sampledatab00919946 S3 Bucket.

```

001.txt (1).json
001.txt (1) > No Selection
1 {"001ne": {
    "Ink": 1, "Asia": 1, "The": 15, "Kyrgyz": 4, "Republic": 2, "Soviet": 2, "This": 2, "In": 3,
    "President": 1, "Askar": 1, "Akaev": 1, "Parliamentary": 1, "Presidential": 1, "US": 1,
    "German": 1, "Embassy": 1, "Soros": 1, "Foundation": 1, "It": 1, "However": 2, "At": 1,
    "UV": 2, "If": 1, "Likewise": 1, "These": 1, "Autumn": 1, "Republies": 1, "Ukraine": 1,
    "Georgia": 1, "Widely": 1, "Local": 1, "Others": 1, "Coalition": 1, "Non": 1,
    "Organizations": 1, "Serbia": 2, "South": 1, "Africa": 1, "Indonesia": 1, "Turkey": 1,
    "Afghanistan": 1, "Christian": 1, "Islamic": 1, "Other": 1, "February": 1, "David": 1,
    "Mikosz": 1, "IFES": 1}}
2

```

Fig. 21: Random JSON file selected from tagsb00919946 S3 bucket.

Timestamp	Log Group	Log Stream	Log Message	Action
2023-07-27T22:07:53.415-03:00	2023-07-28T01:07:53.415Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO 363ne	<button>Copy</button>
2023-07-27T22:07:53.415-03:00	2023-07-28T01:07:53.415Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO 363ne	<button>Copy</button>
2023-07-27T22:07:53.415-03:00	2023-07-28T01:07:53.415Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO 363.txt.json	<button>Copy</button>
2023-07-27T22:07:53.415-03:00	2023-07-28T01:07:53.415Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO 363.txt.json	<button>Copy</button>
2023-07-27T22:07:53.416-03:00	2023-07-28T01:07:53.416Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO { key: '363.txt.json', size: 531, eTag: '50f554315604d7565d7bf8f761c221be', sequencer: '0064C3141AC6E16828' }	<button>Copy</button>
2023-07-27T22:07:53.416-03:00	2023-07-28T01:07:53.416Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO { Sony: 6, PSP: 5, ... }	<button>Copy</button>
2023-07-28T01:07:53.416Z	2023-07-28T01:07:53.416Z	65004886-fe57-4df8-ae87-b88f662d5ccd	INFO { Sony: 6, PSP: 5, ... }	<button>Copy</button>

Fig. 22: Random event cloud watch logs.

```

4 exports.handler = async (event) => {
5   try {
6     for (const record of event.Records) {
7       const { bucket, object } = record.s3;
8       const fileContent = await getFileContent(bucket.name, object.key);
9       const jsonResult = JSON.parse(fileContent);
10      console.log(jsonResult);
11      console.log(`Object key: ${object.key}`);
12      const entities = jsonResult[object.key].substring(0, 3) + "ne";
13      console.log(entities);
14      console.log(`Object key: ${object.key}`);
15      console.log(`Object key: ${object.key}`);
16      console.log(`Object key: ${object.key}`);
17      // Update DynamoDB table with the entities from the current file
18      await updateDynamoDB(entities);
19    }
20  }
21  return {
22    statusCode: 200,
23    body: 'DynamoDB table updated successfully.'
24  };
25} catch (error) {
26  console.error('Error:', error);
27  return {
28    statusCode: 500,
29    body: 'Error updating the DynamoDB table.'
30  };
31}
32
33

```

Fig. 23: The accessDB lambda function code for which cloud watch logs shown in fig. 22.

```

1- [
2-   "Records": [
3-     {
4-       "eventVersion": "2.0",
5-       "eventSource": "aws:s3",
6-       "awsRegion": "us-east-1",
7-       "eventTime": "1970-01-01T00:00:00.000Z",
8-       "eventName": "ObjectCreated:Put",
9-       "userIdentity": {
10-         "principalId": "EXAMPLE"
11-       },
12-       "requestParameters": {
13-         "sourceIPAddress": "127.0.0.1"
14-       },
15-       "responseElements": {
16-         "x-amz-request-id": "EXAMPLE123456789",
17-         "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
18-       },
19-       "s3": {
20-         "s3SchemaVersion": "1.0",
21-         "configurationId": "testConfigRule",
22-         "bucket": {
23-           "name": "example-bucket",
24-           "ownerIdentity": {
25-             "principalId": "EXAMPLE"
26-           },
27-           "arn": "arn:aws:s3:::example-bucket"
28-         }
29-       }
30-     }
31-   ]
32- }

```

Fig. 24: Test event for the accessDB lambda function.

Note

I couldn't find the way to display the lambda function events log, as they are directly triggered from S3 bucket. So I displayed the cloud watch logs to complete the requirement of functional testing.

