

## Cloud and Serverless Computing Project

Dokuparthi Nilesh  
2100031703

### **Home Assignment Evaluator based on SIMILARITY INDEX**

In this project we will be using the services like AWS Lambda, Amazon S3, AWS Textract, AWS IAM, AWS Cloud Watch.

#### **AWS Lambda:**

- It is a serverless service provided by AWS.
- There is no need to provision the resources and servers.
- It is an event-driven architecture that is triggered when the event is triggered the function gets executed.
- It uses pay-as-you-go pricing .
- It has automatic scaling.

#### **Amazon S3:**

- It is an object storage service.
- It is a storage service where the data can be stored and retrieved anytime and anywhere.
- It has got the best durability 99.999999999 .
- It has unlimited storage and where each object should comprise the size of 0 bytes to 5 Tb.

#### **AWS Textract:**

- It is a machine learning tool that can automate the printed text and handwriting.
- Extract text, forms, and tables from documents with structured data, using the Amazon Textract Document Analysis API.

#### **AWS IAM:**

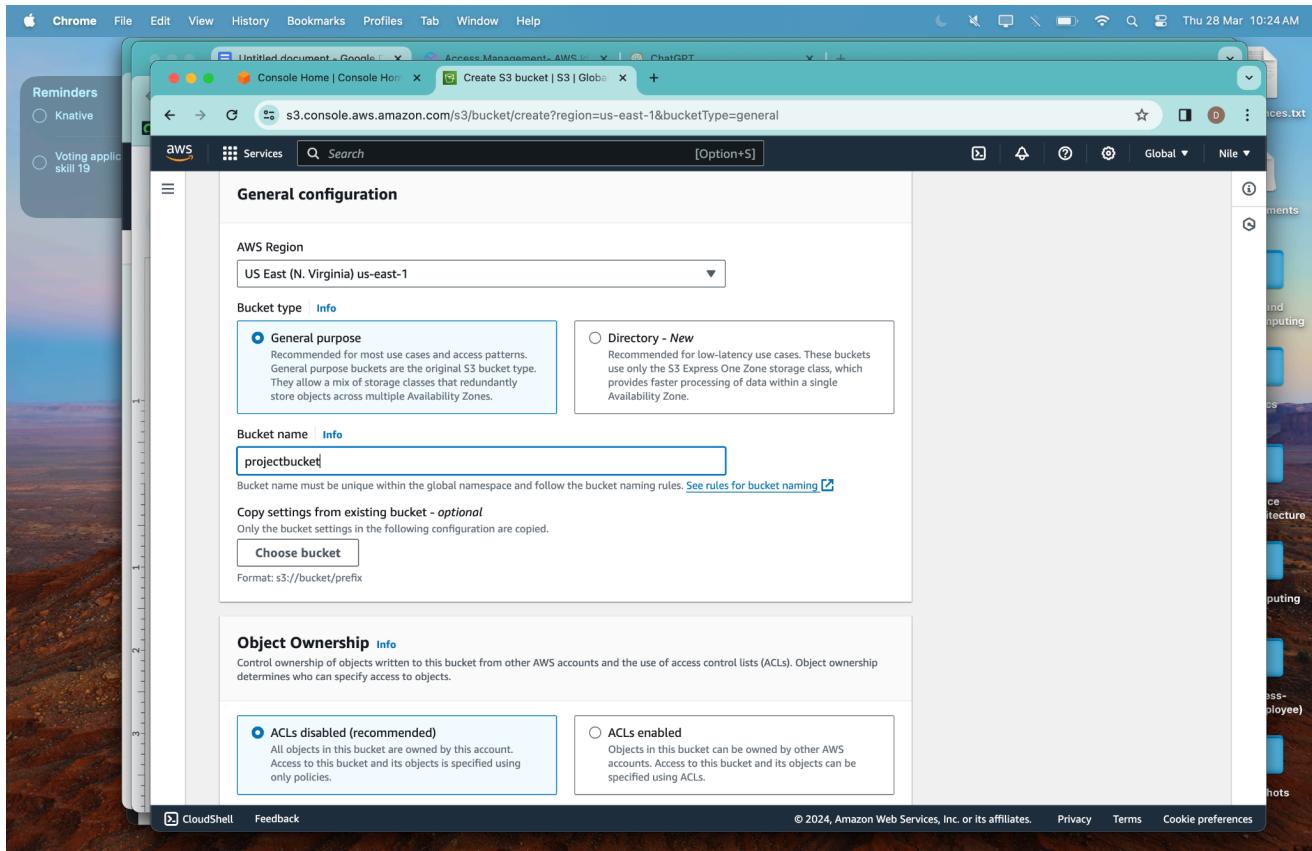
- A web service that helps you securely control access to AWS resources.
- In this project we will be using **Roles**.
- Roles:** They provide temporary permission for accessing the service in AWS.

#### **AWS CloudWatch:**

- Observe and monitor resources and applications on AWS, on premises, and on other clouds.
- It helps you to view the history of events that have been performed by the users.

## Steps of Project :

1. Create an S3 bucket.



2. Enable the ACL (Access Control Lists):

## Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

**ACLs disabled (recommended)**

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

**ACLs enabled**

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

**⚠️** We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.

### Object Ownership

**Bucket owner preferred**

If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.

**Object writer**

The object writer remains the object owner.

**ⓘ** If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. [Learn more](#) [?]

## Block Public Access settings for this bucket

### 3. Enable public access.

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#) [?]

**Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

**Block public access to buckets and objects granted through new access control lists (ACLs)**

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

**Block public access to buckets and objects granted through any access control lists (ACLs)**

S3 will ignore all ACLs that grant public access to buckets and objects.

**Block public access to buckets and objects granted through new public bucket or access point policies**

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

**Block public and cross-account access to buckets and objects through any public bucket or access point policies**

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.



**⚠️ Turning off block all public access might result in this bucket and the objects within becoming public**

AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

I acknowledge that the current settings might result in this bucket and the objects within becoming public.

### 4. Create the bucket.

## Create a Lambda Function:

1. Create a function: Provide a name for the function
2. Choose the RunTime as : Python 3.9
3. And create a role: which includes the permissions Amazon S3, AWS Textract, AWS CloudWatch, AWS Lambda and also give full access to all the services.

**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime Info**  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
 ▼

**Architecture Info**  
Choose the instruction set architecture you want for your function code.  
 x86\_64  
 arm64

**Permissions Info**  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

**▼ Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
 Create a new role with basic Lambda permissions  
 Use an existing role  
 Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
 ▼

[View the project-role role](#) ▼ on the IAM console.

**Identity and Access Management (IAM)**

Search IAM

- Dashboard
- Access management
- Groups
- Schemas
- Identity providers
- Trust settings
- SS Reports
- SS Analyzer
- External access
- Unused access
- Analyzer settings
- Entitlement report
- Initialization activity
- Access control policies (SCPs)

**Permissions policies (4) [Info](#)**

You can attach up to 10 managed policies.

Filter by Type

<input type="checkbox"/>	Policy name <a href="#">Edit</a>	Type	Attached entities
<input type="checkbox"/>	<a href="#">AmazonS3FullAccess</a>	AWS managed	4
<input type="checkbox"/>	<a href="#">AmazonTextractFullAccess</a>	AWS managed	2
<input type="checkbox"/>	<a href="#">AWSLambda_FullAccess</a>	AWS managed	7
<input type="checkbox"/>	<a href="#">CloudWatchLogsFullAccess</a>	AWS managed	1

▶ **Permissions boundary (not set)**

▼ **Generate policy based on CloudTrail events**

You can generate a new policy based on the access activity for this role, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the services and actions used and generate a policy. [Learn more](#)

[Generate policy](#)

No requests to generate a policy in the past 7 days.

And now add two files one as the Main file and the other as sub file in the S3 to find the similarity index between the files.

## newbucket1223211234312 [Info](#)

Objects Properties Permissions Metrics Management Access Points

### Objects (3) [Info](#)

[Copy S3 URI](#)[Copy URL](#)[Download](#)[Open](#)[Delete](#)[Actions ▾](#)[Create folder](#)[Upload](#)

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](#)

 Find objects by prefix Show versions< 1 >

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	responses/	Folder	-	-	-
<input type="checkbox"/>	t1.pdf	pdf	March 25, 2024, 14:48:02 (UTC+05:30)	15.0 KB	Standard
<input type="checkbox"/>	test.pdf	pdf	March 25, 2024, 14:48:04 (UTC+05:30)	2.0 KB	Standard

Test.pdf: This is the main file

**Cloud computing with AWS**

AWS is the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data centers globally. Millions of customers -- including the fastest-growing startups, largest enterprises and leading government agencies -- are using AWS to

lower cost, become more agile, and innovate faster.

T1.pdf: This is the sub file.

**Cloud computing with AWS**  
AWS is the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data centers globally. Millions of customers -- including the fastest-growing startups, largest enterprises and leading government agencies -- are using AWS to lower cost, become more agile, and innovate faster.

And now add the code in the lambda function to check the similarity index of both the files:

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The left sidebar has icons for file operations, search, and navigation. The Explorer tab is open, showing a project structure for a 'SERVERLESS-APPLICATION' with files like 'main.py', 'getEmployee.py', 'insertEmployeeData.py', and several text files. The main editor area is titled 'main.py' and contains the following Python code:

```
1 import boto3
2 from difflib import SequenceMatcher
3
4 # Set up AWS services
5 s3 = boto3.client('s3')
6 textract = boto3.client('textract')
7
8 # Set up similarity threshold
9 SIMILARITY_THRESHOLD = 0.0 # Adjust threshold as needed
10
11 def lambda_handler(event, context):
12     # Get text from main file (test.pdf)
13     main_file_bucket = 'newbucket1223211234312'
14     main_file_key = 'test.pdf'
15     main_file_text = extract_text_from_pdf(main_file_bucket, main_file_key)
16
17     # Get text from sub file (t1.pdf)
18     sub_file_bucket = 'newbucket1223211234312'
19     sub_file_key = 't1.pdf'
20     sub_file_text = extract_text_from_pdf(sub_file_bucket, sub_file_key)
21
22     # Compare similarity of the two files
23     similarity_score = calculate_similarity(main_file_text, sub_file_text)
24     print("Similarity Score:", similarity_score)
25
26     # Create response
27     response = create_response(similarity_score)
28
29     return response
30
31 def extract_text_from_pdf(bucket, key):
32     response = textract.analyze_document(Document={'S3Object': {'Bucket': bu
33
34     text = ''
35     # Extract text from Textract response
36     for block in response['Blocks']:
```

Create an Event to trigger the function:

```
1  "Records": [
2    {
3      "eventVersion": "2.1",
4      "eventSource": "aws:s3",
5      "awsRegion": "us-east-1",
6      "eventTime": "2024-03-29T00:00:00.000Z",
7      "eventName": "ObjectCreated:Put",
8      "userIdentity": {
9        "principalId": "EXAMPLE"
10      },
11      "requestParameters": {
12        "sourceIPAddress": "127.0.0.1"
13      },
14      "responseElements": {
15        "x-amz-request-id": "EXAMPLE123456789",
16        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyz"
17      },
18      "s3": {
19        "s3SchemaVersion": "1.0",
20        "configurationId": "testConfigRule",
21        "bucket": {
22          "name": "newbucket123211234312",
23          "ownerIdentity": {
24            "principalId": "EXAMPLE"
25          },
26          "arn": "arn:aws:s3:::newbucket123211234312"
27        },
28        "object": {
29          "key": "test.pdf",
30          "size": 1024,
31          "eTag": "0123456789abcdef0123456789abcdef",
32          "sequencer": "0A1B2C3D4E5F678901"
33        }
34      }
35    }
36  ]
```

In 38 Col 4 Spaces: 4 UFT-8 LF {} JavaScript

The screenshot shows the AWS Lambda Test Editor. On the left, there's a sidebar with 'Services' and 'Search' buttons, and a 'File' menu with 'Edit' and 'Find'. Below that is a search bar with 'Go to Anything...' and a dropdown for 'mainfunction'. Under 'Environment', there are two entries: 'lambda\_function' and 'lambda\_function\_code'. At the bottom of the sidebar is a 'Code properties' button. The main area has a title 'Event name' with a dropdown set to 'test' and buttons for 'C' and 'Delete'. Below it is a section titled 'Event JSON' with a 'Format JSON' button. The JSON code is as follows:

```
1 - {
2 -   "Records": [
3 -     {
4 -       "eventVersion": "2.1",
5 -       "eventSource": "aws:s3",
6 -       "awsRegion": "us-east-1",
7 -       "eventTime": "2024-03-29T00:00:00.000Z",
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": "newbucket1223211234312",
24 -           "ownerIdentity": {
25 -             "principalId": "EXAMPLE"
26 -           }
27 -         }
28 -       }
29 -     }
30 -   ]
31 - }
```

And now test the function:

The screenshot shows the AWS Lambda Test interface. At the top, there are tabs for 'lambda\_function.x' (selected), 'Environment Var', and 'Execution result'. The status bar indicates 'Status: Succeeded', 'Max memory used: 81 MB', and 'Time: 5435 ms'. The main area displays the 'Execution results' section, which includes the 'Test Event Name' (test), 'Response' (a JSON object with 'statusCode': 200 and 'body': "Similarity Score: 98.18%"), and 'Function Logs' (START, END, REPORT logs). Below the logs, the 'Request ID' is shown as d407004c-741e-4588-8d31-c0e570120771. A sidebar on the left shows the 'Environment' and the file structure: 'mainfunction - /' containing 'lambda\_function.py'.

[Code properties](#) [Info](#)

The status code is 200 and the function worked !!

The similarity index between both the files is 98.18%.

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar navigation includes 'CloudWatch' (selected), 'Favorites and recents', 'Dashboards', 'Alarms' (0 items), 'Logs' (selected), 'Log groups' (selected), 'Log Anomalies', 'Live Tail', 'Logs Insights', 'Metrics', 'X-Ray traces', 'Events', 'Application Signals', 'Network monitoring', 'Insights', and 'Settings'. The main content area displays the log group '/aws/lambda/mainfunction' for the date 2024/03/28. The log entries are as follows:

Timestamp	Message
2024-03-28T10:39:47.798+05:30	INIT_START Runtime Version: python:3.9.v47 Runtime Version ARN: arn:aws:lambda:us... RequestID: d407004c-741e-4588-8d31-c0e570120771 Version: \$LATEST
2024-03-28T10:39:48.347+05:30	START RequestID: d407004c-741e-4588-8d31-c0e570120771 Version: \$LATEST
2024-03-28T10:39:53.781+05:30	Similarity Score: 0.9818181818181818
2024-03-28T10:39:53.785+05:30	END RequestID: d407004c-741e-4588-8d31-c0e570120771
2024-03-28T10:39:53.785+05:30	REPORT RequestID: d407004c-741e-4588-8d31-c0e570120771 Duration: 5435.99 ms Billed...

Below the table, a message states: "No newer events at this moment. Auto retry paused. Resume".

And check the results in the Cloud Watch log: 98.18%.

GitHub Profile : <https://github.com/Nilesh-27/Home-Assignment-Similarity-Index-AWS>