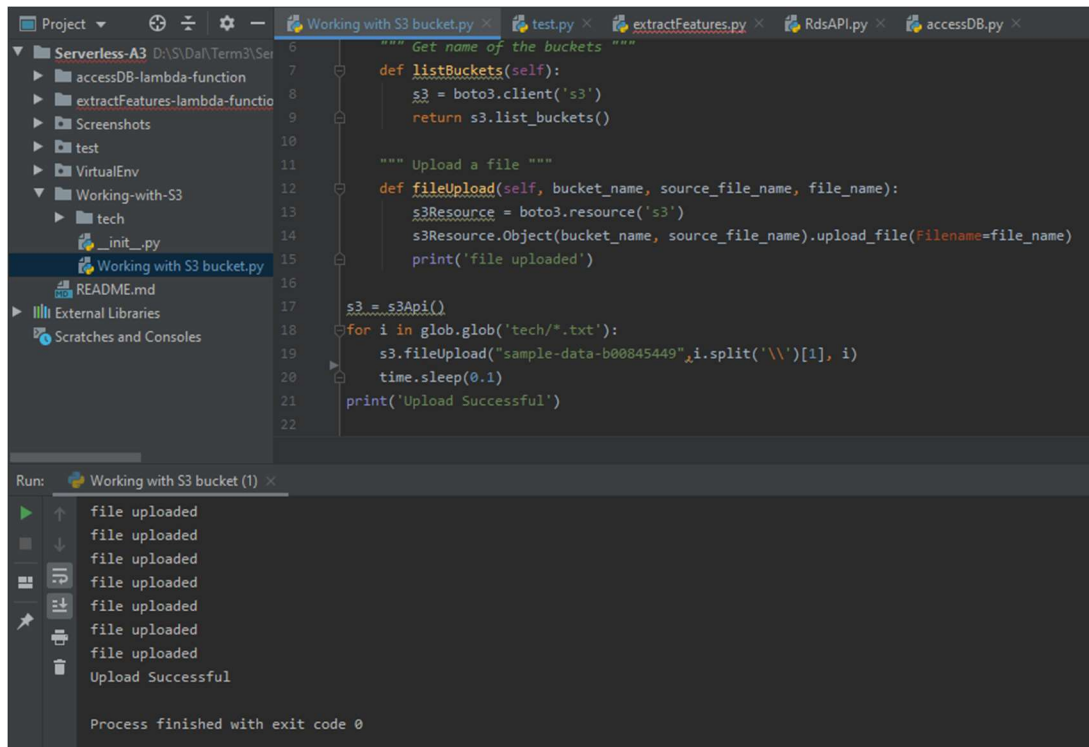


## Assignment 3 – Documentation

### Part A. Build an event-driven serverless application using AWS Lambda.

a. Create your 1st S3 bucket SampleDataB00xxxxxx and upload the files given in the Tech folder one at a time with a delay of 100 milliseconds. You need to write a script or use the SDK to upload the files one at a time to the S3 bucket.



```
6 """ Get name of the buckets """
7
8 def listBuckets(self):
9     s3 = boto3.client('s3')
10    return s3.list_buckets()
11
12 """ Upload a file """
13
14 def fileUpload(self, bucket_name, source_file_name, file_name):
15     s3Resource = boto3.resource('s3')
16     s3Resource.Object(bucket_name, source_file_name).upload_file(Filename=file_name)
17     print('file uploaded')
18
19 s3 = s3Api()
20 for i in glob.glob('tech/*.txt'):
21     s3.fileUpload("sample-data-b00845449", i.split('\\')[1], i)
22     time.sleep(0.1)
23 print('Upload Successful')
```

Run: Working with S3 bucket (1)

```
file uploaded
file uploaded
file uploaded
file uploaded
file uploaded
file uploaded
file uploaded
file uploaded
Upload Successful
Process finished with exit code 0
```

Figure a S3 bucket script

figure a shows the script created to upload all the tech folder files to the S3 bucket named 'sample-data-b00845449'. All the files are uploaded with a delay of 0.1 seconds as specified. I have used 'glob' module to get list of all files.

<div>Upload Create folder Download Actions</div>				US East (N. Virginia)	
<input type="checkbox"/>	395.txt	Jun 26, 2020 11:19:00 AM GMT-0300	4.8 KB	Standard	
<input type="checkbox"/>	396.txt	Jun 26, 2020 11:19:01 AM GMT-0300	5.9 KB	Standard	
<input type="checkbox"/>	397.txt	Jun 26, 2020 11:19:02 AM GMT-0300	2.5 KB	Standard	
<input type="checkbox"/>	398.txt	Jun 26, 2020 11:19:02 AM GMT-0300	2.2 KB	Standard	
<input type="checkbox"/>	399.txt	Jun 26, 2020 11:19:03 AM GMT-0300	6.1 KB	Standard	
<input type="checkbox"/>	400.txt	Jun 26, 2020 11:19:03 AM GMT-0300	2.3 KB	Standard	
<input type="checkbox"/>	401.txt	Jun 26, 2020 11:19:04 AM GMT-0300	15.8 KB	Standard	
<input type="checkbox"/>	402.txt	Jun 26, 2020 11:19:04 AM GMT-0300	15.9 KB	Standard	

< Viewing 301 to 402

Figure b S3 bucket files

After all the files are uploaded, the files can be seen in the bucket as shown in figure b.

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

The screenshot shows the 'Add trigger' configuration page for a Lambda function. The 'Trigger configuration' section is active. Under 'S3', the bucket 'sample-data-b00845449' is selected. The 'Event type' is set to 'Multipart upload completed'. The 'Prefix - optional' field contains 'e.g. images/'. The 'Suffix - optional' field contains '.txt'.

Figure c Add trigger

After creation of a lambda function 'ExtractFeatures', A trigger is added for event 'Multipart upload' for the files with suffix '.txt' in bucket 'sample-data-b00845449' where we added files earlier. The event 'put' can also be used.

The screenshot shows the 'Layers' section for the 'extractFeatures' function. It displays two layers: 'dependencies' (version 7) and 'nltk-packages' (version 1). The 'Add a layer' button is visible in the top right corner.

Merge order	Name	Layer version	Version ARN
1	dependencies	7	arn:aws:lambda:us-east-1:832544204593:layer:dependencies:7
2	nltk-packages	1	arn:aws:lambda:us-east-1:832544204593:layer:nltk-packages:1

Figure d Add layer

As nltk module is used in the function, two layers are added; one contains the nltk module and another layer contains the nltk packages which are required to download for the program (stopwords, punkt corpora, etc.). Hence 2 layers are added to the ExtractFeatures function.

```

10  print(f'uploading {source_file_name} to {bucket_name}')
11  def lambda_handler(event, context):
12      s3 = boto3.client('s3')
13      if(event):
14          event_object = event["Records"][0]
15          file_name = str(event_object["s3"]["object"]["key"])
16          print(file_name)
17          file_object = s3.get_object(Bucket = 'sample-data-b00845449', Key = file_name)
18          content = file_object["Body"].read().decode('utf-8')
19
20          dictionary={}
21          sentences = nltk.sent_tokenize(content)
22          for sentence in sentences:
23              words = nltk.word_tokenize(sentence)
24              tagged = nltk.pos_tag(words)
25              for i in tagged:
26                  if(i[1]=='NNP'):
27                      if(i[0] in dictionary):
28                          dictionary[i[0]] = dictionary[i[0]] + 1
29                      else:
30                          dictionary[i[0]] = 1
31
32      s3 = S3Api.s3Api()
33      s3.sendJSONObject("tagsb00845449",file_name.split('.')[0]+'ne.txt', {str(file_name.split('.')[0])+'ne':dictionary})
34
35

```

Figure e ExtractFunction Lambda handler script

Figure e shows the lambda function for ExtractFunction.

```

1  import boto3
2  import json
3  class S3Api:
4      """ Get name of the buckets """
5      def listBuckets(self):
6          s3 = boto3.client('s3')
7          return s3.list_buckets()
8
9      """ Send content to bucket object """
10     def sendJSONObject(self, bucket_name, source_file_name, content):
11         s3Resource = boto3.resource('s3')
12         s3Resource.Object(bucket_name, source_file_name).put(Body=(bytes(json.dumps(content).encode('UTF-8'))))
13         print('file uploaded')

```

Figure f S3Api function

Figure f shows the S3Api function used in ExtractFunction Lambda Handler to work with S3.

Roles > extractFeatures-role-567pr8gc

## Summary Delete role

<b>Role ARN</b>	arn:aws:iam::832544204593:role/service-role/extractFeatures-role-567pr8gc
<b>Role description</b>	<a href="#">Edit</a>
<b>Instance Profile ARNs</b>	
<b>Path</b>	/service-role/
<b>Creation time</b>	2020-06-23 03:00 ADT
<b>Last activity</b>	2020-06-23 23:19 ADT (Today)
<b>Maximum CLI/API session duration</b>	1 hour <a href="#">Edit</a>

Permissions
Trust relationships
Tags
Access Advisor
Revoke sessions

▼ Permissions policies (2 policies applied)

[Attach policies](#) [Add inline policy](#)

Policy name ▼	Policy type ▼	
▶ AWSLambdaBasicExecutionRole-fe4d61cf-744c-41cc-b5a8-6730d3a171...	Managed policy	✕
▼ AmazonS3FullAccess	AWS managed policy	✕

Figure g Policy for ExtractFeatures

Figure g shows the role for the extractFeatures lambda function. One is the default lambda policy which gives access to cloudwatch logs and another is to give full access to S3.

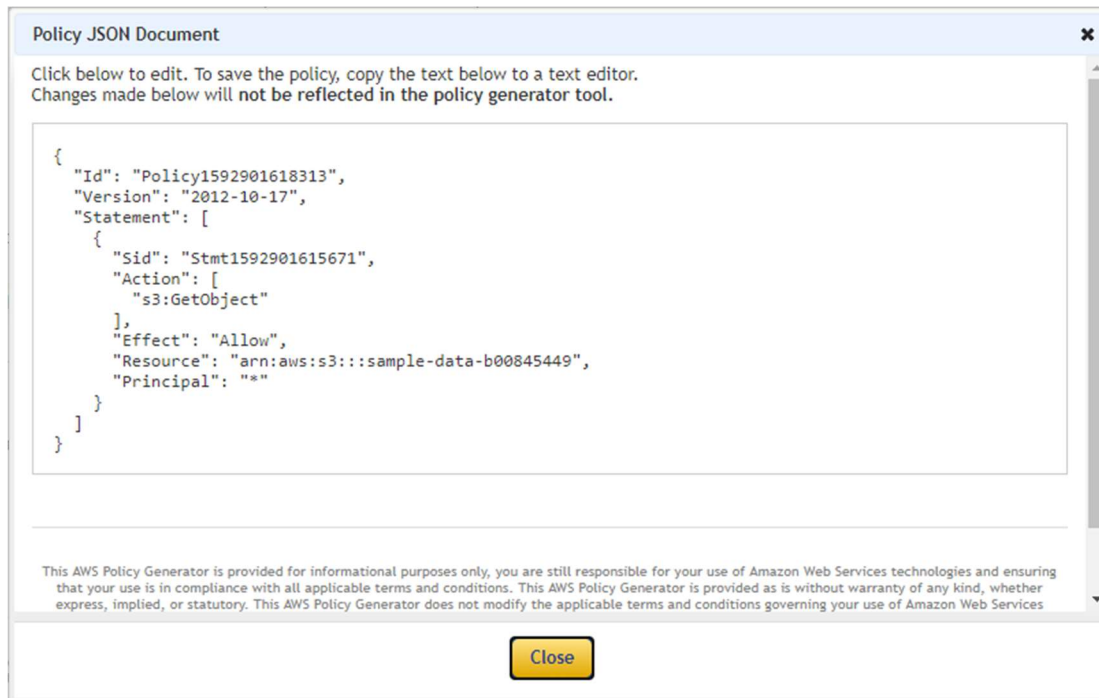


Figure h getObject policy

A policy for specifically getObject() can also be created and put here which will give specific access rights. The policy is provided in figure h.

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

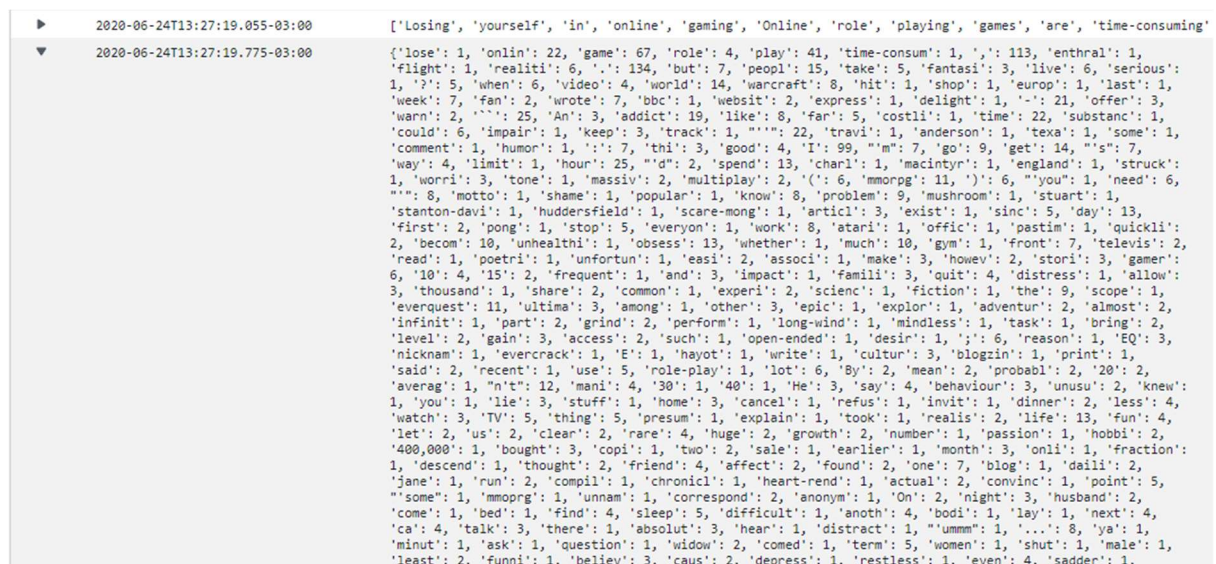


Figure i NamedEntities Extracted in Cloudwatch Logs

Cloudwatch logs shows the NamedEntities Extracted using ExtractFeatures lambda function from sample-data-b00845449 bucket keys. The ExtractFeatures lambda function will save this extracted entities in another bucket 'tagsb00845449'.

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

```
File Edit Format View Help
{"001ne": {"Ink": 1, "Asia": 1, "Kyrgyz": 4, "Republic": 2, "Soviet": 1, "President": 1, "Askar": 1, "Akaev": 1, "Parliamentary": 1, "Presidential": 1, "Embassy": 1, "Soros": 1, "Foundation": 1, "UV": 2, "Autumn": 1, "Ukraine": 1, "Georgia": 1, "Coalition": 1, "Non-governmental": 1, "Organizations": 1, "Serbia": 2, "South": 1, "Africa": 1, "Indonesia": 1, "Turkey": 1, "Afghanistan": 1, "Christian": 1, "Islamic": 1, "February": 1, "David": 1, "Mikosz": 1, "IFES": 1}}
```

Figure j 001ne.txt

Figure J shows the 001ne.txt generated in the bucket ‘tagsb00845449’. A dictionary of named entities with their frequencies in 001ne key.

e. This file will be saved as 001ne.txt in a new bucket - TagsB00xxxxxx.

tagsb00845449				
Overview Properties Permissions Management Access points				
Q Type a prefix and press Enter to search. Press ESC to clear.				
<a href="#">Upload</a> <a href="#">+ Create folder</a> <a href="#">Download</a> <a href="#">Actions</a>				US East (N. Virginia)
Viewing 1 to 300 >				
<input type="checkbox"/>	Name ▾	Last modified ▾	Size ▾	Storage class ▾
<input type="checkbox"/>	001ne.txt	Jun 26, 2020 11:30:01 AM GMT-0300	470.0 B	Standard
<input type="checkbox"/>	002ne.txt	Jun 26, 2020 11:30:01 AM GMT-0300	141.0 B	Standard
<input type="checkbox"/>	003ne.txt	Jun 26, 2020 11:30:02 AM GMT-0300	127.0 B	Standard
<input type="checkbox"/>	004ne.txt	Jun 26, 2020 11:30:02 AM GMT-0300	225.0 B	Standard
<input type="checkbox"/>	005ne.txt	Jun 26, 2020 11:30:03 AM GMT-0300	412.0 B	Standard

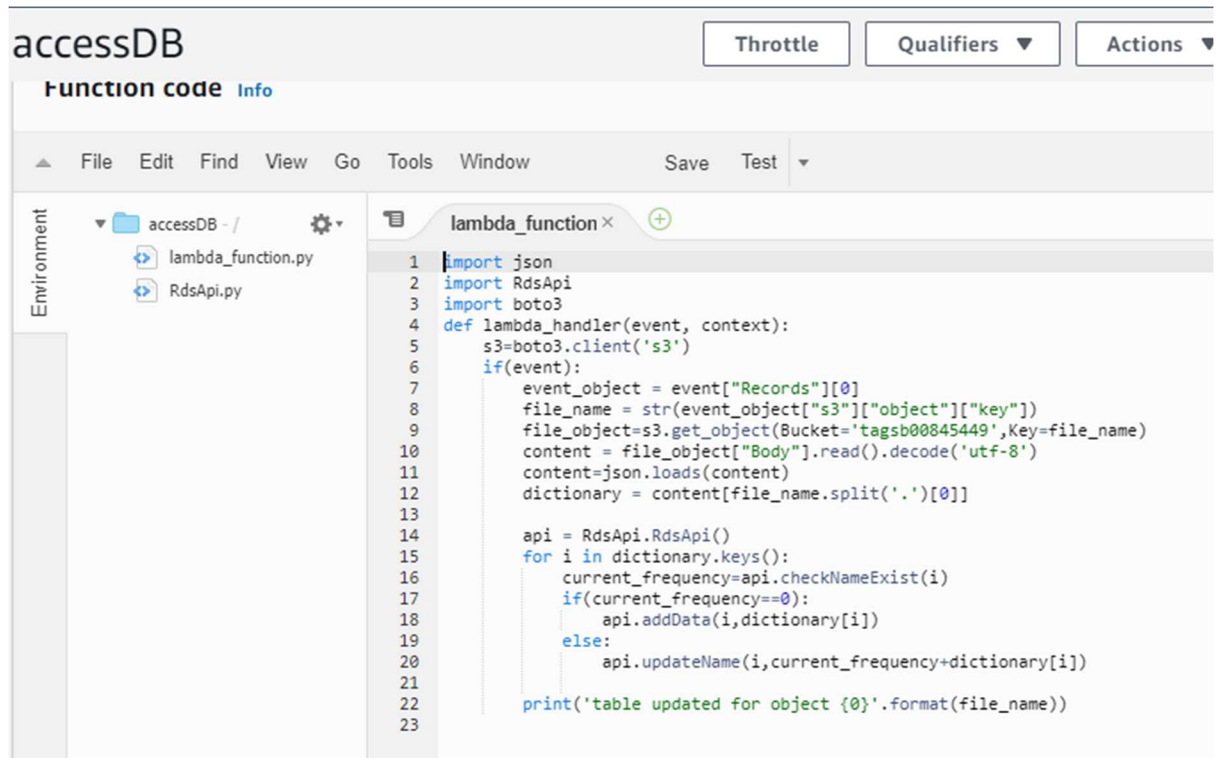
Figure k tagsb00845449 bucket files

Figure K shows all the files generated using extractFeature lambda handler where each file containing the named entity of each key in bucket sample-data-b00845449.

Note: all python programs are developed and tested for version 3.7. All the screenshots are provided separately in ‘screenshots’ folder in root directory. Python source code is provided in the root directory.



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



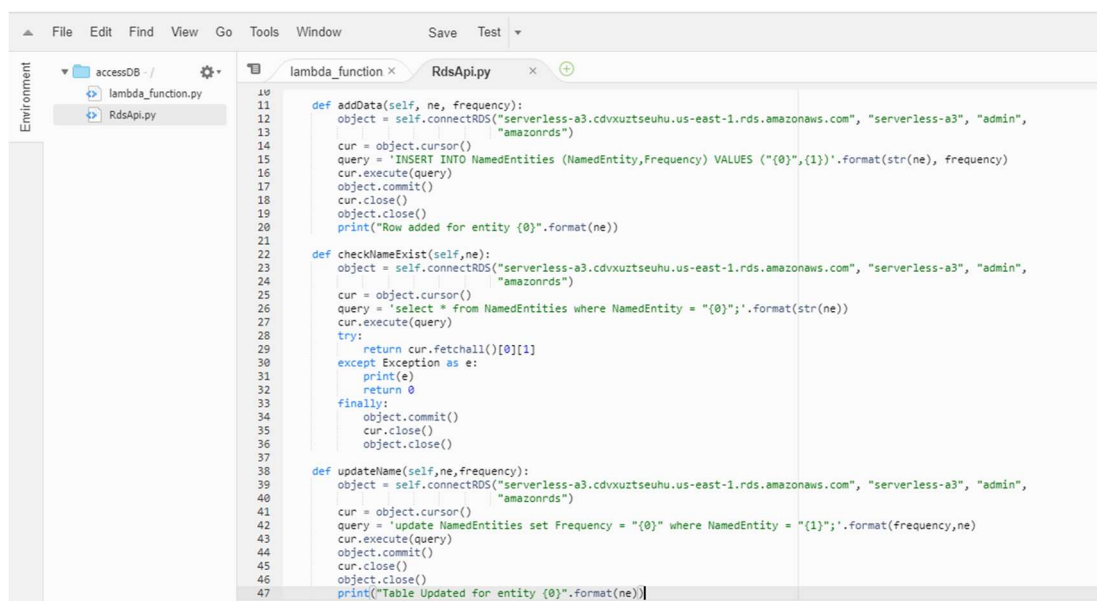
```

1 import json
2 import RdsApi
3 import boto3
4 def lambda_handler(event, context):
5     s3=boto3.client('s3')
6     if(event):
7         event_object = event["Records"][0]
8         file_name = str(event_object["s3"]["object"]["key"])
9         file_object=s3.get_object(Bucket='tagsb00845449',Key=file_name)
10        content = file_object["Body"].read().decode('utf-8')
11        content=json.loads(content)
12        dictionary = content[file_name.split('.')[0]]
13
14        api = RdsApi.RdsApi()
15        for i in dictionary.keys():
16            current_frequency=api.checkNameExist(i)
17            if(current_frequency==0):
18                api.addData(i,dictionary[i])
19            else:
20                api.updateName(i,current_frequency+dictionary[i])
21
22        print('table updated for object {}'.format(file_name))
23

```

Figure l accessDB lambda handler

Figure l shows accessDB lambda function script which fetches every object from tagsb00845449 bucket and saves it to AWS RDS SQL database.



```

10
11
12 def addData(self, ne, frequency):
13     object = self.connectRDS("serverless-a3.cdvoxztseuhu.us-east-1.rds.amazonaws.com", "serverless-a3", "admin",
14                               "amazonrds")
15     cur = object.cursor()
16     query = 'INSERT INTO NamedEntities (NamedEntity,Frequency) VALUES ("{}",{})'.format(str(ne), frequency)
17     cur.execute(query)
18     object.commit()
19     cur.close()
20     object.close()
21     print("Row added for entity {}".format(ne))
22
23 def checkNameExist(self,ne):
24     object = self.connectRDS("serverless-a3.cdvoxztseuhu.us-east-1.rds.amazonaws.com", "serverless-a3", "admin",
25                               "amazonrds")
26     cur = object.cursor()
27     query = 'select * from NamedEntities where NamedEntity = "{}"'.format(str(ne))
28     cur.execute(query)
29     try:
30         return cur.fetchall()[0][1]
31     except Exception as e:
32         print(e)
33         return 0
34     finally:
35         object.commit()
36         cur.close()
37         object.close()
38
39 def updateName(self,ne,frequency):
40     object = self.connectRDS("serverless-a3.cdvoxztseuhu.us-east-1.rds.amazonaws.com", "serverless-a3", "admin",
41                               "amazonrds")
42     cur = object.cursor()
43     query = 'update NamedEntities set Frequency = "{}" where NamedEntity = "{}"'.format(frequency,ne)
44     cur.execute(query)
45     object.commit()
46     cur.close()
47     object.close()
48     print("Table Updated for entity {}".format(ne))
49

```

Figure m RDSApi script

Figure m shows the RDSApi script used for the AccessDB lambda handler to work with RDS.

*g. accessDB is your 2nd Lambda function. This Lambda function reads each named entity JSON file and updates the MySQL database table (two fields - key, value).*

►	2020-06-26T11:36:36.318-03:00	Table Updated for entity Charles
►	2020-06-26T11:36:36.393-03:00	Table Updated for entity MacIntyre
►	2020-06-26T11:36:36.441-03:00	Table Updated for entity England
►	2020-06-26T11:36:36.497-03:00	Table Updated for entity Massively
►	2020-06-26T11:36:36.555-03:00	Table Updated for entity Multiplayer
►	2020-06-26T11:36:36.617-03:00	Table Updated for entity Role
►	2020-06-26T11:36:36.678-03:00	Table Updated for entity Playing
►	2020-06-26T11:36:36.768-03:00	Table Updated for entity Game
►	2020-06-26T11:36:36.828-03:00	Table Updated for entity MMORPG
►	2020-06-26T11:36:36.875-03:00	Table Updated for entity Stuart
►	2020-06-26T11:36:36.932-03:00	Table Updated for entity Stanton-Davies
►	2020-06-26T11:36:36.993-03:00	Table Updated for entity Huddersfield
►	2020-06-26T11:36:37.050-03:00	Table Updated for entity Pong
►	2020-06-26T11:36:37.114-03:00	Table Updated for entity Atari
►	2020-06-26T11:36:37.172-03:00	Table Updated for entity MMORPGs
►	2020-06-26T11:36:37.219-03:00	Table Updated for entity EverQuest
►	2020-06-26T11:36:37.276-03:00	Table Updated for entity Ultima
►	2020-06-26T11:36:37.335-03:00	Table Updated for entity EQ
►	2020-06-26T11:36:37.394-03:00	Table Updated for entity EverCrack
►	2020-06-26T11:36:37.452-03:00	Table Updated for entity E
►	2020-06-26T11:36:37.507-03:00	Table Updated for entity Hayot

*Figure n cloudwatch logs for accessDB*

Figure n shows logs which conveys table updated for every entity present in the files of tagsb00845449 bucket. accessDB is fetching every entity from files of tagsb00845449 and adding it to SQL database with their frequency. If the entity is already present, then the frequency is updated.

h. E.g. 001ne.txt contains “001ne”: {“Asia”:1, “Soviet”:1.....etc.}. Then this Lambda function will update a MySQL database table where “Asia” will be a value for field “NamedEntity”, and “1” will be the value for field “Frequency”. The field “NamedEntity” can be considered as the primary key.

The screenshot displays a MySQL database interface. On the left, the 'SCHEMAS' panel shows a tree view with 'innodb', 'serverless-a3' (selected), 'Tables', 'Views', 'Stored Procedures', 'Functions', and 'sys'. The 'serverless-a3' schema is expanded, showing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. Below this, the 'Administration' and 'Schemas' tabs are visible, with 'Schemas' selected. The 'Information' tab shows 'No object selected'.

The main area displays a SQL query in the 'Query 1' tab. The query is as follows:

```

1 • create database `serverless-a3`;
2 • use `serverless-a3`;
3 • drop table NamedEntities;
4
5 • CREATE TABLE NamedEntities
6 • (
7 •   NamedEntity varchar(200),
8 •   Frequency int
9 • );
10
11 • select * from NamedEntities;
12

```

Below the query, the 'Result Grid' shows the results of the query. The table has two columns: 'NamedEntity' and 'Frequency'. The results are as follows:

NamedEntity	Frequency
China	53
Chinese	4
Xinhua	1
News	107
Agency	4
October	26
December	51
Laws	3
August	17
Roads	3
NamedEntities	11

Figure 0 serverless-a3 database queries and entries

Figure 0 shows the database created from AWS RDS. The database contains a table with 2 columns ‘NamedEntity’ and ‘Frequency’. After the accessDB lambda handler is implemented, all the entities are stored in the database as shown in figure 0.



## **Part B - Paper Summary**

The architecture of cloud computing consists of the interconnection of different distributed systems and their usages. Many organizations have started using cloud technologies in this decade. But cloud computing is vulnerable to various kinds of privacy issues and malware attacks which creates a need for a strong identity and access management mechanism. The paper discusses the issues related to authentication, access management, and security, and the techniques to overcome these issues.

IAM in the cloud environment is a very important factor while migrating to cloud service. Lack of efficient mechanism in IAM will create multiple challenges like identity management, risk management, data security, privacy, data leakage, and transparency. Cloud service providers are responsible for these managements and they must ensure that data and applications are protected, and the infrastructure is secure. IAM systems perform different operations for providing security like authentication, authorization, storage, and verification provisions, the security of identity.

There are many different types of authentication mechanisms are physical security mechanisms( access cards and biometrics), digital security mechanisms (credentials, SSH keys, multifactor authentications), PIN (asymmetric encryption technology), SSO (one password for all cloud applications). SSH keys are not better if the private keys are not secured whereas their advantage is authentication is performed without passing the password. There are many types of SSO methods are Enterprise SSO (maintaining session cookies), OpenID (the open standard protocol that allows user's authentication to the relying parties), OAuth (one way or mutual way authentication), SAML(works on token-based request-response services).

There are different types of authorization mechanisms for access control are MAC, DAC, RBAC, ABAC. MAC permits OS or kernel which are set by system managers, DAC controls permissions through data owner, RBAC provides access rights based on roles and privileges of users, and ABAC defines access control by use of policies.

Identity and access management authenticates users, devices, and services and provides rights to grant or deny data. It controls access to resources based on predefined policies. IAM also does Authentication management and manages the password and digital certificates. SPML is a framework used for identity management in IAM. After authentication, IAM manages authority and ensures that the resources are secure and accessed concerning the policies. Privacy and interoperability are the main issues in existing IAM approaches. Many organizations such as IBM, Oracle, RSA, SailPoint provide the IAM system to secure the information by controlling access permission of the users. Each organization uses different ways and techniques to achieve the best possible security and uses a variety of myriad solutions.

Threats in the cloud infrastructure include data security, virus, availability of resources, and multitenancy. Due to the increased processing capacity of devices, security is mostly compromised using a brute-force attack. Viruses spread to all the users of the cloud if one user gets malware on their local system. The availability of resources is impacted in the event of a security breach. There are many different threats and attacks possible in cloud services discussed thoroughly in the paper and proper precautions should be taken for it.

### My views

Cloud computing is one of the most important areas for data storage and processing, but there are scopes of vulnerabilities for authorization, authentication, data privacy, and security. IAM plays a major role in managing the security of users, resources, and applications/services. The paper describes a lot of different techniques used by cloud providers to keep our applications and services secure on the cloud and different organizations have different solutions to achieve the same, but while using the cloud services, we can take many precautions or use better approaches to maintain security. For authorization, we can use multifactor authentication along with our normal techniques. SSH shells must be used while developing and using the application. ABAC mechanism to be used to create specific policies for specific resources while working on different cloud services. Cloud computing services will not be secure if users' local system security will be compromised, hence we should keep our systems safe from viruses/malware.

## References

- [1]. 2020. [online] Available at: <<https://www.youtube.com/watch?v=EsqjHDpLpB4>> [Accessed 26 June 2020].
- [2]. Docs.aws.amazon.com. 2020. *What Is AWS Lambda? - AWS Lambda*. [online] Available at: <<https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>> [Accessed 26 June 2020].
- [3]. 2020. [online] Available at: <<https://www.youtube.com/watch?v=vXiZO1c5Sk0>> [Accessed 26 June 2020].
- [4]. Programcreek.com. 2020. *Nltk.Ne\_Chunk Python Example*. [online] Available at: <[https://www.programcreek.com/python/example/91258/nltk.ne\\_chunk](https://www.programcreek.com/python/example/91258/nltk.ne_chunk)> [Accessed 26 June 2020].
- [5]. Greene, D. and Cunningham, P., 2006. *Practical Solutions To The Problem Of Diagonal Dominance In Kernel Document Clustering*. [online] Available at: <<https://dl.acm.org/doi/10.1145/1143844.1143892>> [Accessed 26 June 2020].
- [6]. Indu, P.M. Rubesh Anand, Vidhyacharan Bhaskar, "Identity and access management in cloud environment: Mechanisms and challenges", Engineering Science and Technology, an International Journal, Volume 21, Issue 4, 2018, Pages 574-588, ISSN 2215-0986, <https://doi.org/10.1016/j.jestch.2018.05.010>