Event-driven serverless application using AWS Lambda.

Step 1: Login to AWS Credentials and open S3, Lambda and Dynamo DB as shown in Fig 1,2,3.

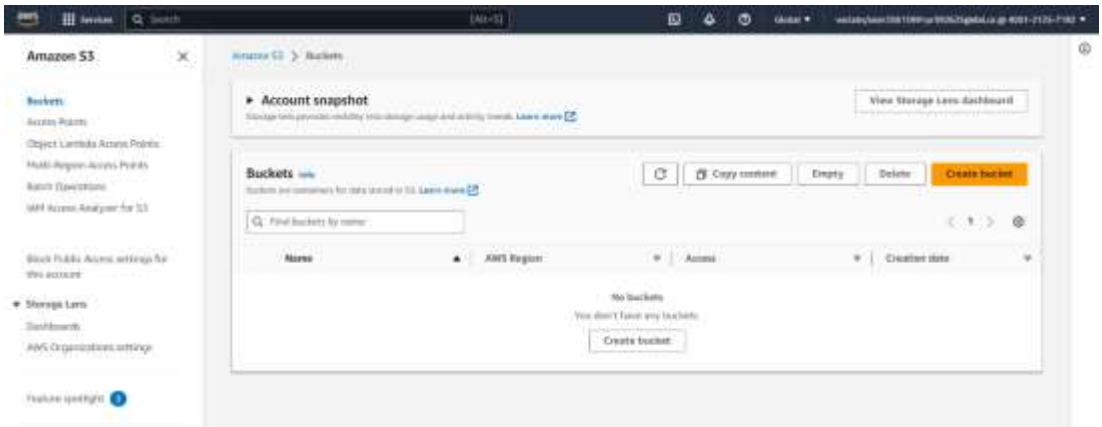

Fig 1

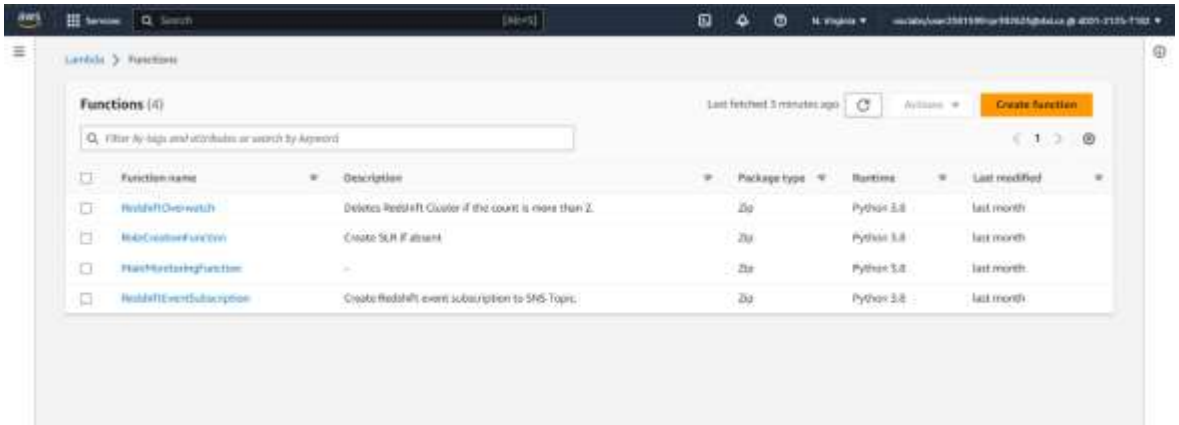

Fig 2

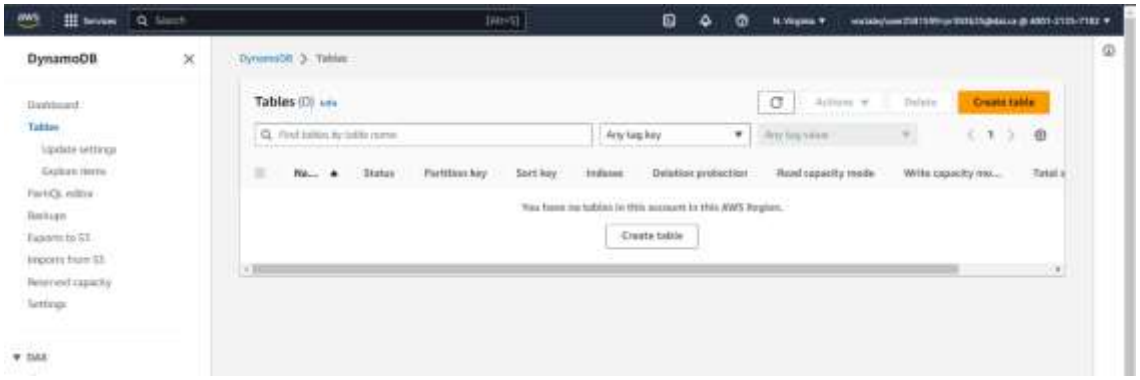

Fig 3

Step 2: Now execute the code to create S3 bucket as shown in Fig 4 and we can see the buckets named sampleb00913117 and tagsb00913117 have been created in Fig 5 and in Fig 6, 7 we can see that they are empty as we just created it.

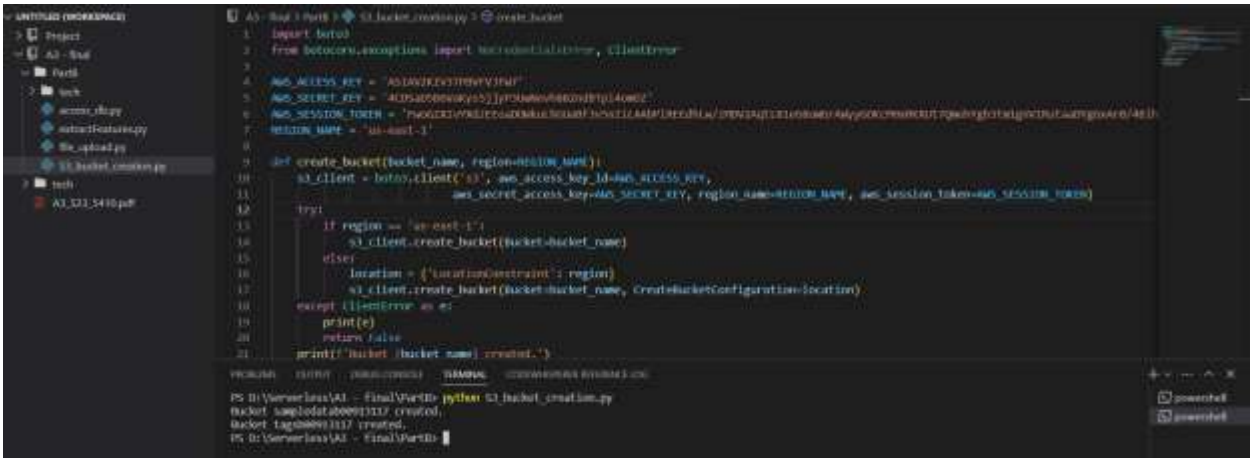

Fig 4

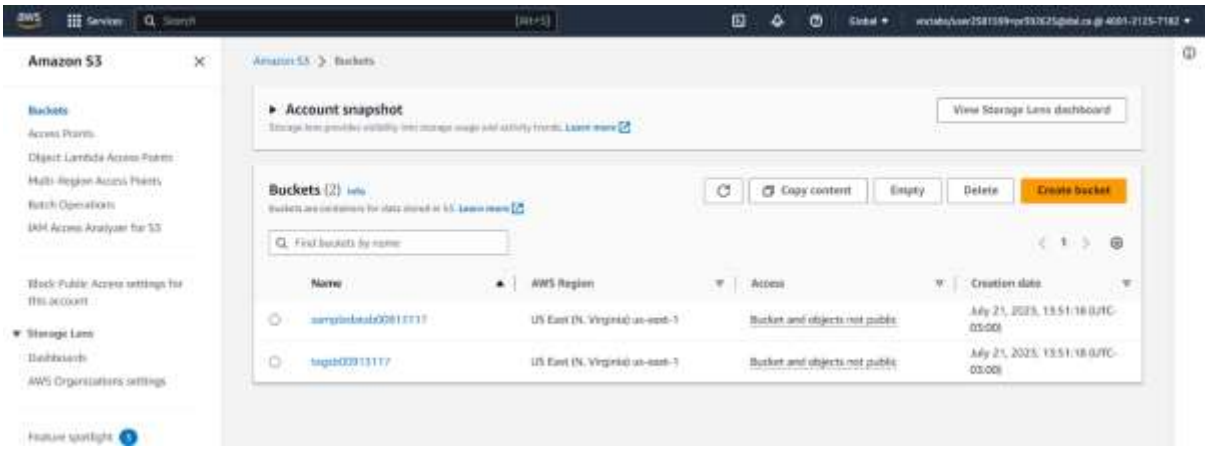

Fig 5

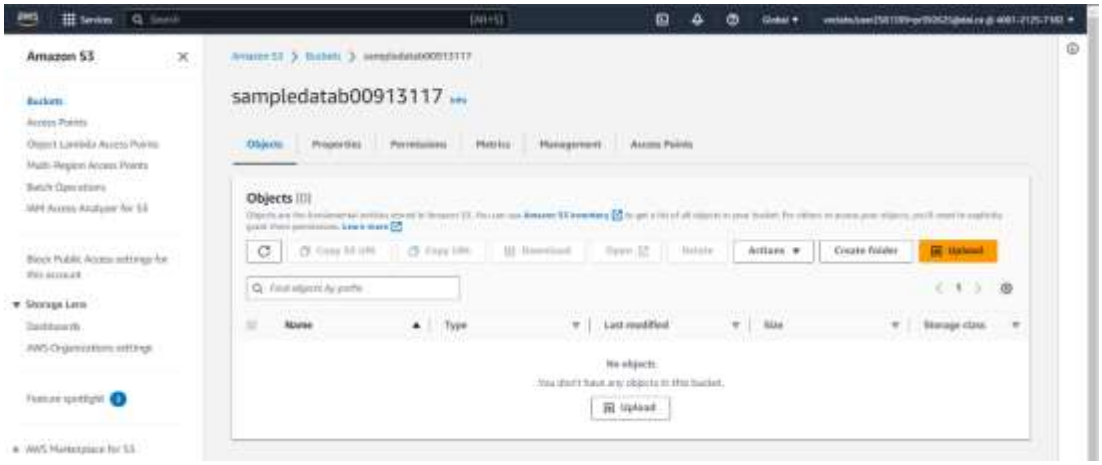

Fig 6

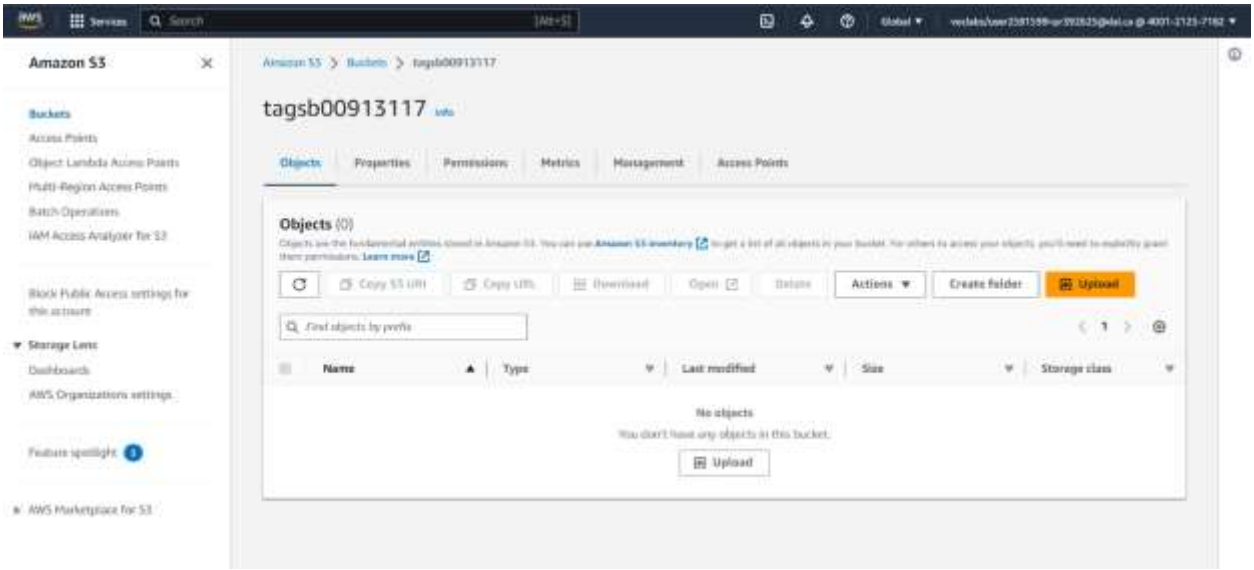

Fig 7

Step 3: Now we will create a Lambda Function named **extractFeatures** as shown in Fig 8, 9.

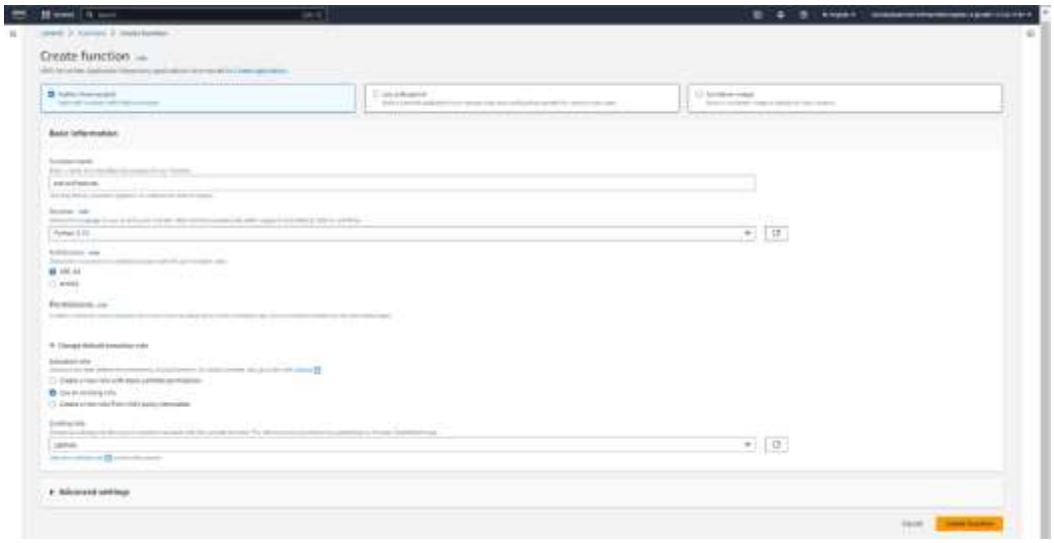

Fig 8

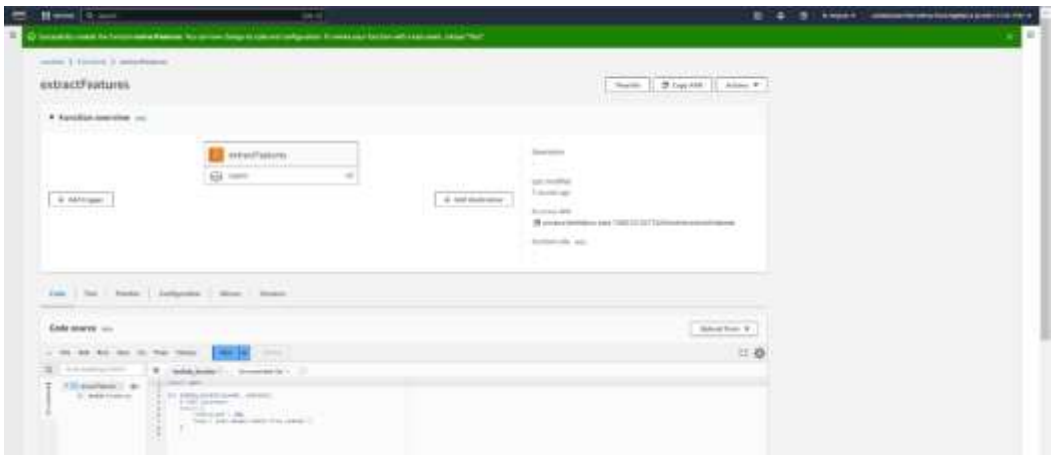

Fig 9

Step 4: Now we will add Trigger to this Lambda function and find S3 as shown in Fig 9 because we want to trigger our lambda function if any file is uploaded in S3 sampledatab00913117 as shown in Fig 10,11,11(a). This lambda function will extract the Named entities from the file and create a JSON array of named entities* for that file. This file will be saved as 001ne.txt in tagsb00913117.

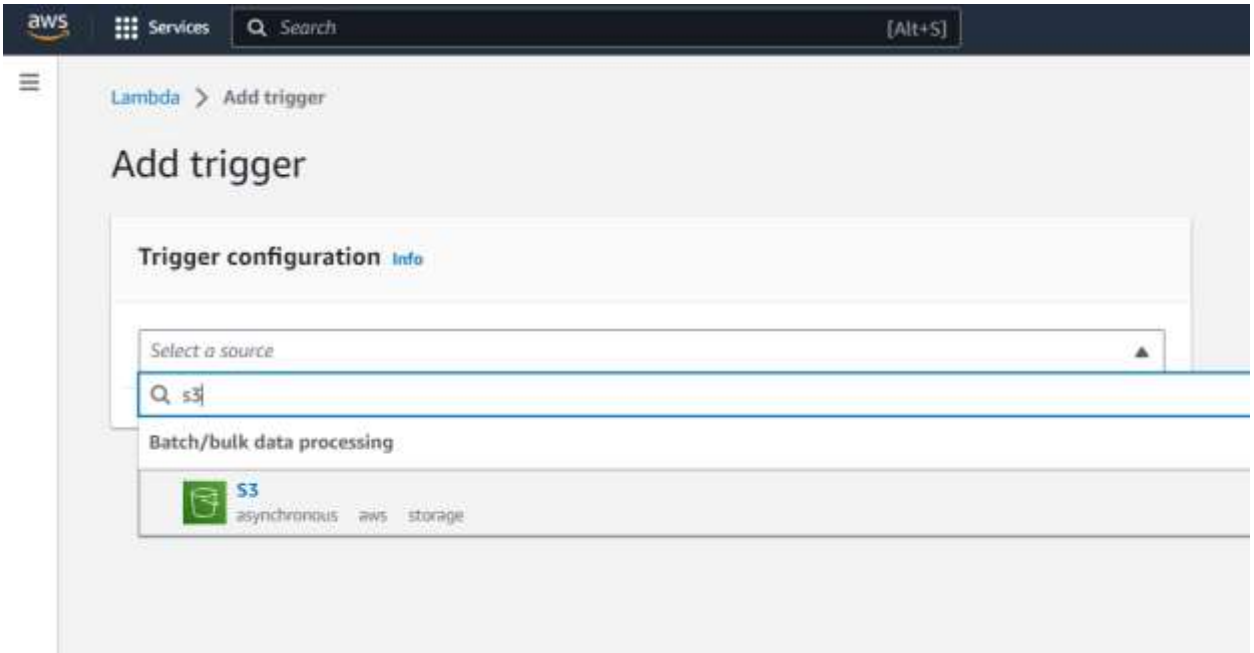

Fig 10

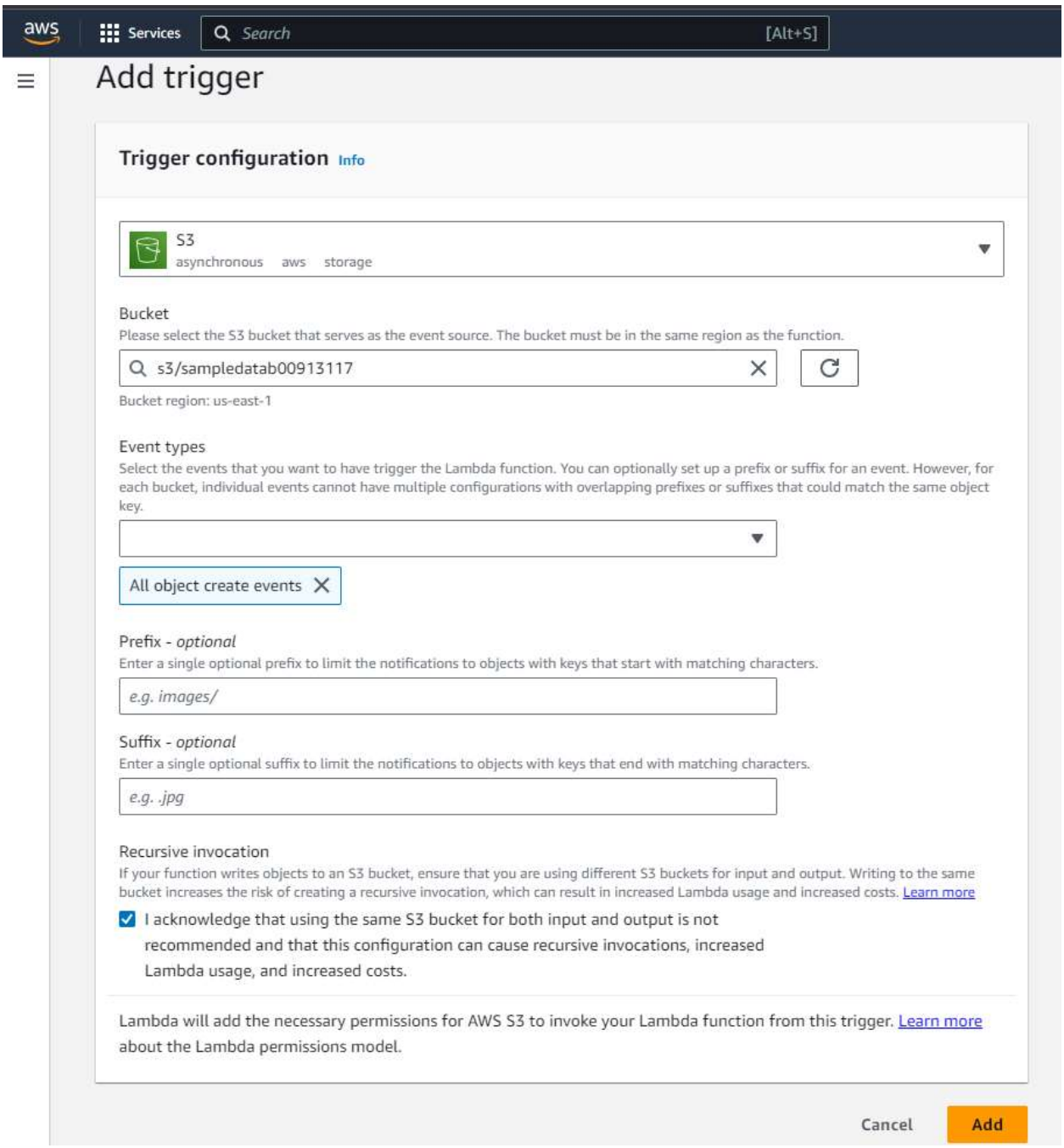
aws
Services
Search
[Alt+S]
Add trigger
Trigger configuration Info
S3
asynchronous aws storage
Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.
s3/sampledatab00913117
Bucket region: us-east-1
Event types
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.
All object create events
Prefix - optional
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.
e.g. images/
Suffix - optional
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.
e.g. .jpg
Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. Learn more
I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.
Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.
Cancel
Add

Fig 11

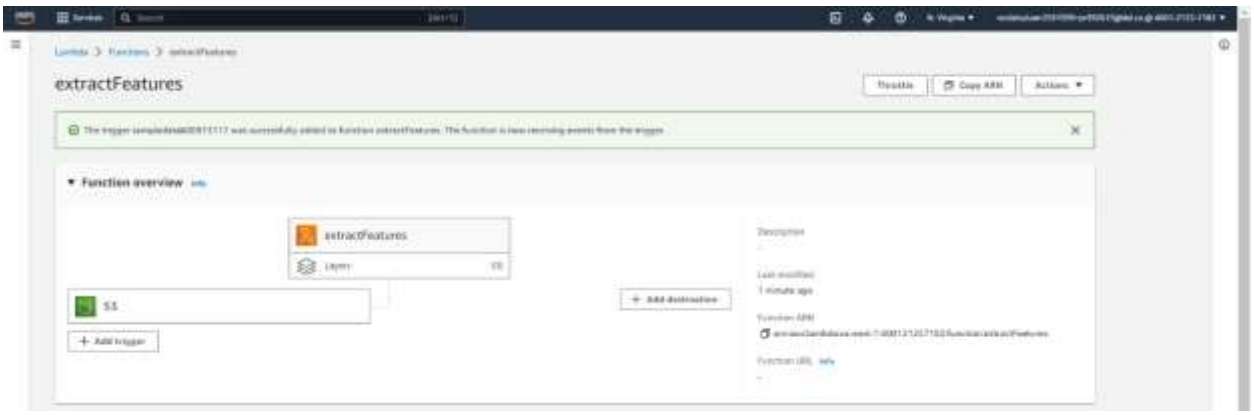

Fig 11(a)

Step 5: We will now write the code to extract features in the code snippet below as shown in Fig 12 and then click on deploy as shown in Fig 13.

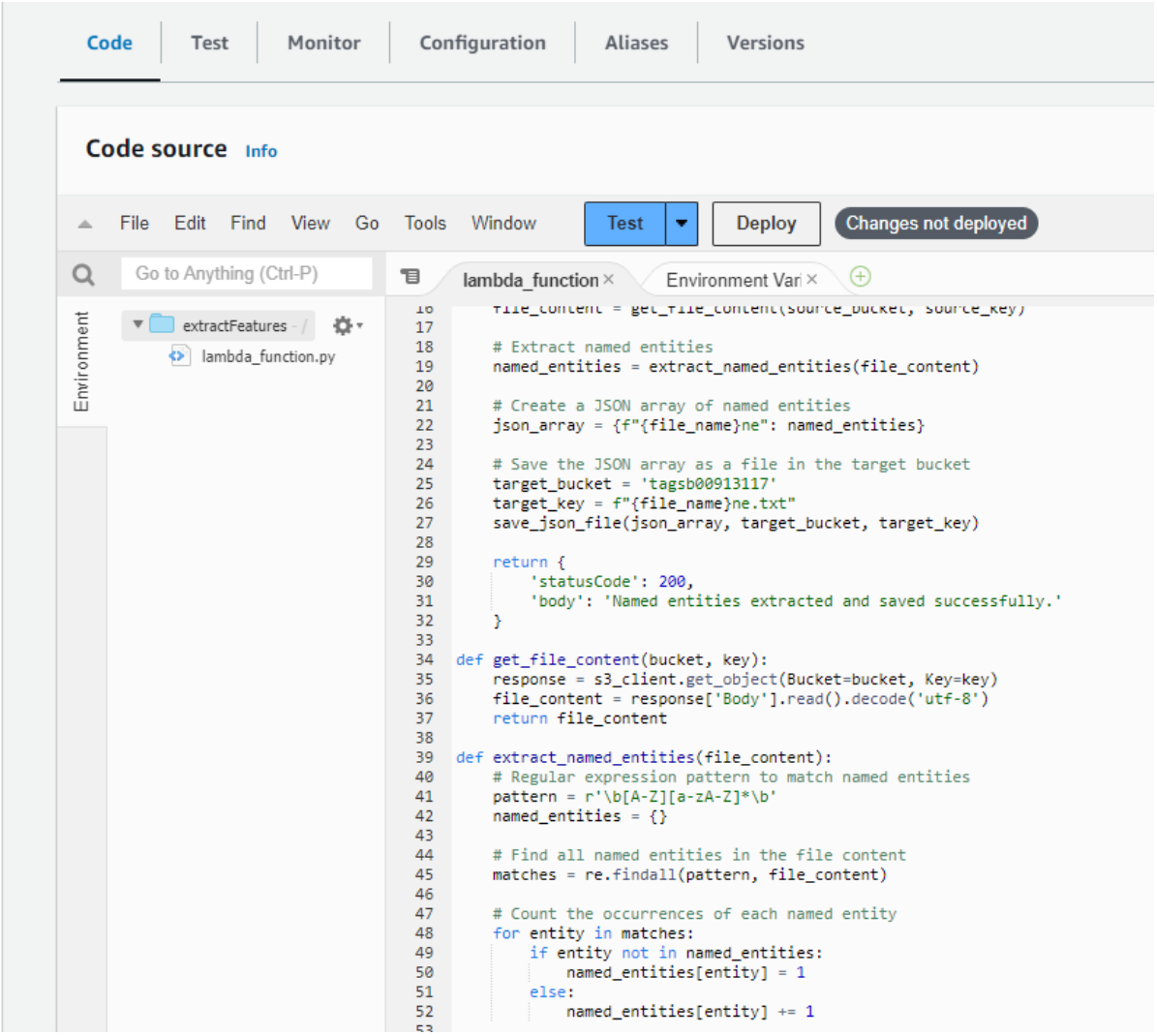

Fig 11

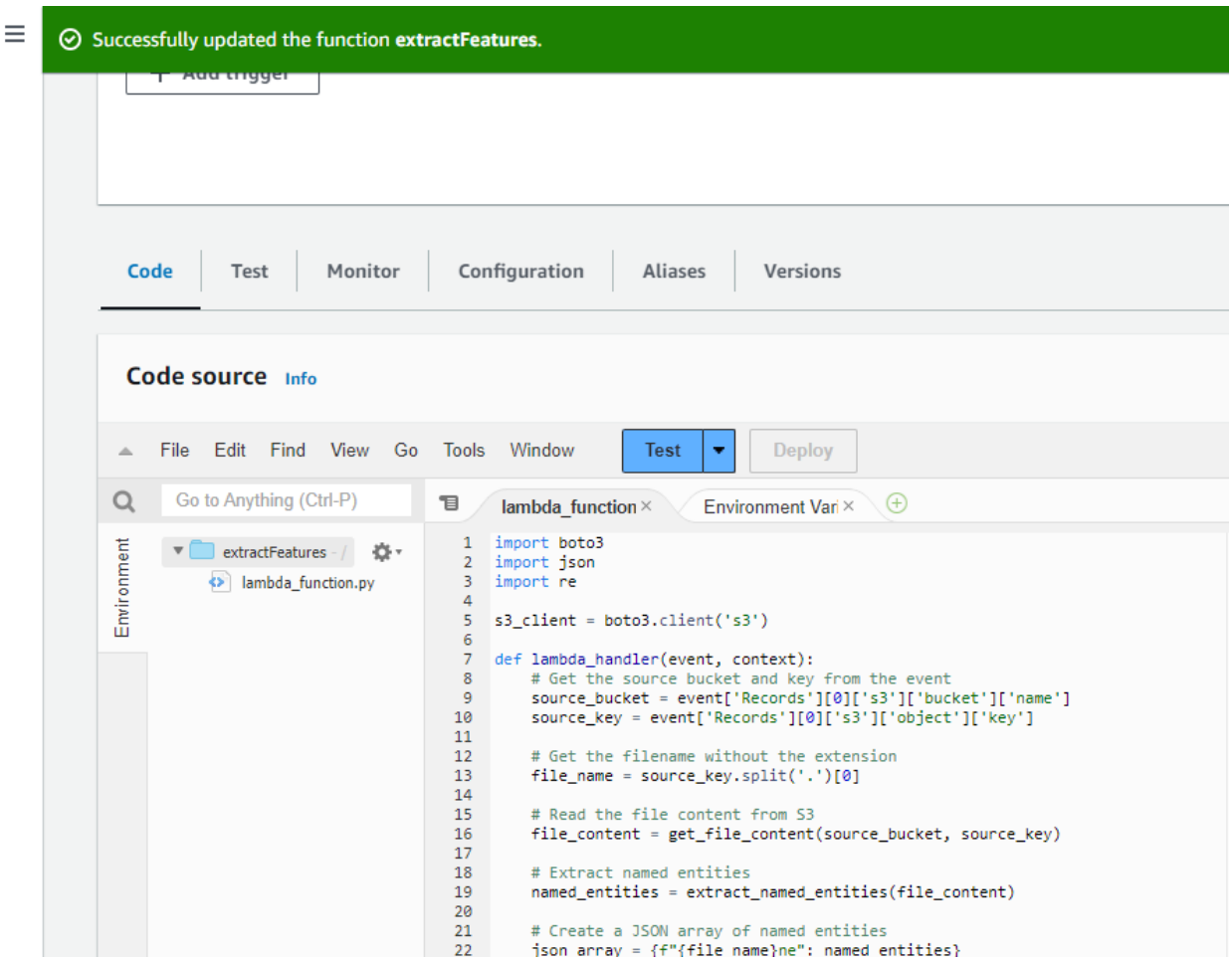

Fig 12

Code added:

```
import boto3
import json
import re

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Get the source bucket and key from the event
    source_bucket = event['Records'][0]['s3']['bucket']['name']
    source_key = event['Records'][0]['s3']['object']['key']

    # Get the filename without the extension
    file_name = source_key.split('.')[0]

    # Read the file content from S3
    file_content = get_file_content(source_bucket, source_key)
```

```
    # Extract named entities
    named_entities = extract_named_entities(file_content)

    # Create a JSON array of named entities
    json_array = {f"{file_name}ne": named_entities}

    # Save the JSON array as a file in the target bucket
    target_bucket = 'tagsb00913117'
    target_key = f"{file_name}ne.txt"
    save_json_file(json_array, target_bucket, target_key)

    return {
        'statusCode': 200,
        'body': 'Named entities extracted and saved successfully.'
    }

def get_file_content(bucket, key):
    response = s3_client.get_object(Bucket=bucket, Key=key)
    file_content = response['Body'].read().decode('utf-8')
    return file_content

def extract_named_entities(file_content):
    # Regular expression pattern to match named entities
    pattern = r'\b[A-Z][a-zA-Z]*\b'
    named_entities = {}

    # Find all named entities in the file content
    matches = re.findall(pattern, file_content)

    # Count the occurrences of each named entity
    for entity in matches:
        if entity not in named_entities:
            named_entities[entity] = 1
        else:
            named_entities[entity] += 1

    return named_entities

def save_json_file(data, bucket, key):
    json_content = json.dumps(data)
    s3_client.put_object(Body=json_content, Bucket=bucket, Key=key)
```

Step 6: Now we will create another function named accessDB to connect to DynamoDB as shown in Fig 13, 14, 15.

Fig 13

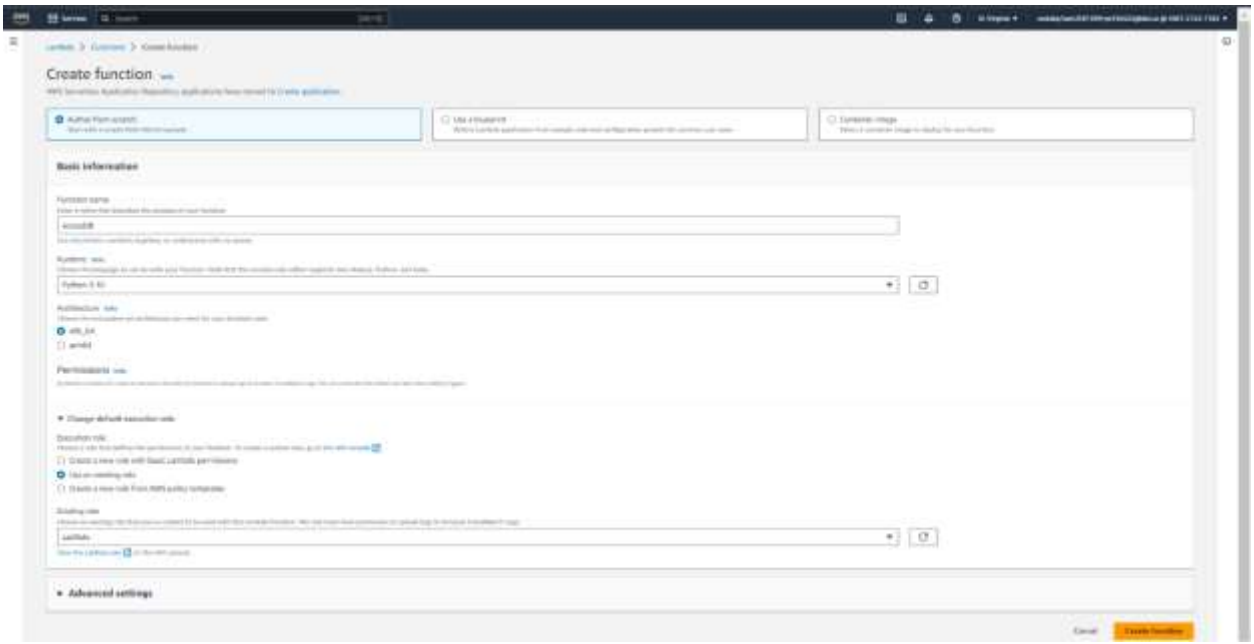

Fig 14

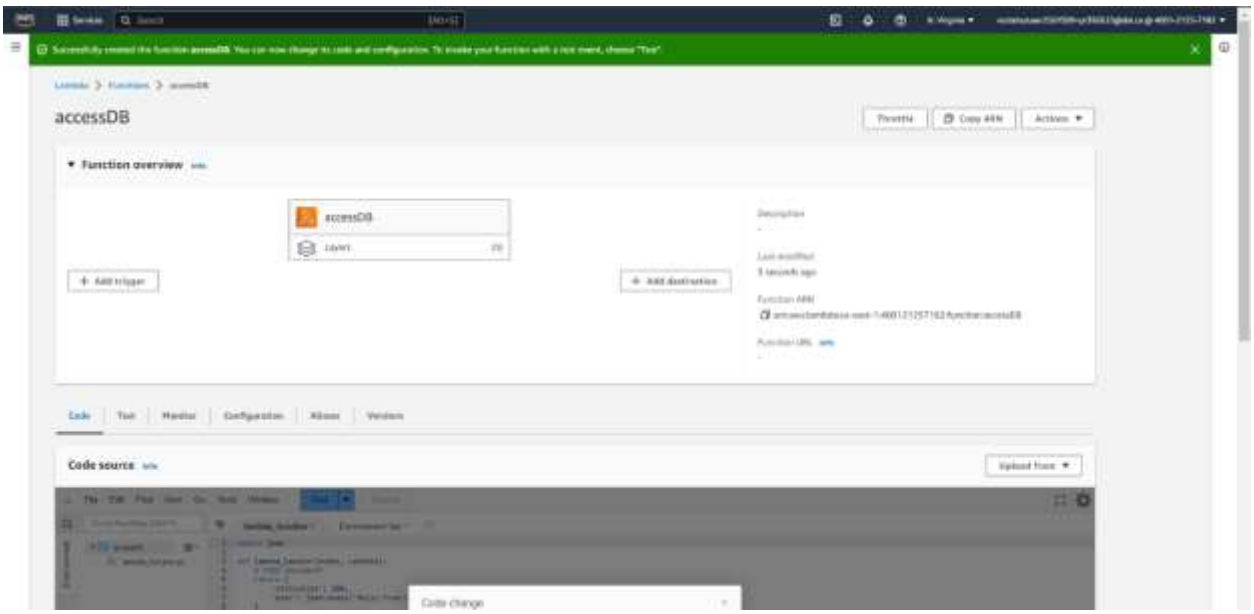

Fig 15

Code added:

```
import boto3
import json
import logging

dynamodb_client = boto3.client('dynamodb')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Get the bucket name and key from the event
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Read the named entities JSON file from S3
    try:
        named_entities = read_named_entities(bucket, key)
    except json.JSONDecodeError as e:
        logger.error(f"Error decoding JSON file: {e}")
        return {
            'statusCode': 500,
            'body': 'Error decoding JSON file.'
        }

    # Update the DynamoDB table
```

```
    update_dynamodb(named_entities)

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

def read_named_entities(bucket, key):
    s3_client = boto3.client('s3')
    response = s3_client.get_object(Bucket=bucket, Key=key)
    named_entities_content = response['Body'].read().decode('utf-8')

    try:
        named_entities = json.loads(named_entities_content)
        return named_entities
    except json.JSONDecodeError as e:
        raise e

def update_dynamodb(named_entities):
    table_name = 'fileDb'

    for key, value in named_entities.items():
        for sub_key, sub_value in value.items():
            dynamodb_client.put_item(
                TableName=table_name,
                Item={
                    'key': {'S': sub_key},
                    'value': {'S': str(sub_value)}
                }
            )
```

Step 7: Like the first function we will add trigger with S3 and select tagsb00913117 as shown in Fig 16. Also, we will add the code in the code snippet down and deploy it as shown in Fig 17. This lambda function will be triggered when something is stored in tagsb00913117 and will store it in the DynamoDB table which we will create next.

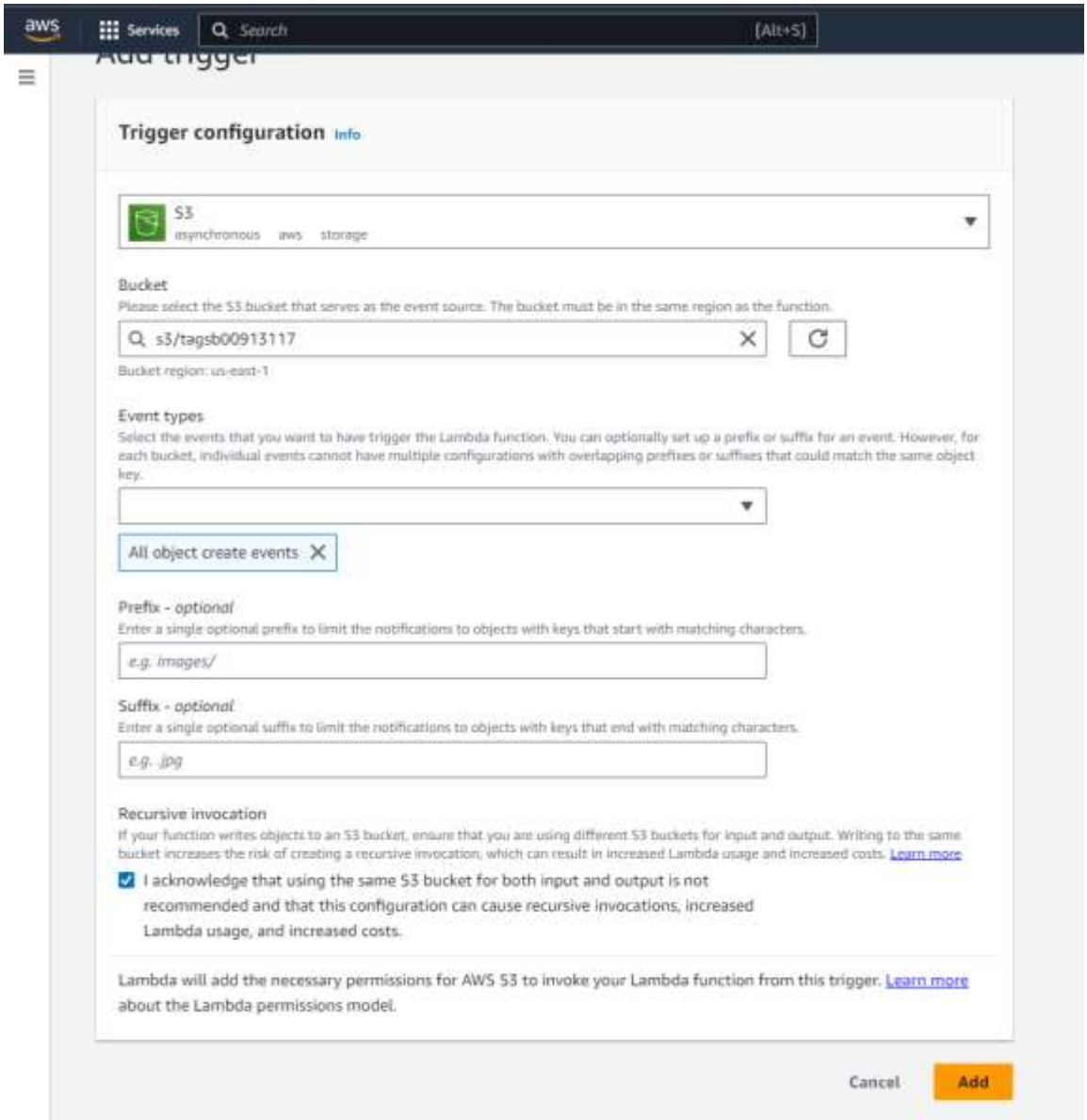
aws
Services
Search
[Alt+S]
Add trigger
Trigger configuration Info
S3
asynchronous aws storage
Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.
s3/tagsb00913117
Bucket region: us-east-1
Event types
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.
All object create events
Prefix - optional
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.
e.g. images/
Suffix - optional
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.
e.g. .jpg
Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. Learn more
I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.
Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.
Cancel
Add

Fig 16

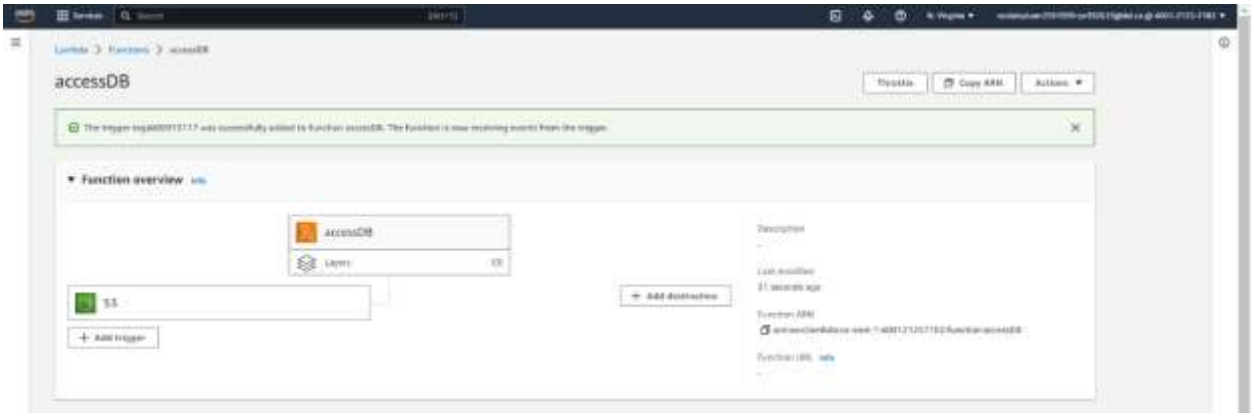
accessDB
Function overview
S3

Fig 17

Step 8: We will open DynamoDB and create a table named fileDb. We will add Partition key as “key” as shown in Fig 18, 19.

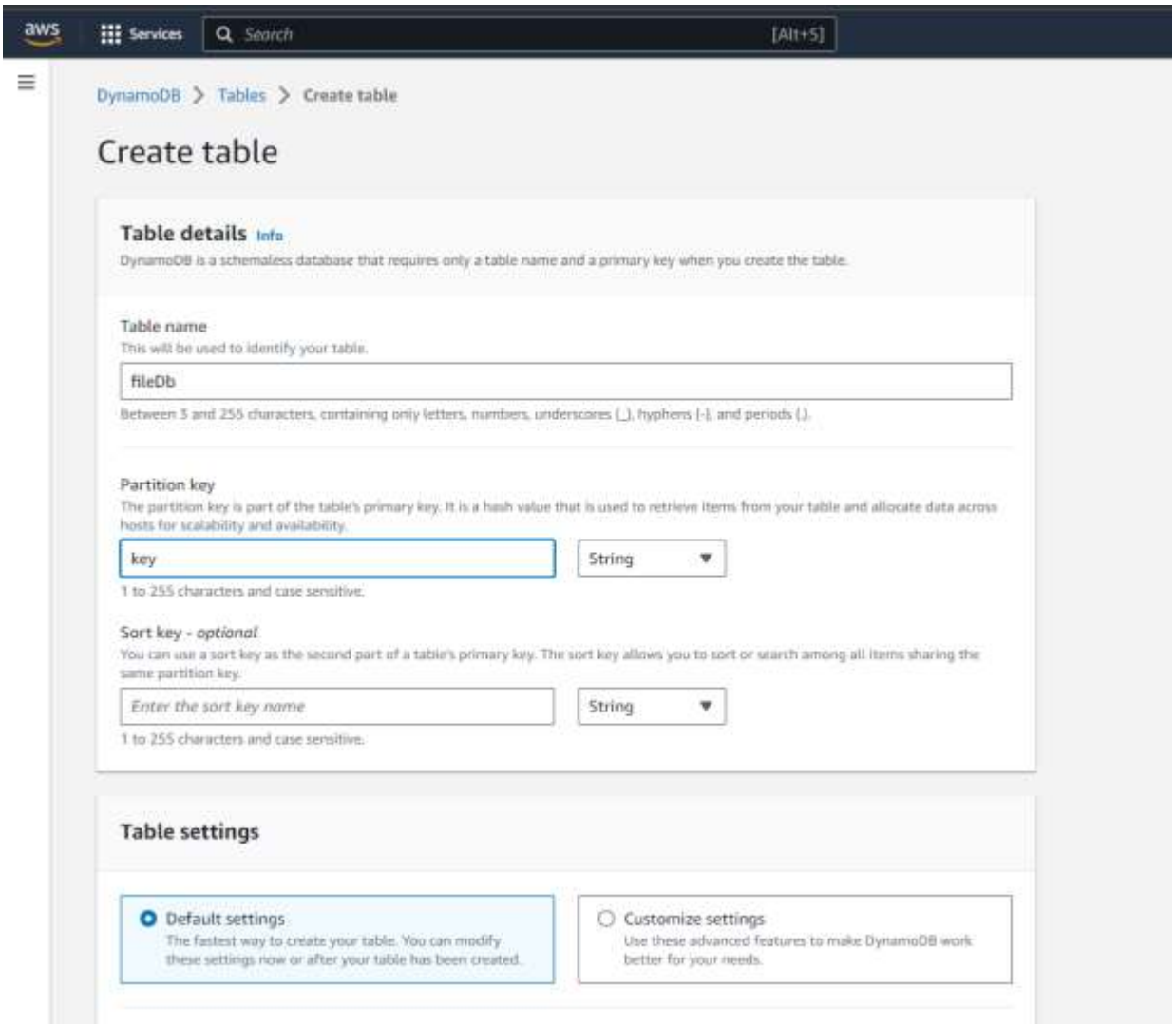

Fig 18

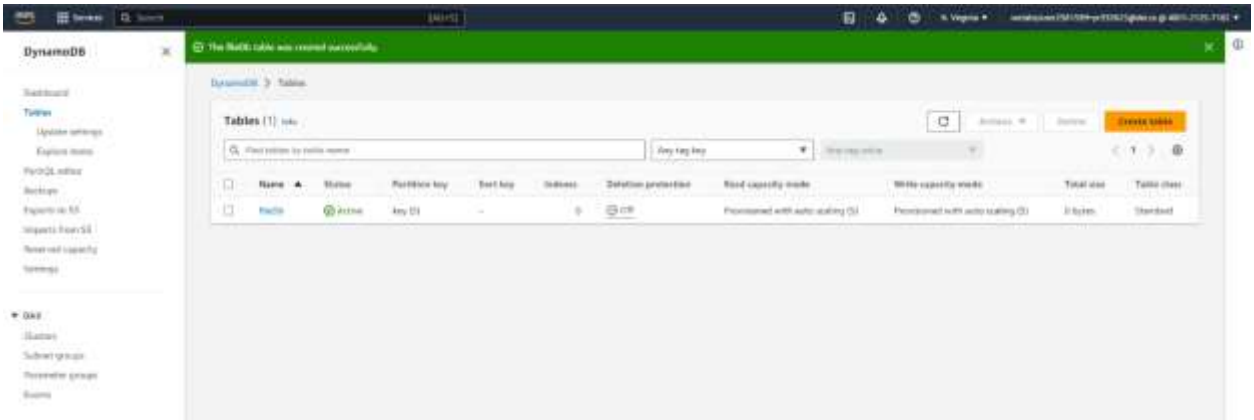

Fig 19

Step 9: Now we will run the code file_upload.py which will upload all the 401 files in tech folder to S3 buckets as shown in Fig 20, 21 and we will see that both the lambda functions are also triggered by opening the cloud watch monitor logs as shown in Fig 22, 23(a), 23(b) 24, 25(a), 25(b).

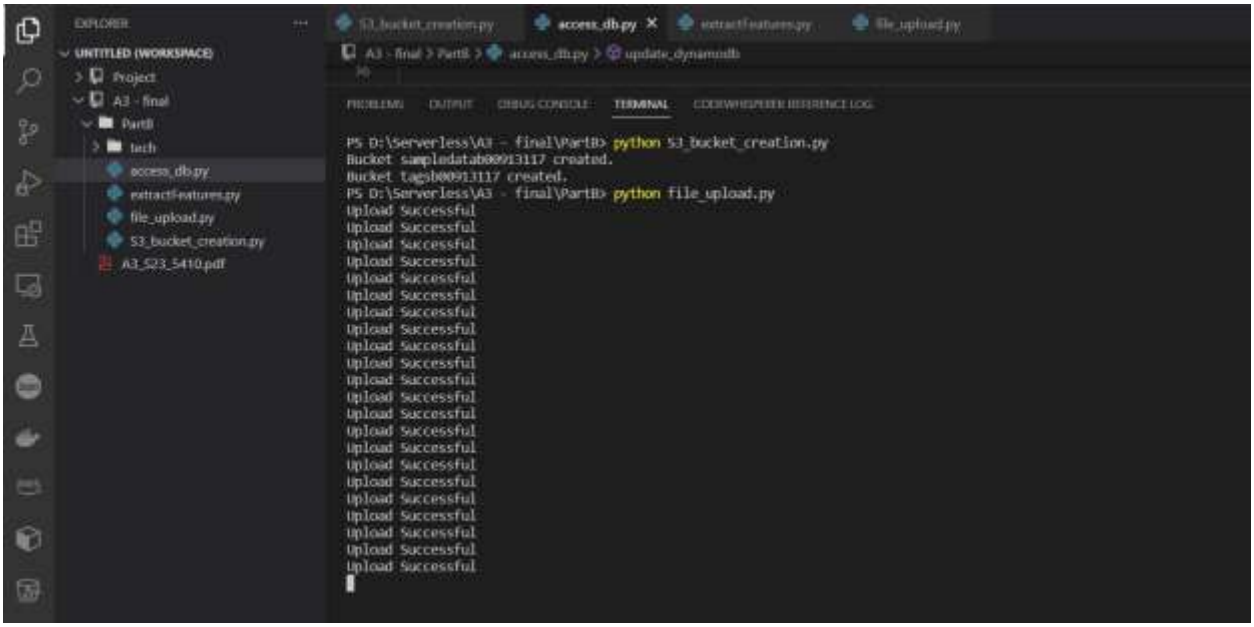

Fig 20

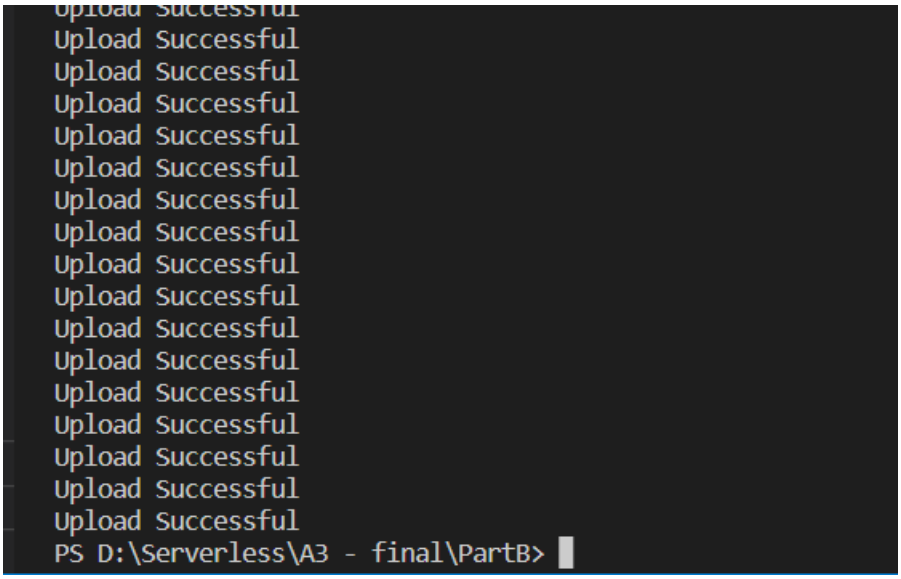

Fig 21

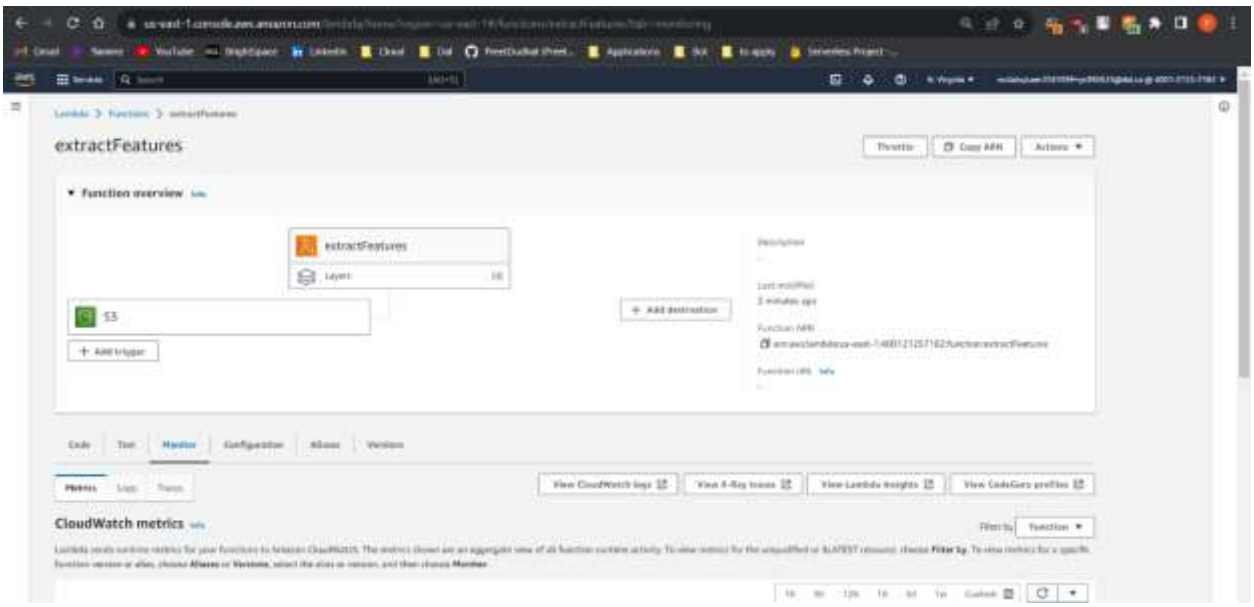

Fig 22

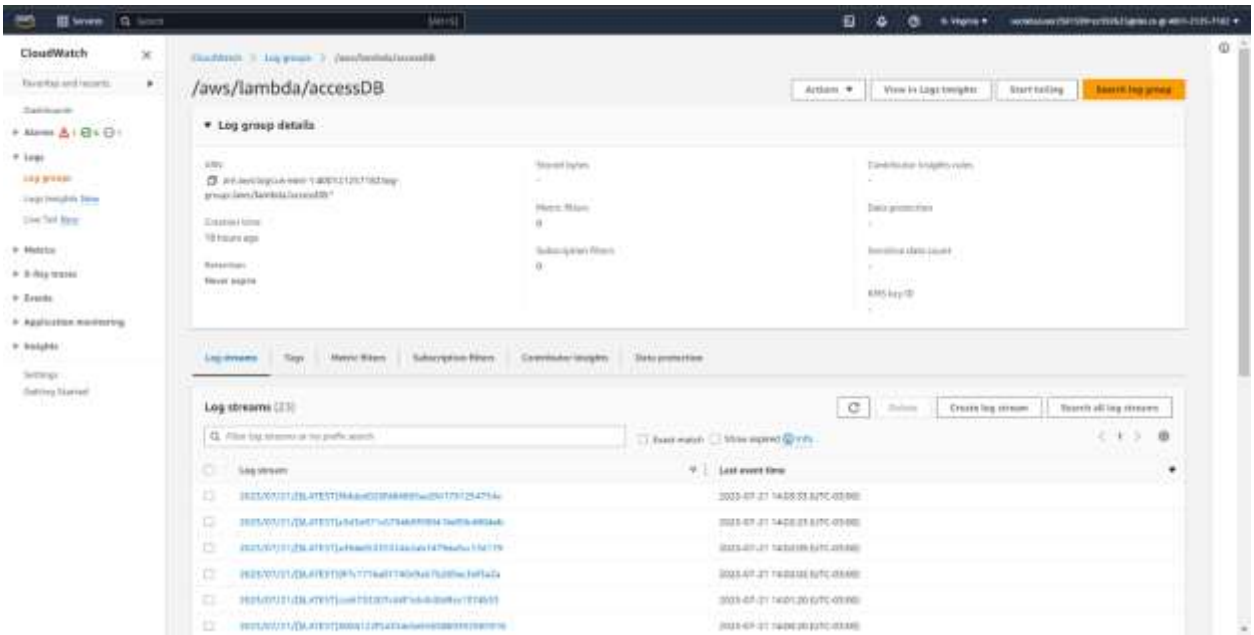

Fig 23(a)

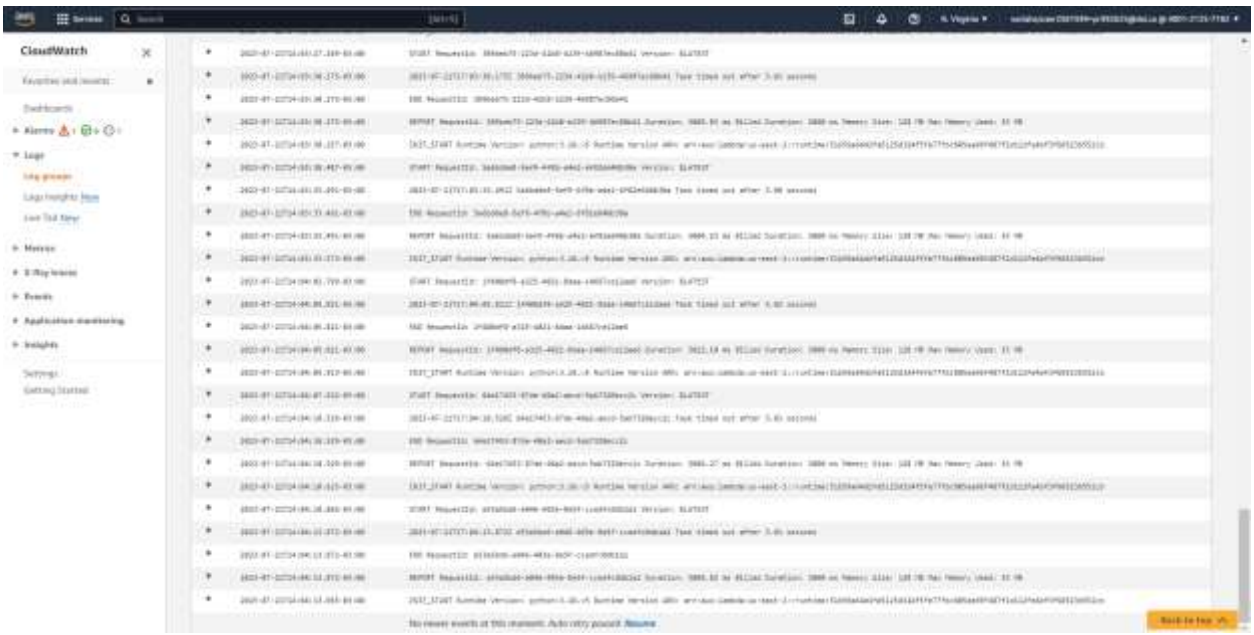

Fig 23(b)

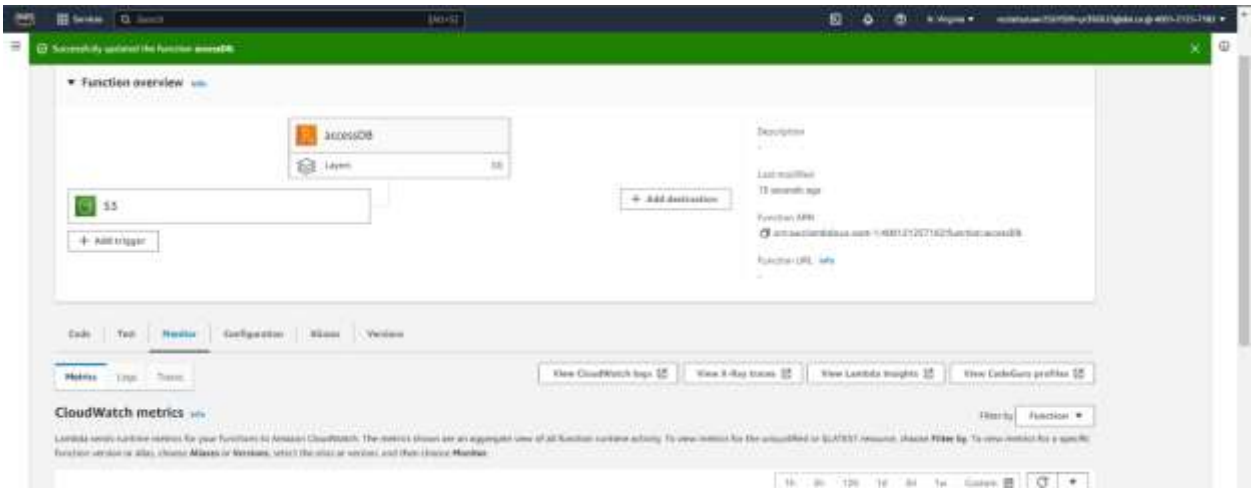

Fig 24

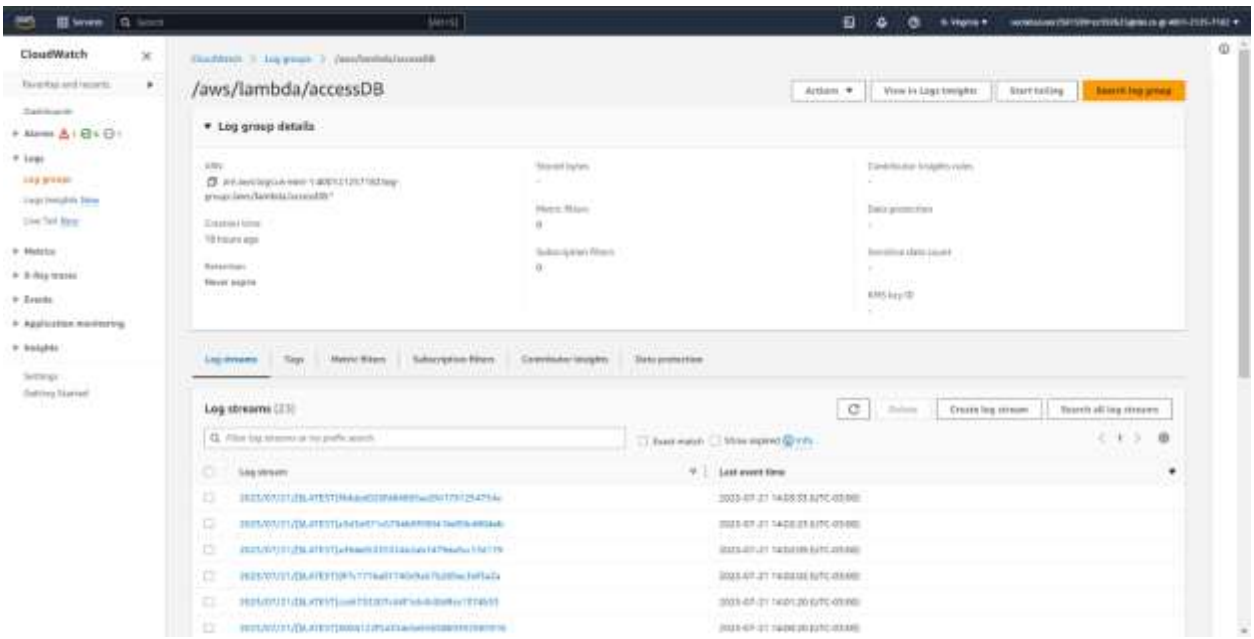

Fig 25(a)

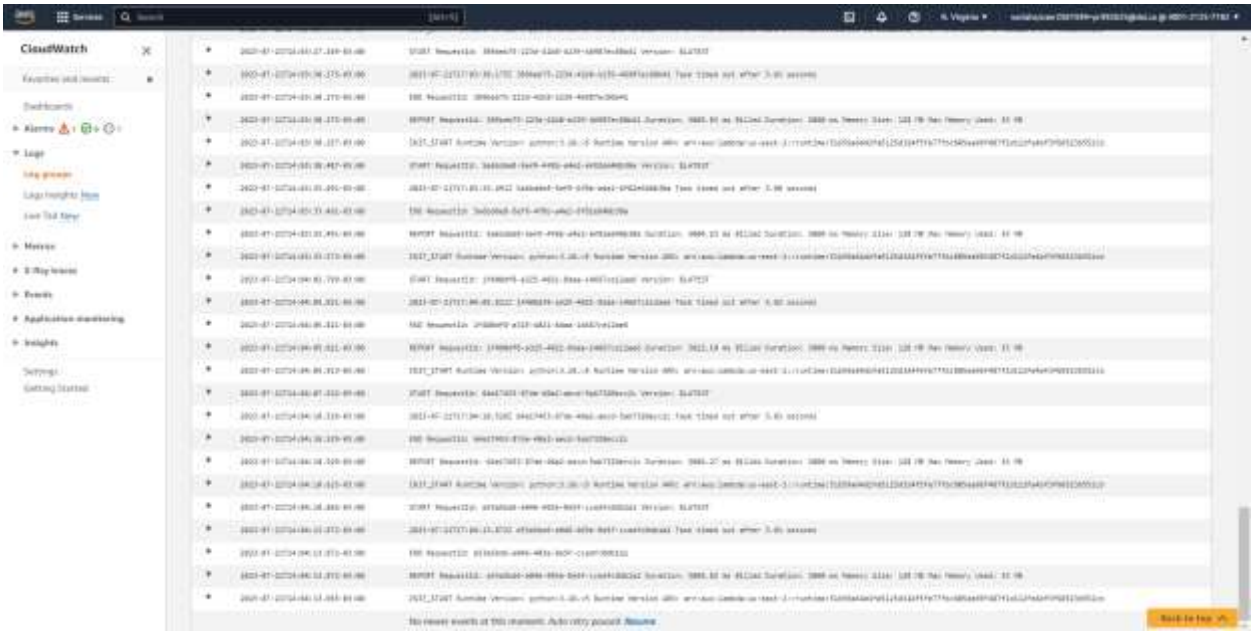

Fig 25(b)

Step 10: We can see in the S3 buckets the files have been uploaded as shown in Fig 26, 27.

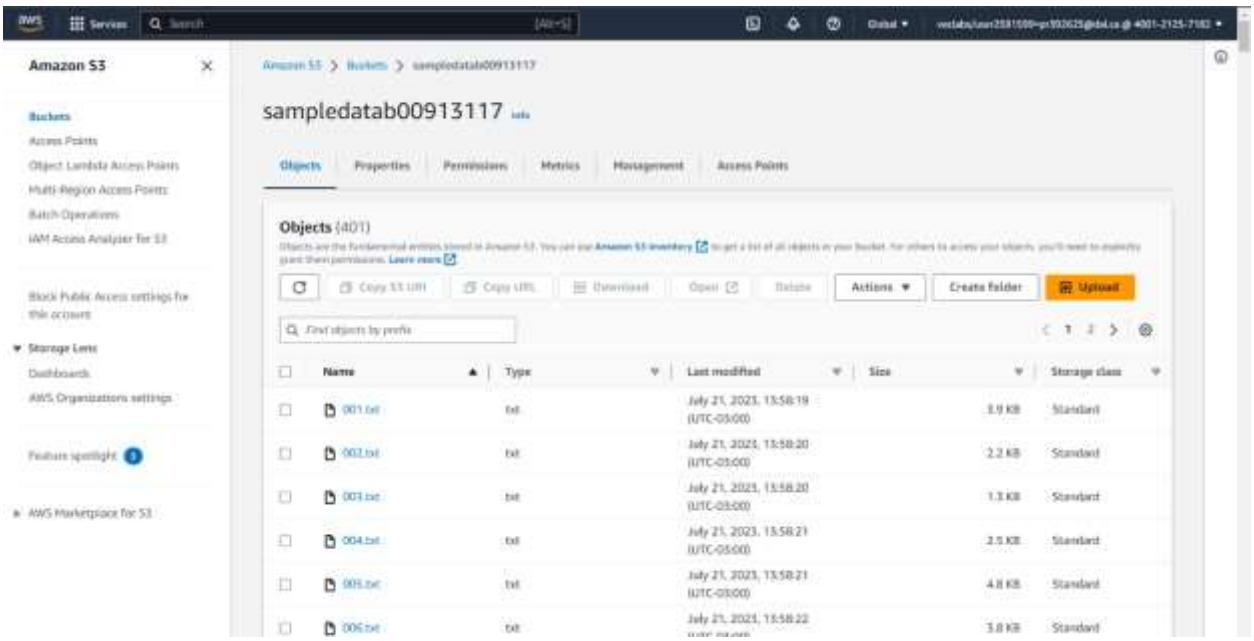

Fig 26

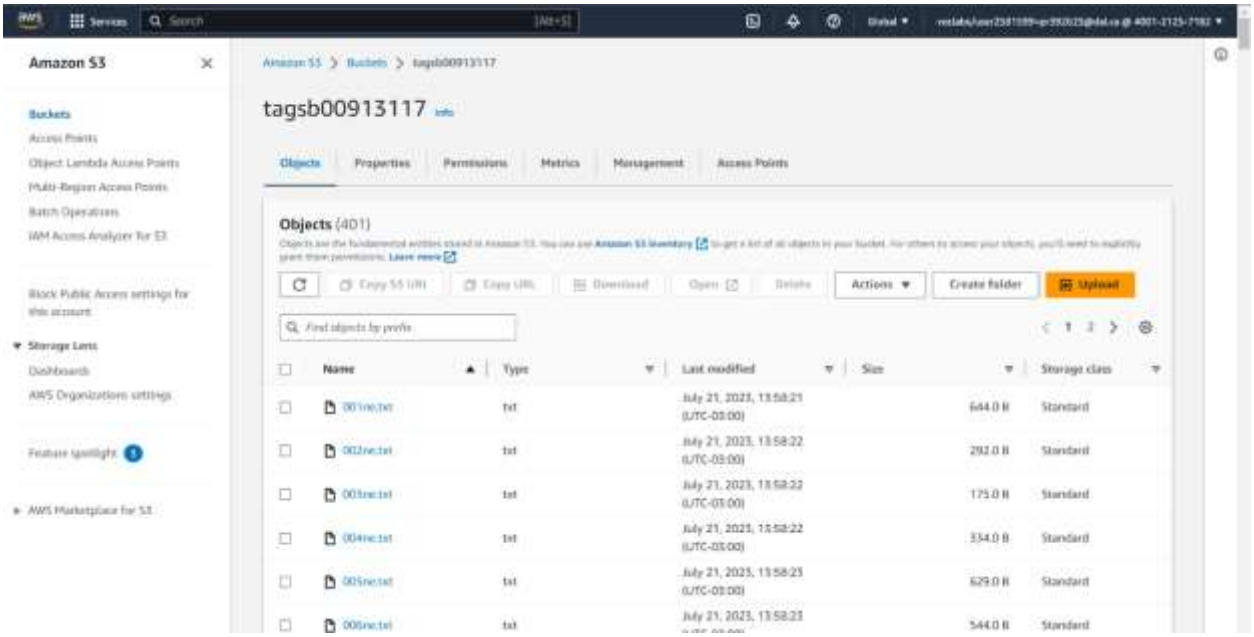

Fig 27

Step 11: We will open our fileDB and explore the table items as shown in Fig 28, 29, 30. We can see that there are more than 1400 items fetched.

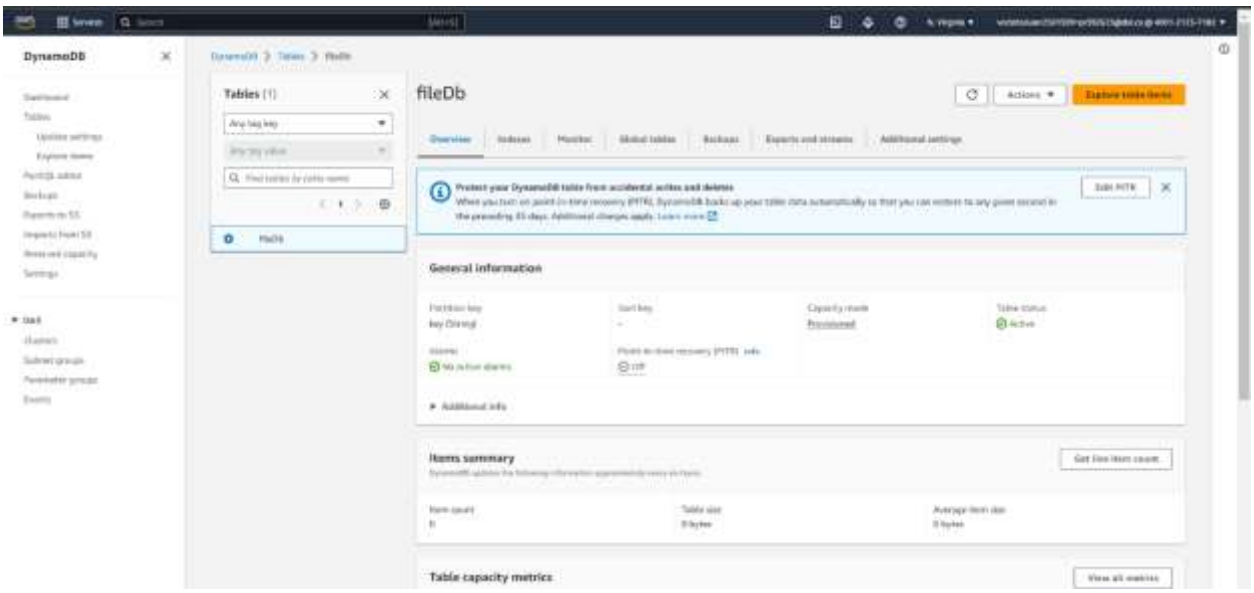
DynamoDB
Tables (1)
fileDb
Actions
Overview
General information
Items summary
Table capacity metrics

Fig 28

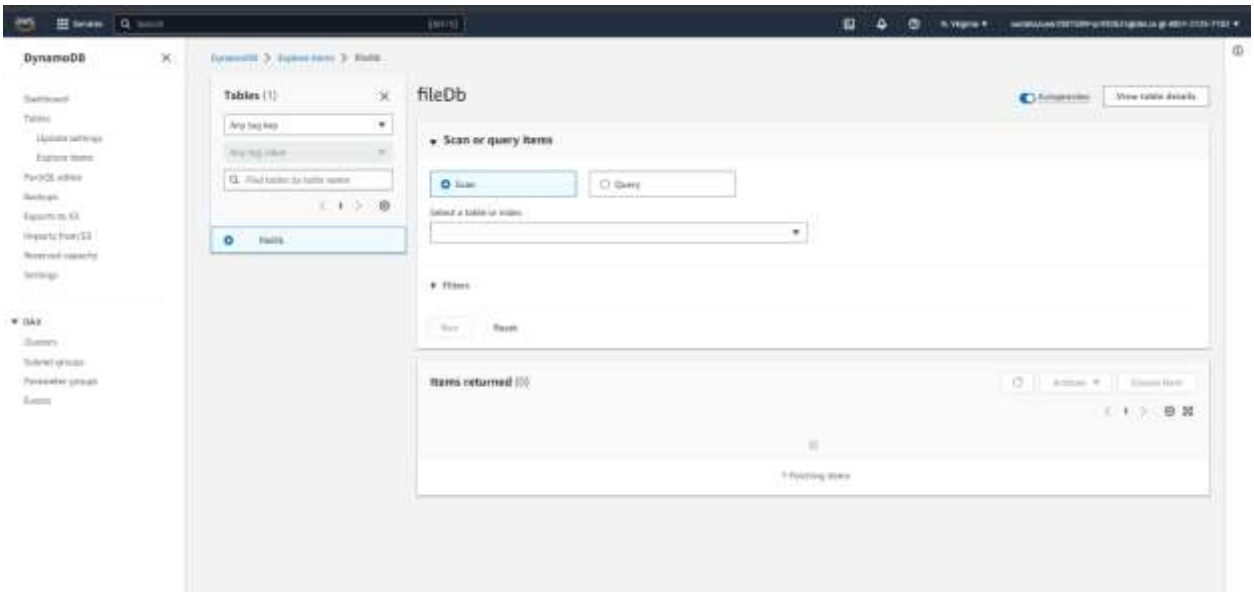
DynamoDB
Tables (1)
fileDb
Scan or query items
Scan
Query
Filters
Items returned (0)

Fig 29

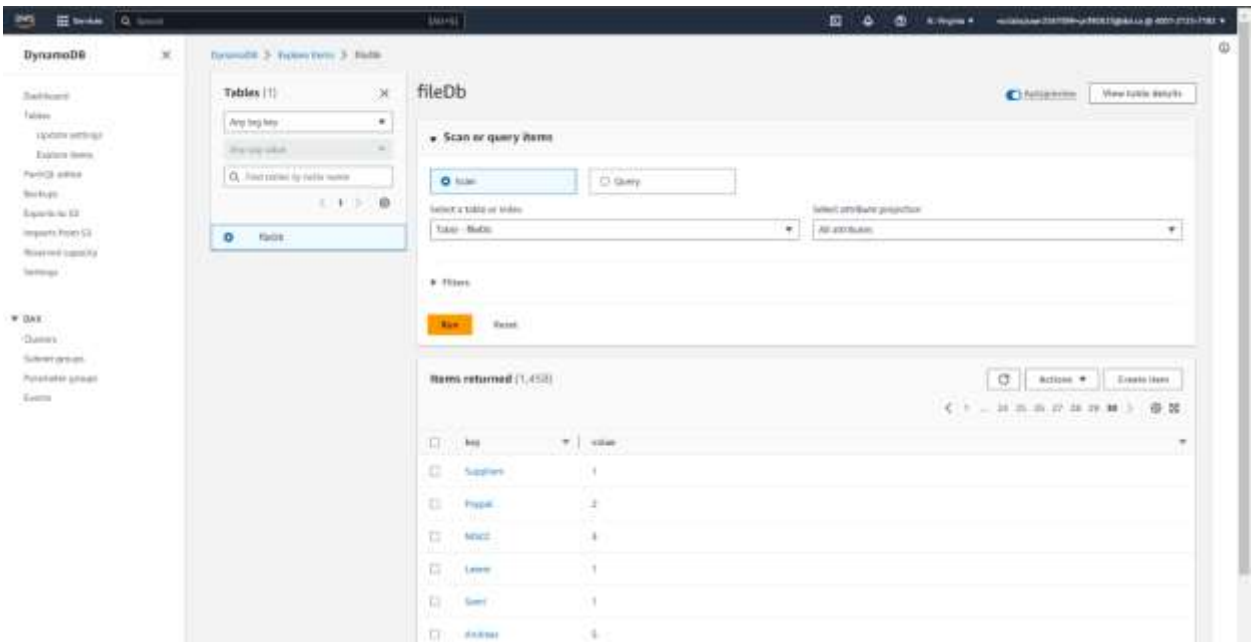

Fig 30

Step 12: We will download any random file and search in Query to check if the values have been stored or not. I have downloaded 20ne.txt and searched four items and it matched which verifies that the items have been successfully stored in the table as shown in Fig 31, 32, 33, 34, 35.

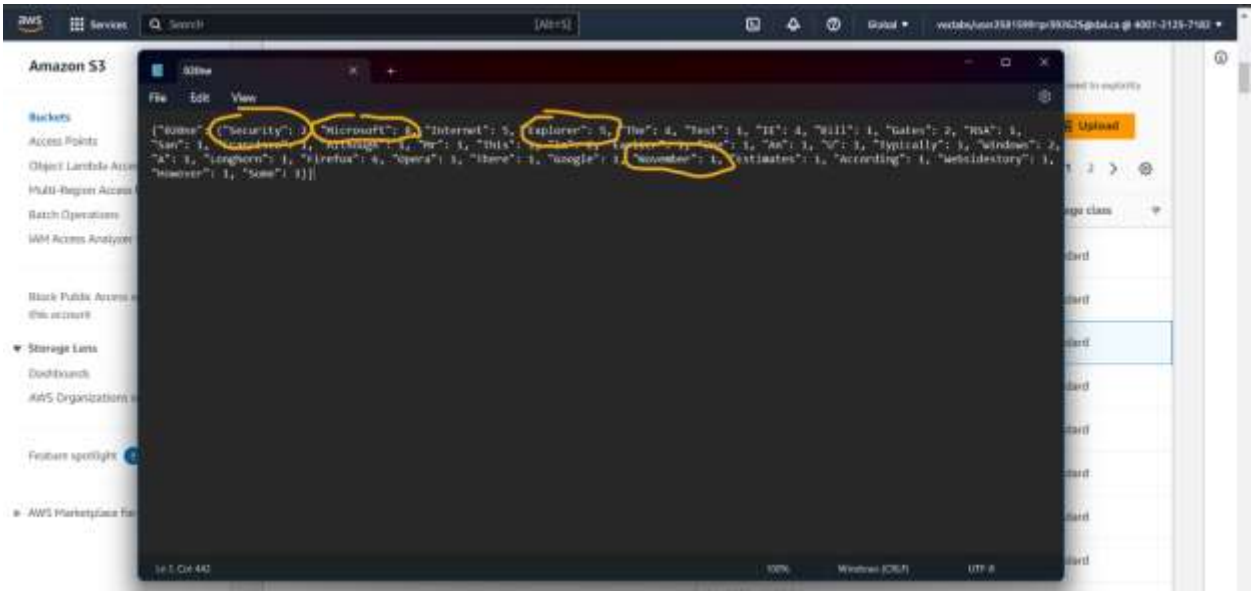

Fig 31

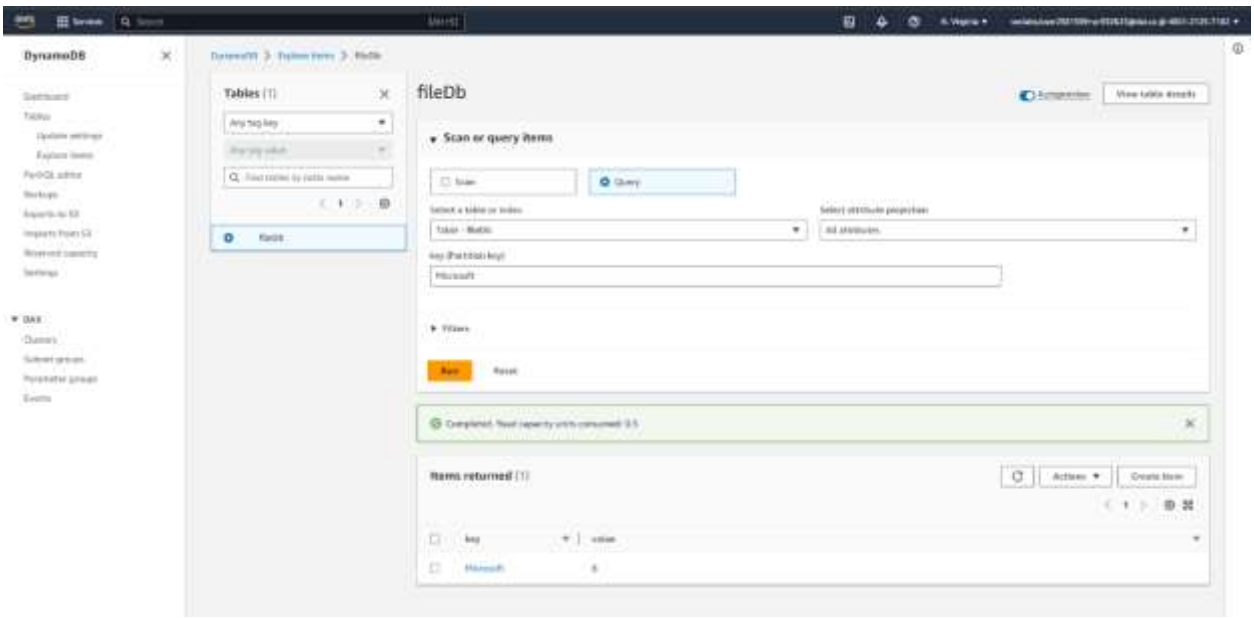
DynamoDB
Tables (1)
fileDb
View table details
Scan or query items
Scan
Query
Filters
Run
Reset
Items returned (1)
Actions
Create item
key
value

Fig 32

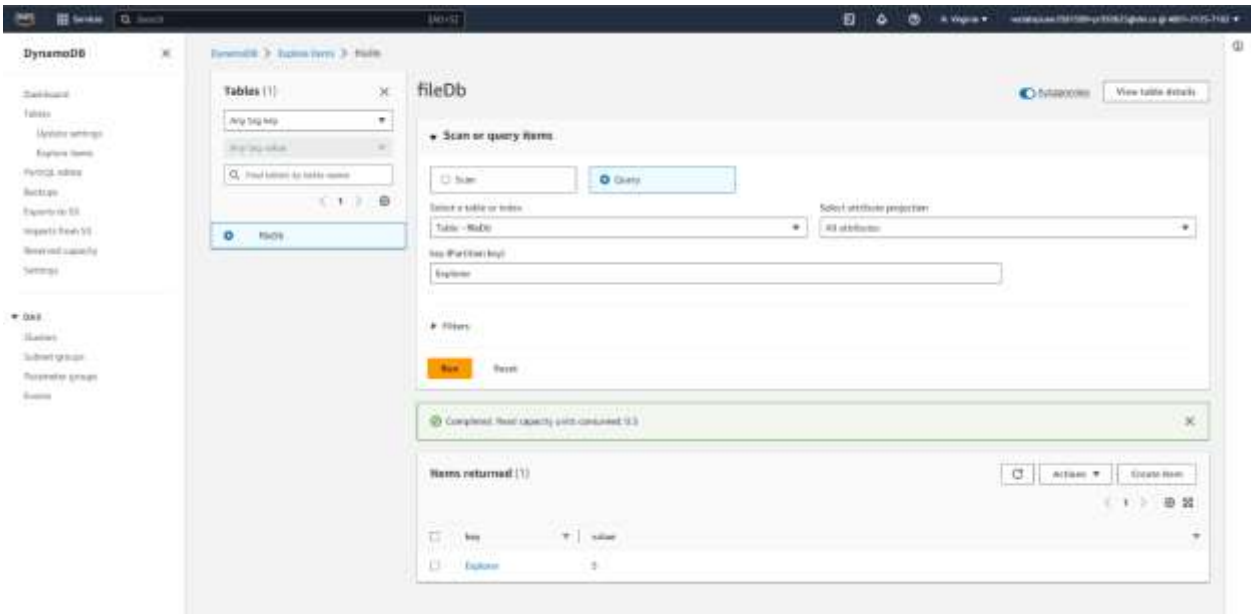
DynamoDB
Tables (1)
fileDb
View table details
Scan or query items
Scan
Query
Filters
Run
Reset
Items returned (1)
Actions
Create item
key
value

Fig 33

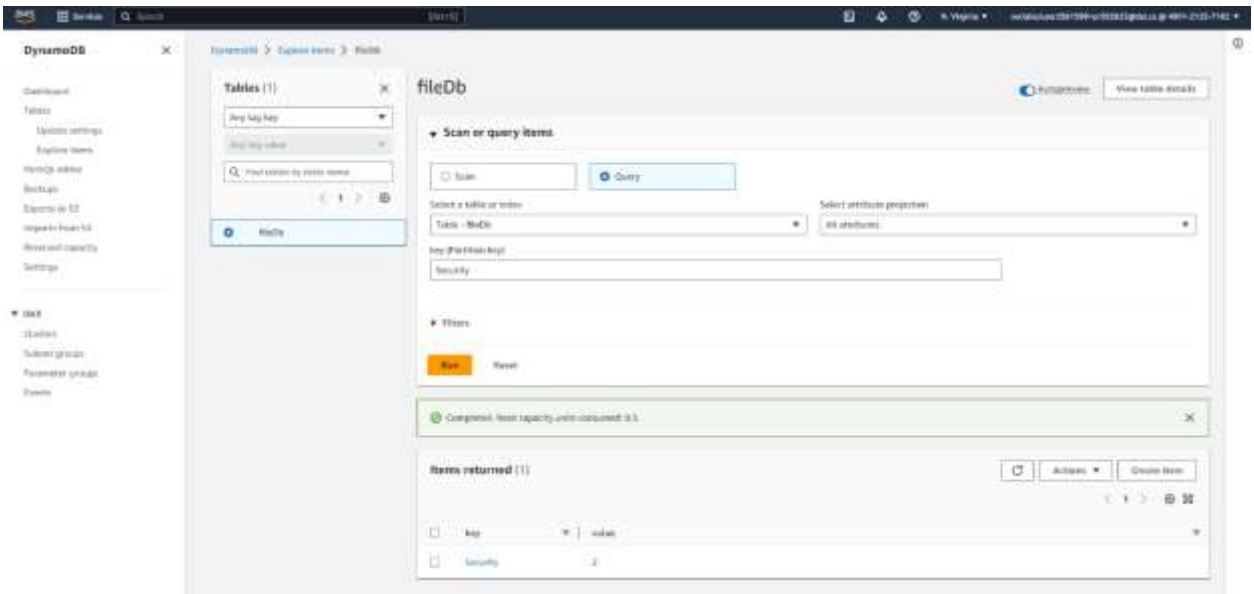
DynamoDB
Tables (1)
fileDb
View table details
Scan or query items
Scan
Query
Filters
Run
Reset
Items returned (1)
Actions
Create item
key
value

Fig 34

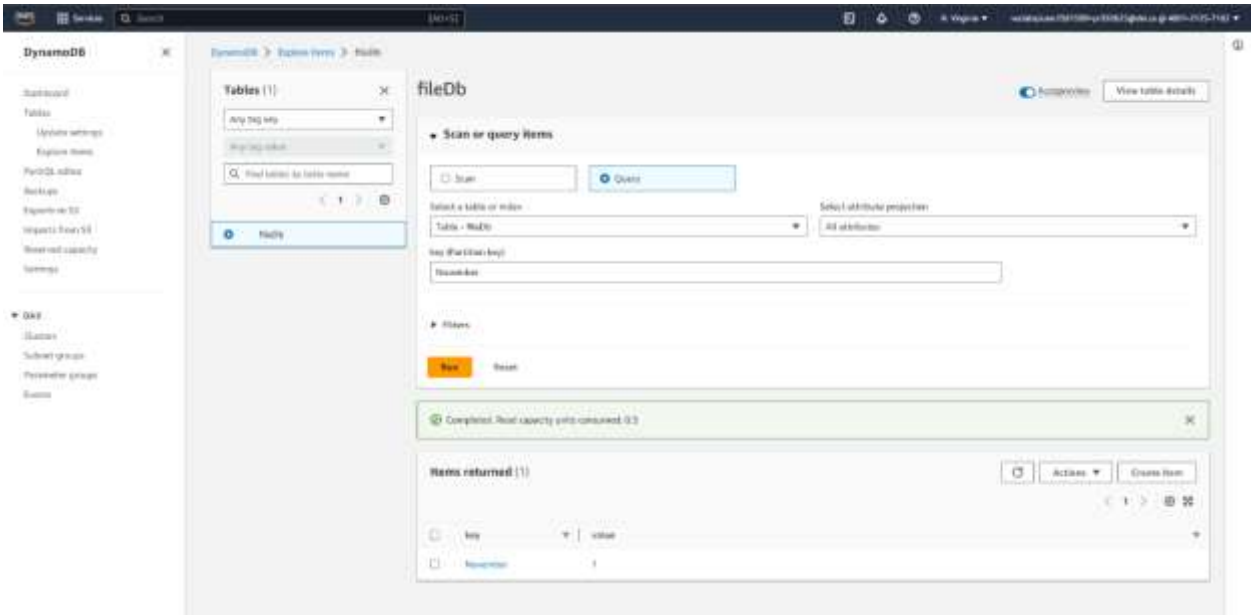
DynamoDB
Tables (1)
fileDb
View table details
Scan or query items
Scan
Query
Filters
Run
Reset
Items returned (1)
Actions
Create item
key
value

Fig 35