Serverless Labs

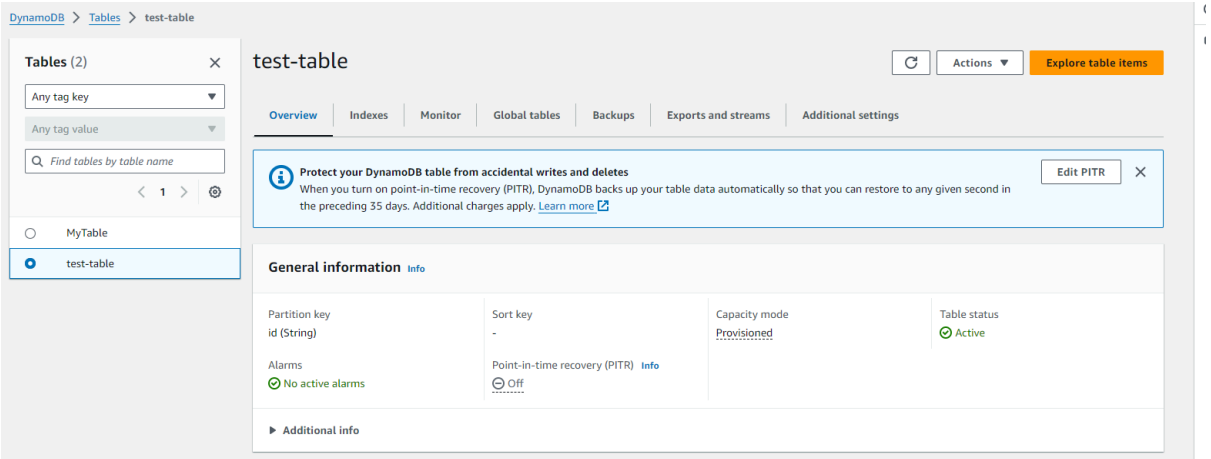
Creating a Serverless API

Define API:

I will be designing a simple RESTful API for a note application that allows us to

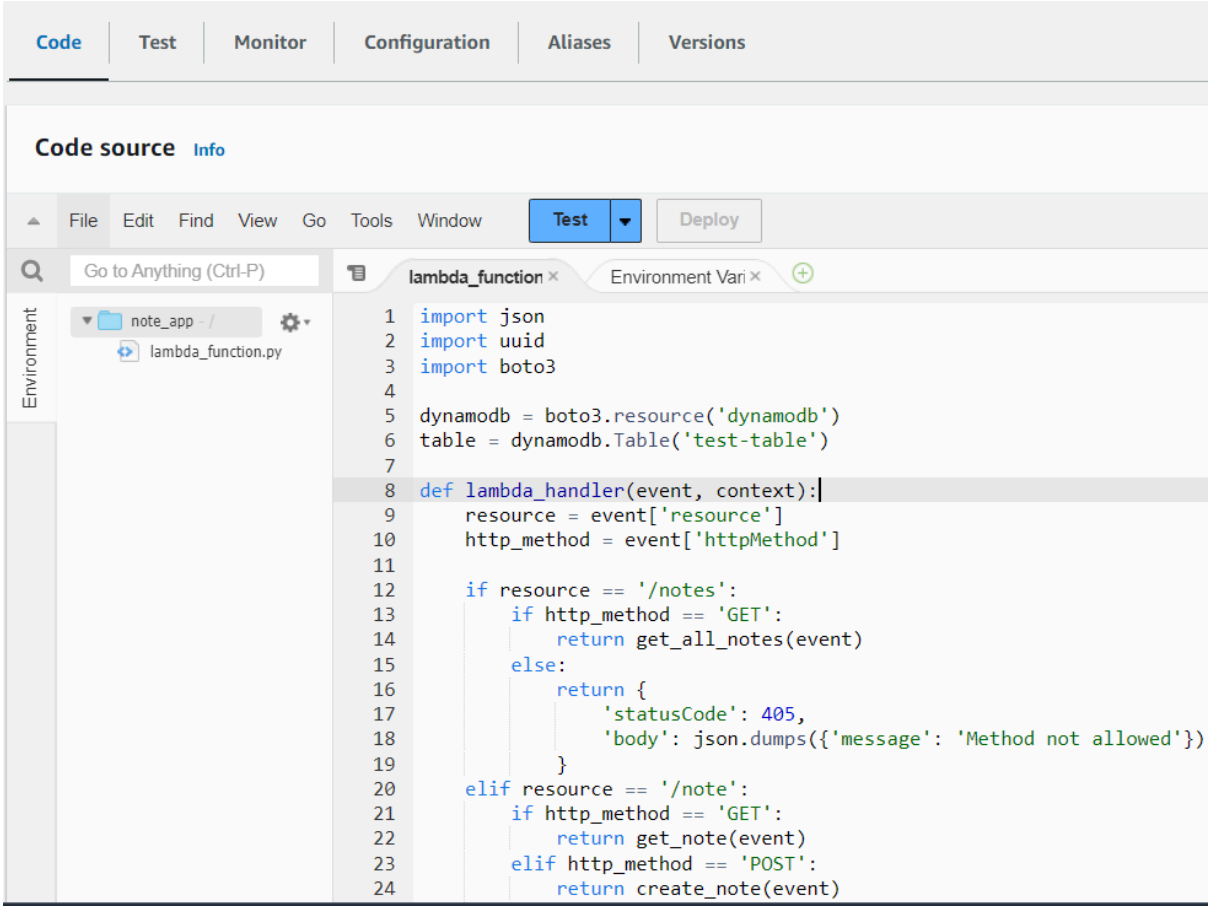
- ➔ List all the notes(GET)
- ➔ List a note(GET)
- ➔ Create a note(POST)
- ➔ Update a note(PUT)
- ➔ Delete a note(DELETE)

Step 1: Create a table in DynamoDB to store all the notes. I created a table named "test-table" with partition key "id".



Step 2: Create a lambda function

I created a lambda function named "note_app":



The above lambda function includes following code:

import json

import uuid

import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('test-table')

def lambda_handler(event, context):

resource = event['resource']

http_method = event['httpMethod']

if resource == '/notes':

```
    if http_method == 'GET':
        return get_all_notes(event)
    else:
        return {
            'statusCode': 405,
            'body': json.dumps({'message': 'Method not allowed'})
        }
elif resource == '/note':
    if http_method == 'GET':
        return get_note(event)
    elif http_method == 'POST':
        return create_note(event)
    elif http_method == 'PUT':
        return update_note(event)
    elif http_method == 'DELETE':
        return delete_note(event)
    else:
        return {
            'statusCode': 405,
            'body': json.dumps({'message': 'Method not allowed'})
        }
else:
    return {
        'statusCode': 404,
        'body': json.dumps({'message': 'Resource not found'})
    }
```

```
def get_all_notes(event):
    try:
        response = table.scan()
        return {
            'statusCode': 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type",
                "Access-Control-Allow-Methods": "GET"
            },
            'body': json.dumps(response['Items']),
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({'message': str(e)})
        }

def get_note(event):
    try:
        note_id = event['queryStringParameters']['id']
        response = table.get_item(Key={'id': note_id})
        if 'Item' in response:
            return {
                'statusCode': 200,
                "headers": {
                    "Access-Control-Allow-Origin": "*",
```

```
                "Access-Control-Allow-Headers": "Content-Type",
                "Access-Control-Allow-Methods": "GET"
            },
            'body': json.dumps(response['Item'])
        }
    else:
        return {
            'statusCode': 404,
            'body': json.dumps({'message': 'Note not found'})
        }
except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({'message': str(e)})
    }

def create_note(event):
    try:
        note_id = str(uuid.uuid4())
        note_data = json.loads(event['body'])
        note_data['id'] = note_id
        table.put_item(Item=note_data)
        return {
            'statusCode': 201,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type",
```

```
            "Access-Control-Allow-Methods": "POST"
        },
        'body': json.dumps({'message': 'Note created successfully'})
    }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({'message': str(e)})
        }

def update_note(event):
    try:
        note_id = event['queryStringParameters']['id']
        update_data = json.loads(event['body'])
        update_expression = 'SET ' + ', '.join([f'#{k} = :{k}' for k in update_data.keys()])
        expression_attribute_names = {f'#{k}': k for k in update_data.keys()}
        expression_attribute_values = {f':{k}': v for k, v in update_data.items()}
        table.update_item(
            Key={'id': note_id},
            UpdateExpression=update_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues=expression_attribute_values
        )
        return {
            'statusCode': 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
```

```
            "Access-Control-Allow-Headers": "Content-Type",
            "Access-Control-Allow-Methods": "PUT"
        },
        'body': json.dumps({'message': 'Note updated successfully'})
    }
  except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({'message': str(e)})
    }


def delete_note(event):
  try:
    note_id = event['queryStringParameters']['id']
    table.delete_item(Key={'id': note_id})
    return {
        'statusCode': 204,
        'body': json.dumps({'message': 'Note deleted successfully'})
    }
  except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps({'message': str(e)})
    }
```

Step 3: Navigate to API Gateway to create API. Select lambda function (note_app) for all API requests. Here is a sample of the GET request. Similarly, other methods are created.

Method type

GET

Integration type

Lambda function
Integrate your API with a Lambda function.

HTTP
Integrate with an existing HTTP endpoint.

Mock
Generate a response based on API Gateway mappings and transformations.

AWS service
Integrate with an AWS Service.

VPC link
Integrate with a resource that isn't accessible over the public internet.

Lambda proxy integration
Send the request to your Lambda function as a structured event.

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1

arn:aws:lambda:us-east-1:695898526652:function:note

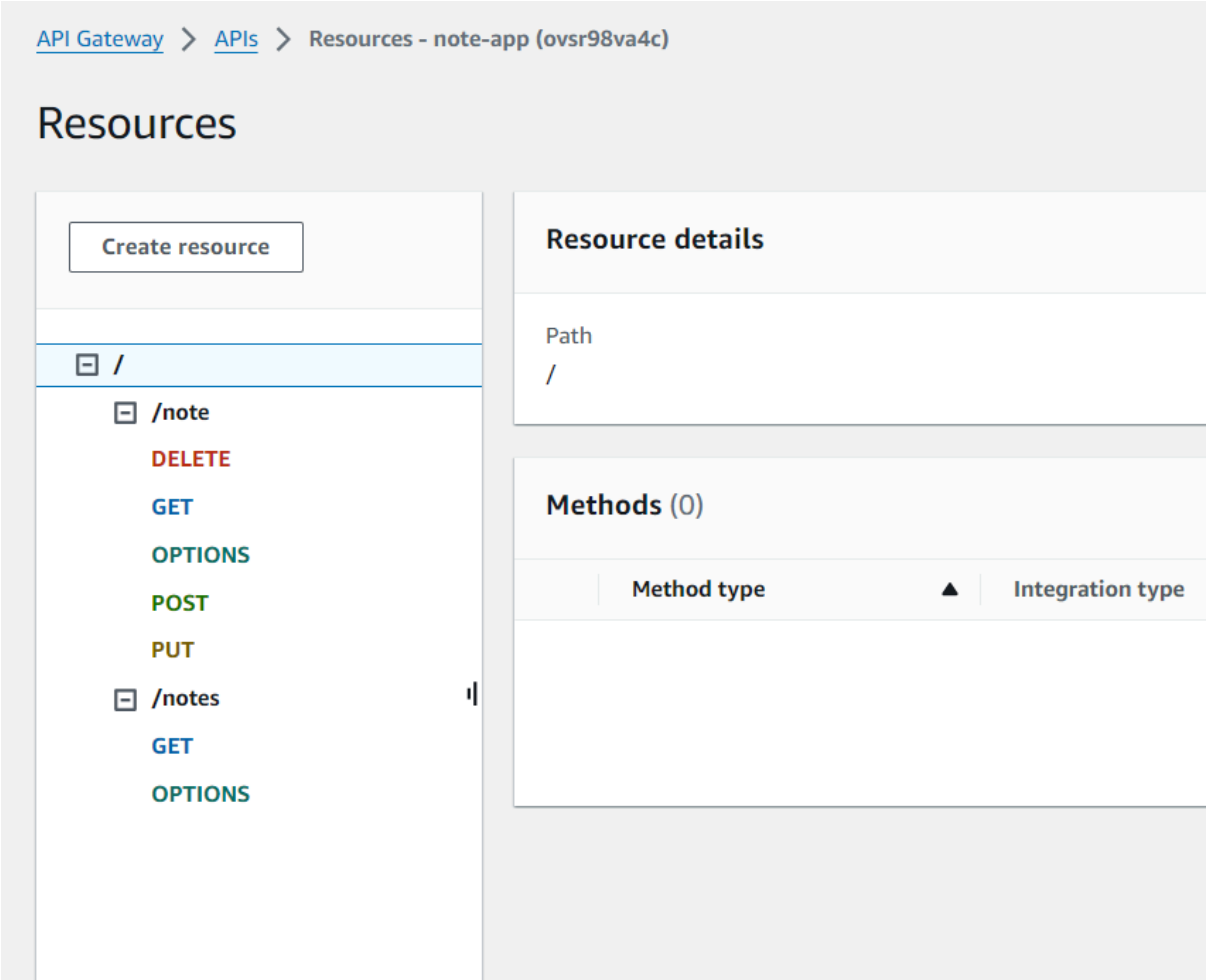
Grant API Gateway permission to invoke your Lambda function. To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.

Default timeout
The default timeout is 29 seconds.

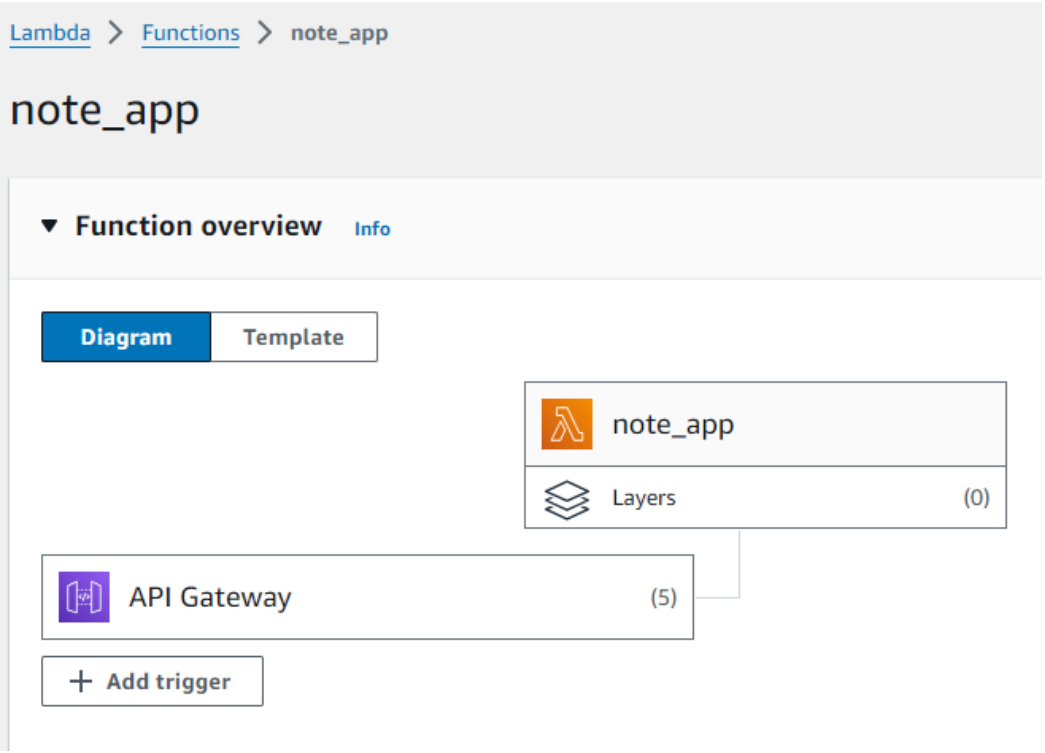
Cancel

Create method

After creating all the methods, it looks as below:



Overview of lambda function architecture:



Step 4: Lets now test all the APIs in postman.

Post note

note app / **Post note**

Save

POST https://ovsr98va4c.execute-api.us-east-1.amazonaws.com/test/note Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "title": "Grocery List",
3     "content": "Need to pick up eggs, milk, bread, apples, and chicken breasts for the week. Running low on essentials and need to restock
        to ensure meals are covered."
4 }
```

Body Cookies Headers (10) Test Results (1/1) Status: 201 Created Time: 325 ms Size: 491 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "Note created successfully"
3 }
```

Get note by id

note app / **Get note**

Save

GET https://ovsr98va4c.execute-api.us-east-1.amazonaws.com/test/note?id=3ebc94a8-f9a5-4353-adea-5a4915f269f5 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
id	3ebc94a8-f9a5-4353-adea-5a4915f269f5		
Key	Value	Description	

Body Cookies Headers (10) Test Results Status: 200 OK Time: 335 ms Size: 683 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2     "content": "Need to pick up eggs, milk, bread, apples, and chicken breasts for the week. Running low on essentials and need to restock
        to ensure meals are covered.",
3     "id": "3ebc94a8-f9a5-4353-adea-5a4915f269f5",
4     "title": "Grocery List"
5 }
```

Update note by id

note app / **Update note**

PUT https://ovsr98va4c.execute-api.us-east-1.amazonaws.com/test/note?id=3ebc94a8-f9a5-4353-adea-5a4915f269f5

Params • Authorization Headers (8) Body • Pre-request Script Tests • Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "title": "Grocery List",
3     "content": "Need to pick up eggs, milk, bread, and apples"
4 }
```

Body Cookies Headers (10) Test Results (1/1) Status: 200 OK Time: 73

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "Note updated successfully"
3 }
```

Delete note

note app / **Delete note**

DELETE https://ovsr98va4c.execute-api.us-east-1.amazonaws.com/test/note?id=3ebc94a8-f9a5-4353-adea-5a4915f269f5

Params • Authorization Headers (6) Body Pre-request Script Tests • Settings

Query Params

	Key	Value	Description
☑	id	3ebc94a8-f9a5-4353-adea-5a4915f269f5	
	Key	Value	Description

Body Cookies Headers (7) Test Results (1/1) Status: 204 No Content Time: 2.19 s Size: 341 E

Pretty Raw Preview Visualize JSON

```
1
```

Get all notes

note app / **Get notes**

GET https://ovsr98va4c.execute-api.us-east-1.amazonaws.com/test/notes

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (10) Test Results (1/1)

Pretty Raw Preview Visualize JSON


```
[
    {
        "content": "Exercise, read, save money, learn new skill.",
        "id": "8dce263c-4e54-48cf-b833-61410af2f080",
        "title": "Personal Goals"
    },
    {
        "content": "Review last week, assign tasks, address concerns, set goals."
        "id": "c080ee8e-fbcc-462d-a6ab-6efc32099ffd",
        "title": "Meeting Agenda"
    },
    {
        "content": "Need eggs, milk, bread, apples, and chicken breasts.",
        "id": "d6437487-9fe7-42a9-a81c-1b05933e859f",
        "title": "Grocery List"
    }
]
```


All the methods are functioning well.

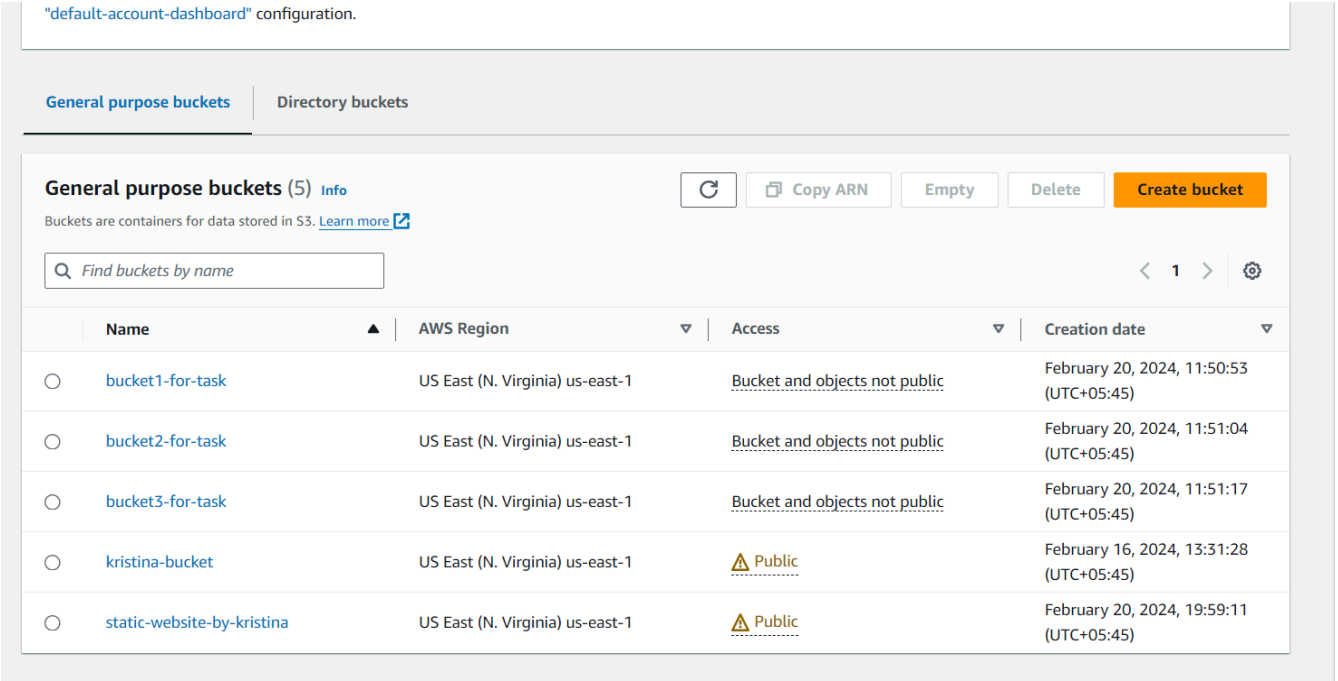
Building a Serverless Web Application

Create a serverless web application using AWS Lambda, API Gateway, S3, and DynamoDB.

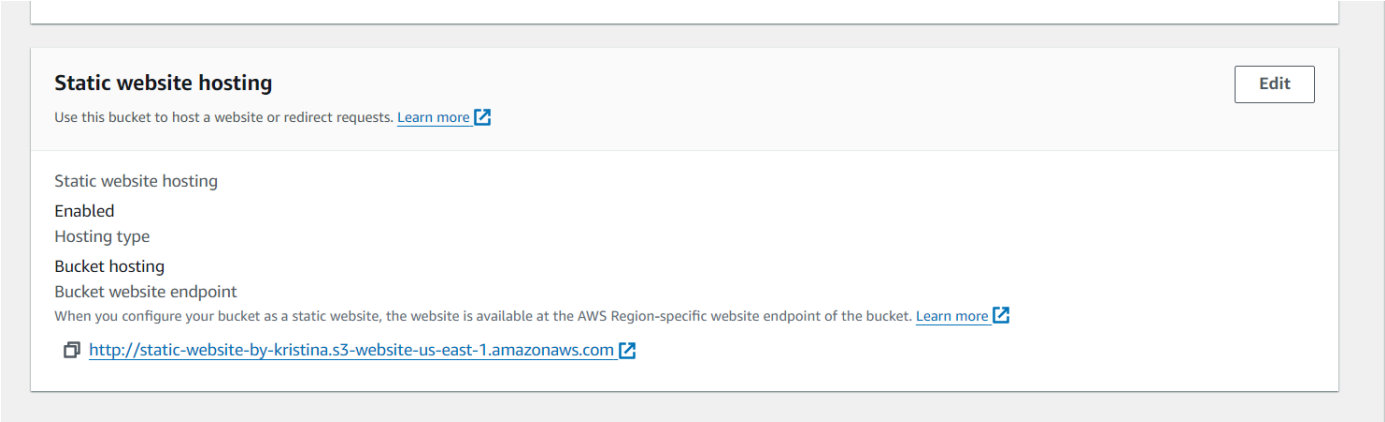
I have already created an API Gateway that triggers Lambda which interacts with the DynamoDB table as represented above.

Now, I will be hosting a static web application on S3 bucket.

Step 1: Created a bucket named “static-website-by-kristina” .



Step 2: Enable static website hosting which is under the “Properties” tab inside the bucket.



Step 3: Made the bucket publicly accessible by disabling block public access and also added a statement that allows public access to all the objects inside the “static-website-by-kristina” bucket.

Permissions overview

Access

⚠ Public

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these set objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Block *all* public access

⚠ Off

▶ **Individual Block Public Access settings for this bucket**

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owı

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::static-website-by-kristina/*"
    }
  ]
}
```

Step 4: Uploaded a folder named “static website” which has two files “index.html” that includes the home page and “error.html” that includes the error page.

Objects | Properties | Permissions | Metrics | Management | Access Poin

Objects (1) Info

Copy S3 URI | Copy URL | Download | Open

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of a
grant them permissions. Learn more

Find objects by prefix

Name	Type	Last modified
static_website/	Folder	-

Amazon S3 > Buckets > static-website-by-kristina > static_website/

static_website/

Objects | Properties

Objects (2) Info

Copy S3 URI | Copy URL | Download

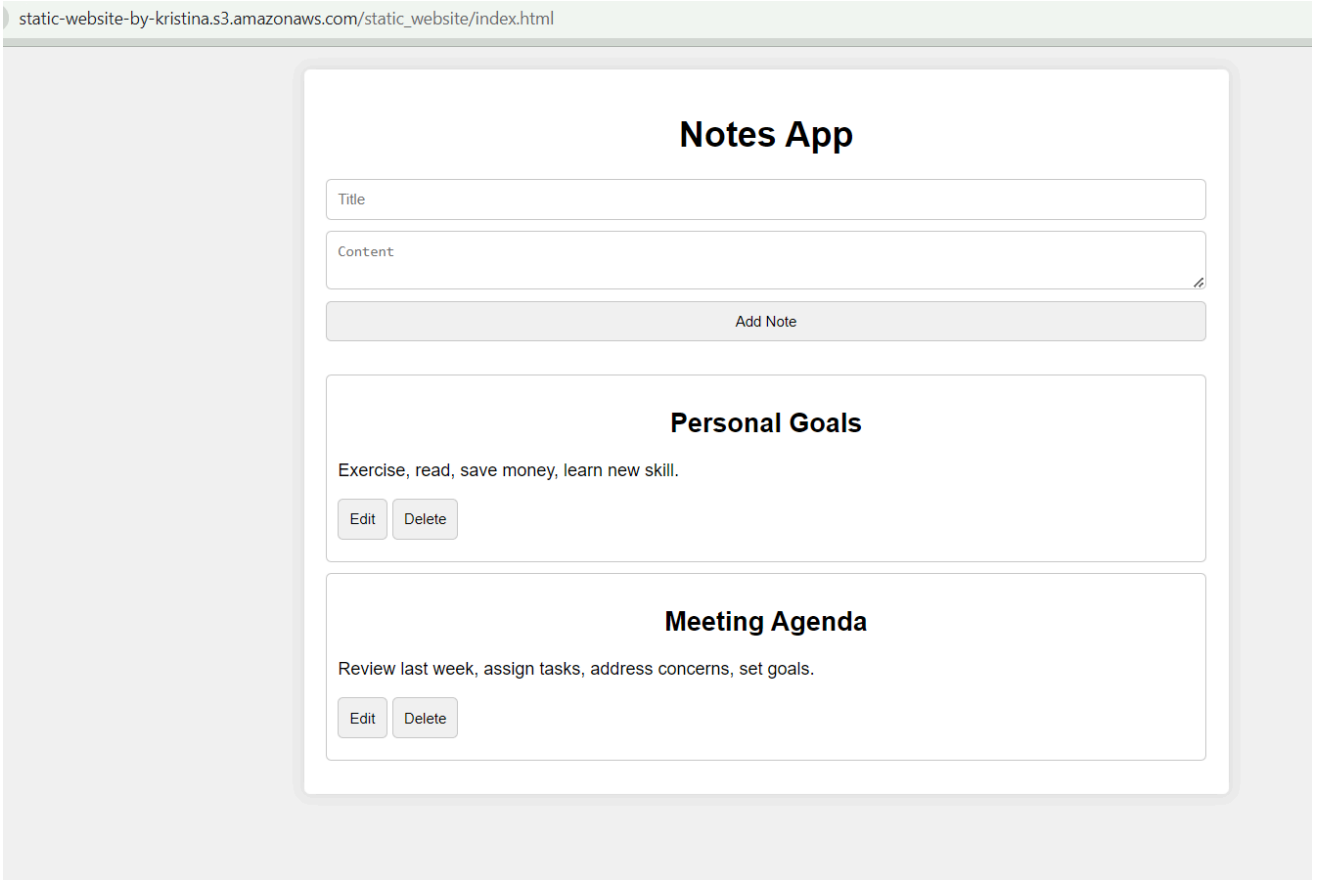
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 invent
grant them permissions. Learn more

Find objects by prefix

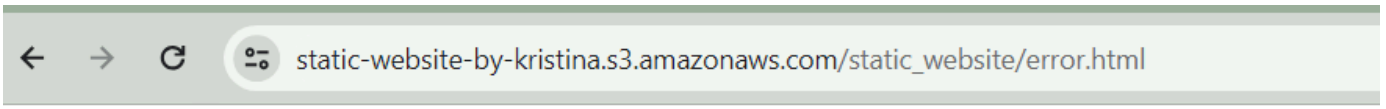
Name	Type	L
error.html	html	F 0
index.html	html	F 1

Now, we can access the static website by using the object url of “index.html”.

Step 5: Copy object url of “index.html” and paste it in new tab. You can see a note application.

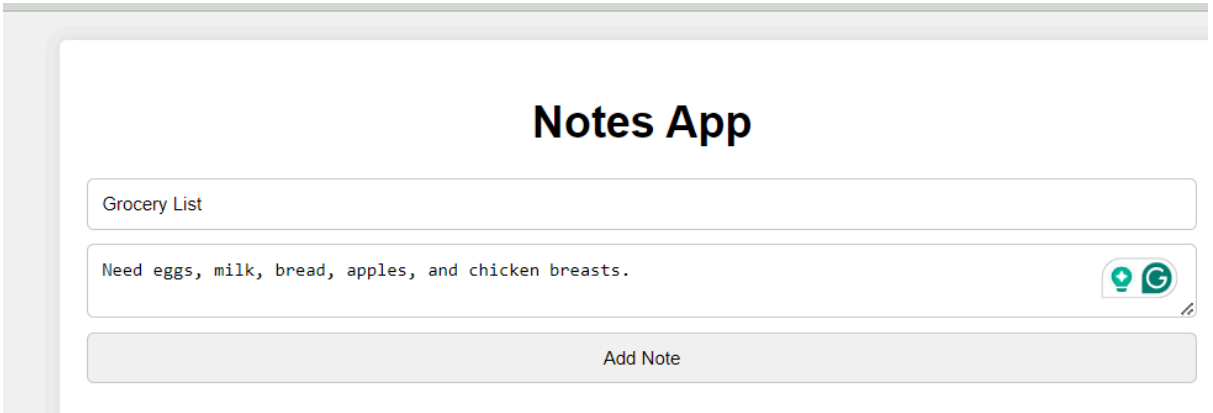


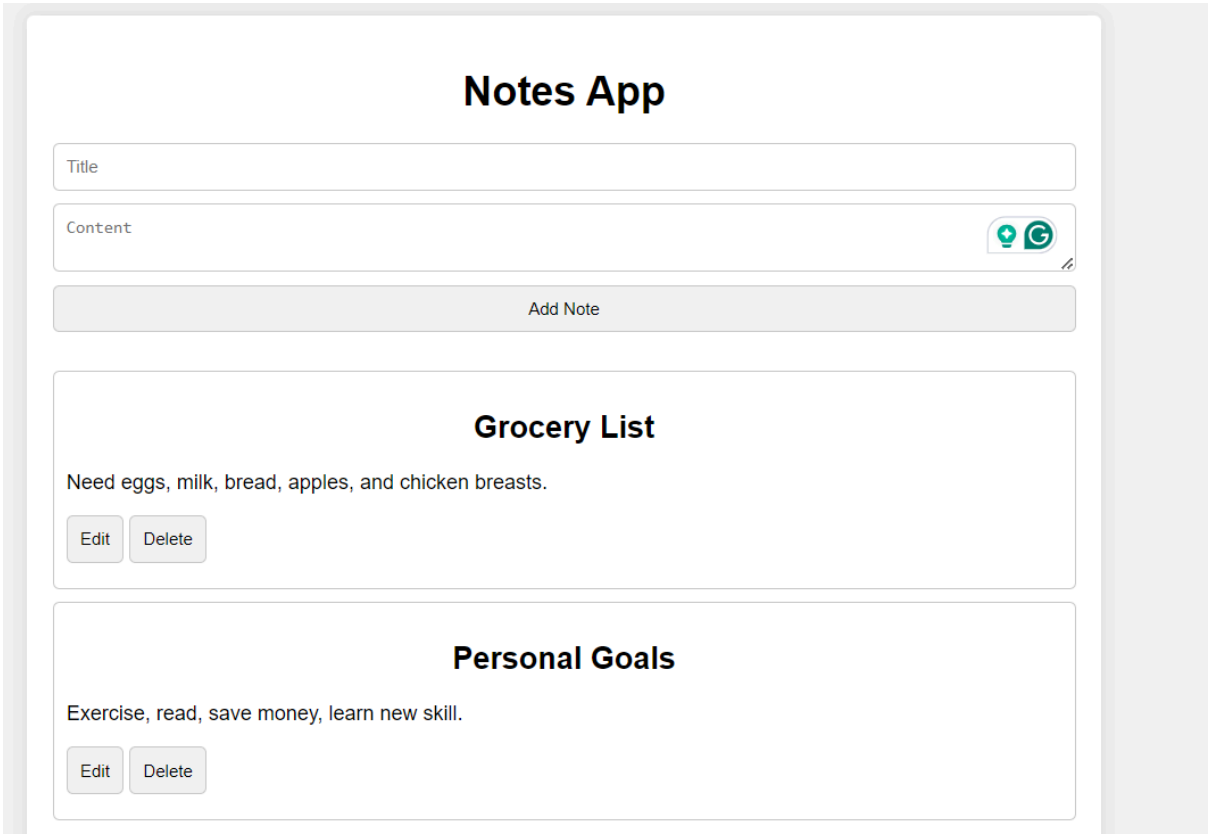
If there is any error, then you will be redirected to the error page.



Error

New note has been added as follow:





Serverless Data Processing Pipeline

Building a serverless pipeline for processing data (e.g., log processing or ETL jobs).

I will create 3 buckets. When I upload a text file on bucket 1, it will automatically replicate on bucket 2 and the text inside the file will be converted to uppercase and automatically uploaded to bucket 3.

Step 1: Create three buckets.

General purpose buckets | Directory buckets

General purpose buckets (5) Info

Buckets are containers for data stored in S3. Learn more

Copy ARN | Empty | Delete | Create bucket

Find buckets by name

1

Name	AWS Region	Access	Creation date
bucket1-for-task	US East (N. Virginia) us-east-1	Bucket and objects not public	February 20, 2024, 11:50:53 (UTC+05:45)
bucket2-for-task	US East (N. Virginia) us-east-1	Bucket and objects not public	February 20, 2024, 11:51:04 (UTC+05:45)
bucket3-for-task	US East (N. Virginia) us-east-1	Bucket and objects not public	February 20, 2024, 11:51:17 (UTC+05:45)

Step 2: Create a lambda function named "replicate" that performs specified functionality and deploy it.

Code source Info

File Edit Find View Go Tools Window | Test | Deploy

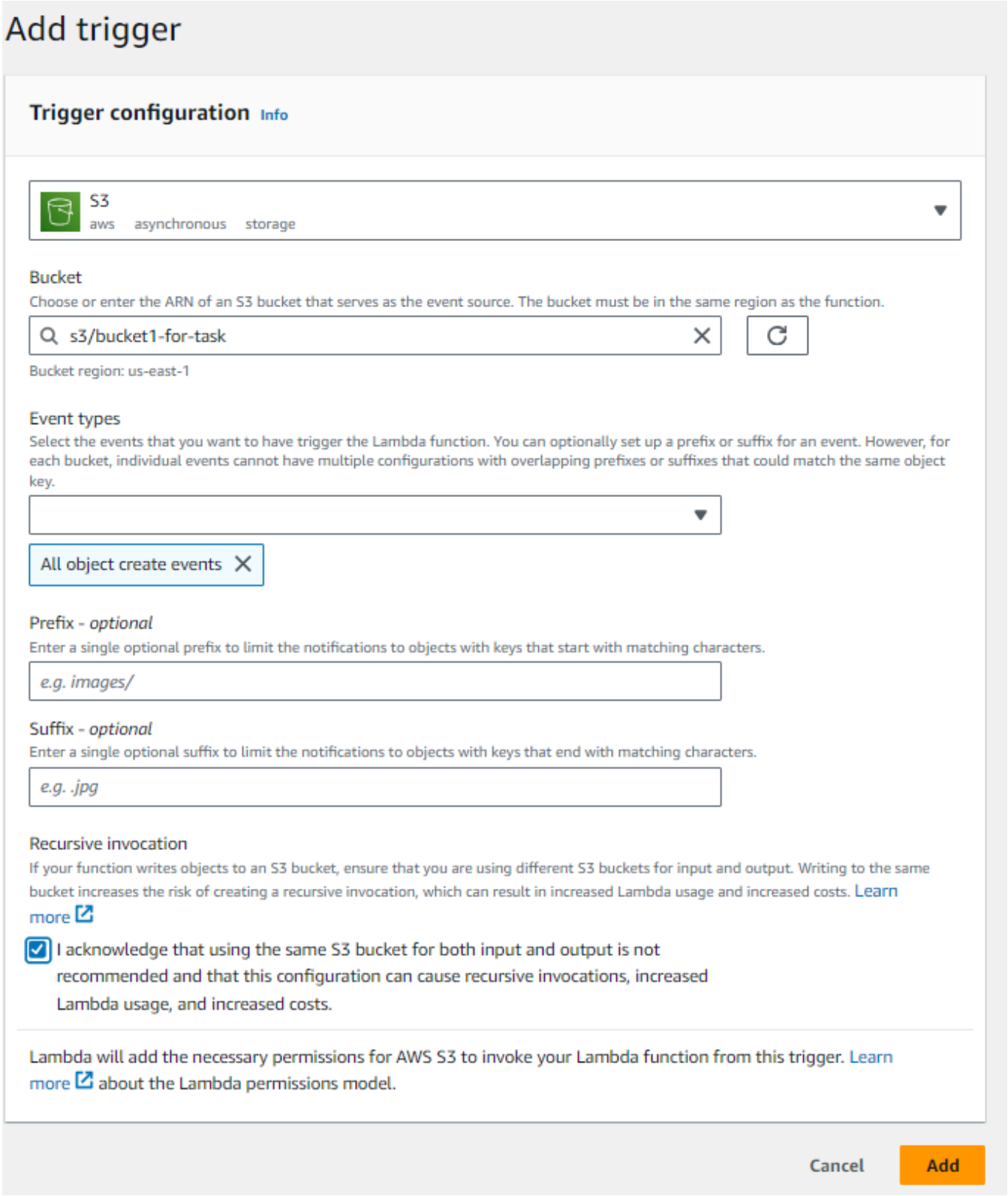
Go to Anything (Ctrl-P)

replicate
lambda_function.py

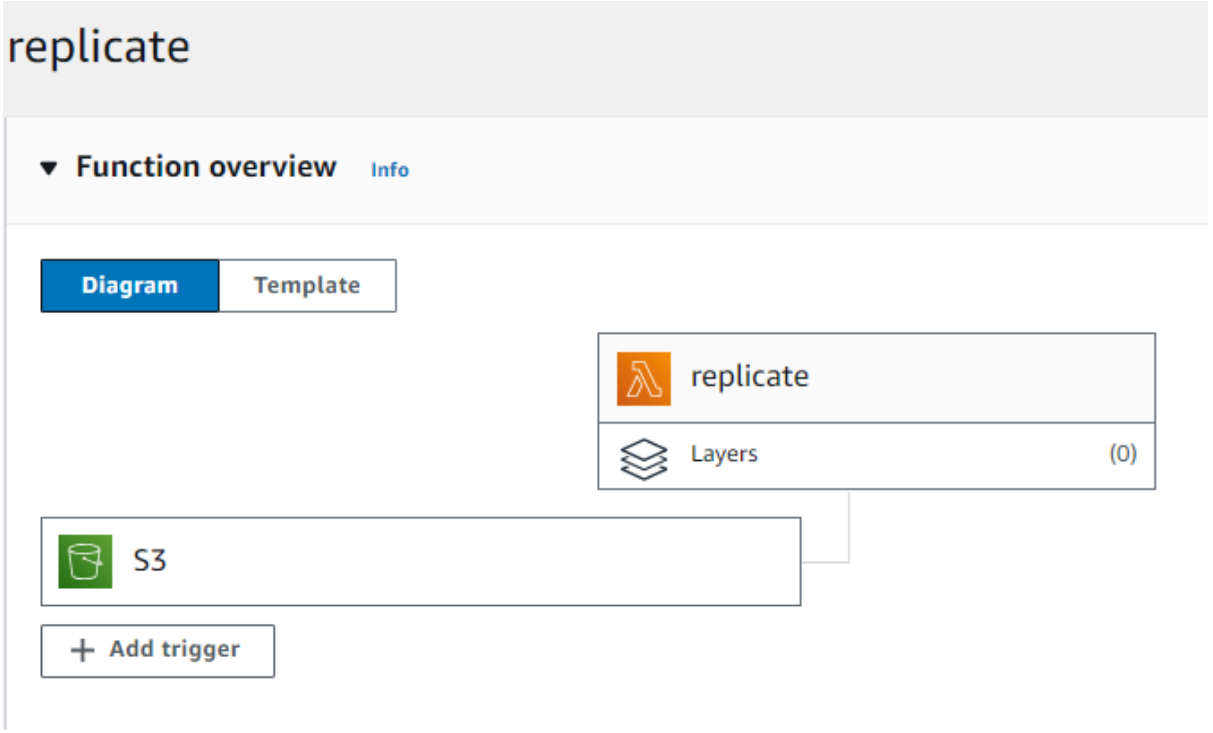
lambda_function | Environment Vari

```python
import boto3
import json

s3 = boto3.client('s3')

def lambda_handler(event, context):
    source_bucket = event['Records'][0]['s3']['bucket']['name']
    source_key = event['Records'][0]['s3']['object']['key']

    destination_bucket = 'bucket2-for-task'
    destination_key = source_key
    s3.copy_object(CopySource={'Bucket': source_bucket, 'Key': source_key},
                   Bucket=destination_bucket, Key=destination_key)

    response = s3.get_object(Bucket=destination_bucket, Key=destination_key)
    content = response['Body'].read().decode('utf-8').upper()
    destination_bucket_c = 'bucket3-for-task'
    transformed_key = destination_key

    print(content)
    s3.put_object(Bucket=destination_bucket_c, Key=transformed_key, Body=content)
    s3_a = boto3.resource('s3')
    s3_a.meta.client.download_file(destination_bucket_c, transformed_key, 'C:/Users/kristinan/Downloads/destination_key')

    print(f"File {key} copied to 'boto-s3-2' and updated version saved as '{destination_key}' in 'boto-s3'")

    return {
        'statusCode': 200,
        'body': f"File {key} processed and saved successfully."
    }
```

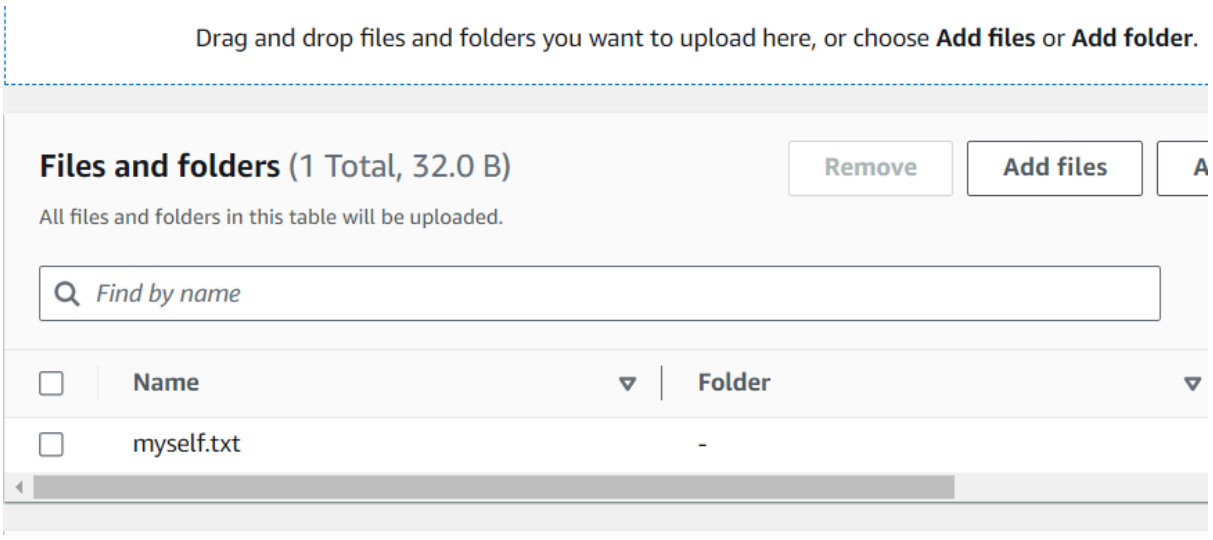
Step 3: Add trigger. The lambda function will be triggered by "bucket1-for-task" when a file is uploaded.



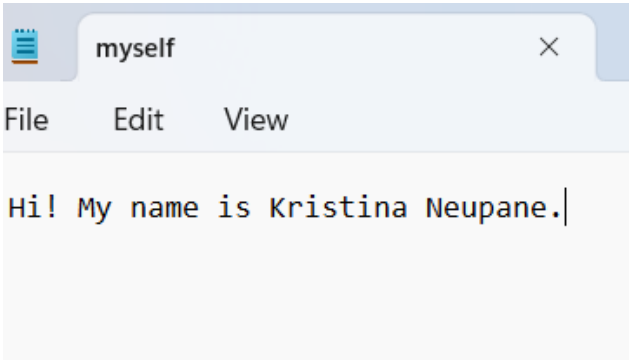
This is the final lambda architecture:



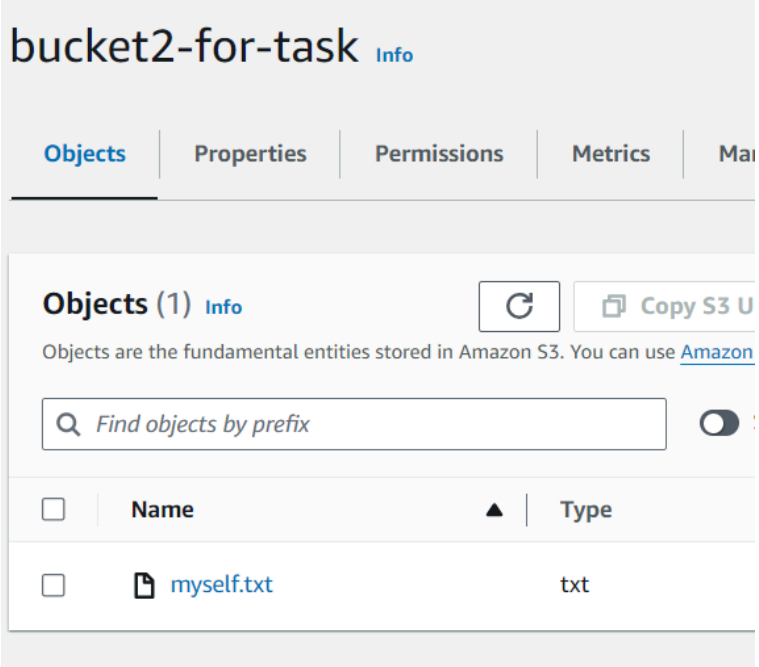
Step 4: Now, we will try uploading a text file in "bucket1-for-task".



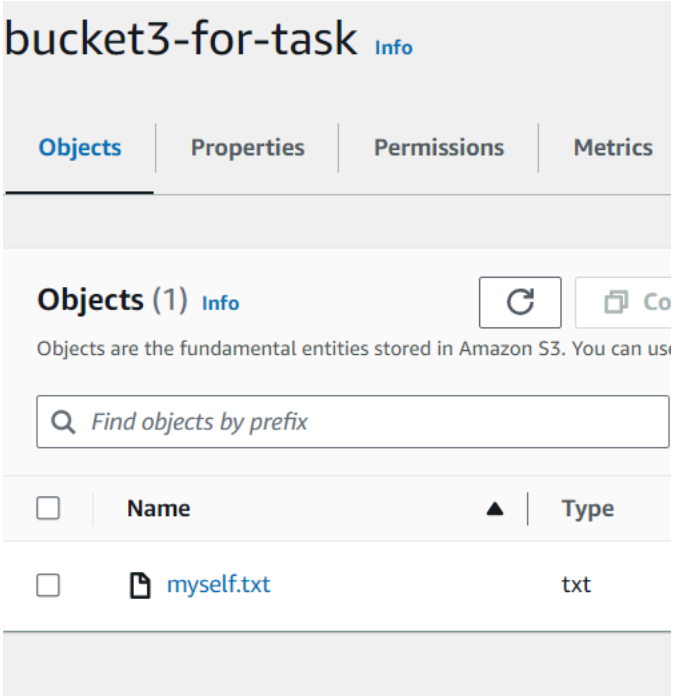
The text file contains following content:



In “bucket2-for-task”, the text file is automatically replicated:



Similarly, in “bucket3-for-task”, a text file is uploaded.



Step 5: Let’s check the logs in “CloudWatch” to check the pipeline performance and verify if the lambda function is executed.

Navigate to the CloudWatch and inside Logs, there is a “Log groups”. There you can see a list of lambda functions. Select the “replicate” lambda function and there will be the log streams. You can select the latest log stream.

CloudWatch > Log groups > /aws/lambda/replicate > 2024/02/23/[$LATEST]94b68238b3a442bfa1f3a1e773e16fca

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

Filter events

Timestamp	Message
	No older events at this moment. *Retry*
2024-02-23T21:00:26.742+05:45	INIT_START Runtime Version: python:3.12.v19 Runtime Version ARN: arn:aws:lambda:us-east-
2024-02-23T21:00:27.249+05:45	START RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce Version: $LATEST
2024-02-23T21:00:27.899+05:45	HI! MY NAME IS KRISTINA NEUPANE.
2024-02-23T21:00:29.419+05:45	[ERROR] FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/kristinan/Down
2024-02-23T21:00:29.440+05:45	END RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce
2024-02-23T21:00:29.440+05:45	REPORT RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce Duration: 2190.58 ms Billed Durat
2024-02-23T21:01:34.592+05:45	START RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce Version: $LATEST
2024-02-23T21:01:35.144+05:45	HI! MY NAME IS KRISTINA NEUPANE.
2024-02-23T21:01:36.318+05:45	[ERROR] FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/kristinan/Down
2024-02-23T21:01:36.339+05:45	END RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce
2024-02-23T21:01:36.339+05:45	REPORT RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce Duration: 1746.94 ms Billed Durat
2024-02-23T21:03:34.591+05:45	START RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce Version: $LATEST
2024-02-23T21:03:35.091+05:45	HI! MY NAME IS KRISTINA NEUPANE.
2024-02-23T21:03:36.318+05:45	[ERROR] FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/kristinan/Down
2024-02-23T21:03:36.319+05:45	END RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce
2024-02-23T21:03:36.319+05:45	REPORT RequestId: c5e01a95-2e0f-4469-82c0-2dd4573658ce Duration: 1728.23 ms Billed Durat

Here, we can see the text is printed in uppercase as we have added print statement in our lambda function.