Serverless Labs

1. Building a Serverless Web Application

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

Approach:

- **Set Up Backend:** Create Lambda functions to handle backend logic. These functions will interact with a DynamoDB table for data storage.
- **API Gateway:** Set up API Gateway to create RESTful endpoints that trigger the Lambda functions.
- **Frontend Hosting:** Host a static website on S3 that interacts with the backend via API Gateway.
- **Integration:** Ensure that the frontend can successfully send requests to the backend and display responses.

Goal: Understand the basics of building and connecting serverless backend services with a static frontend, enabling a fully serverless web application.

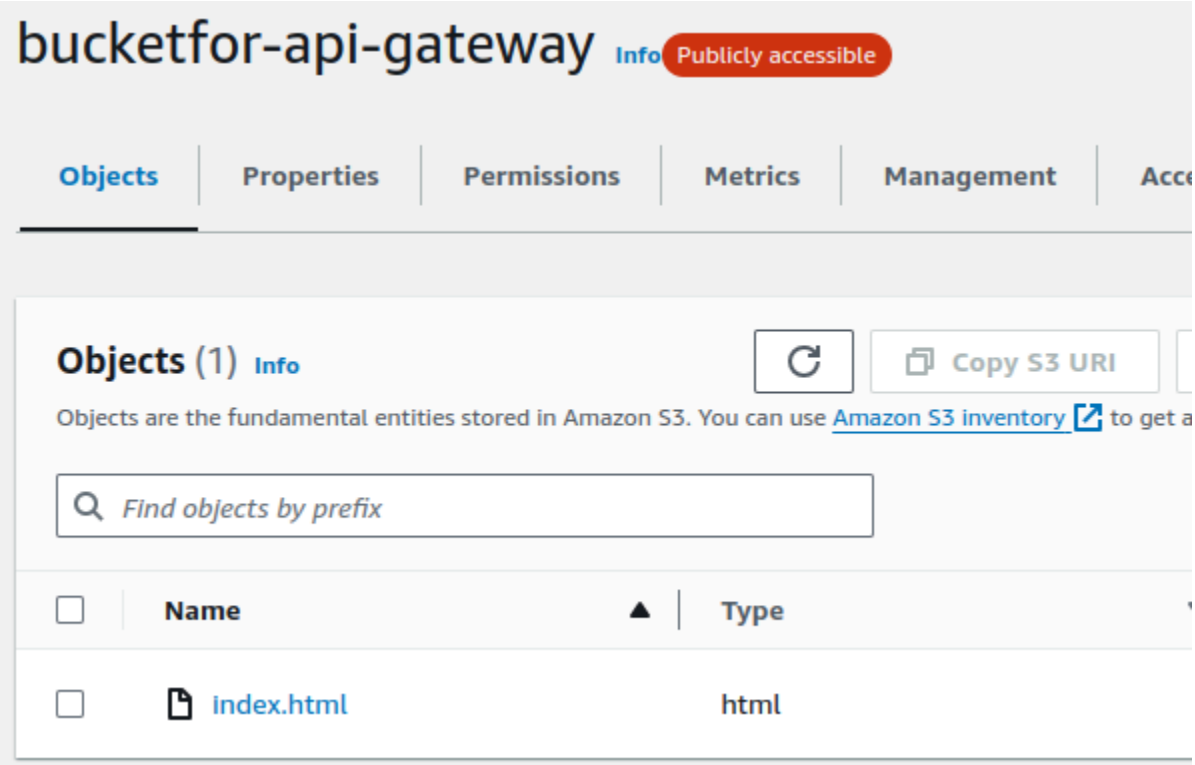
GENERAL STEPS

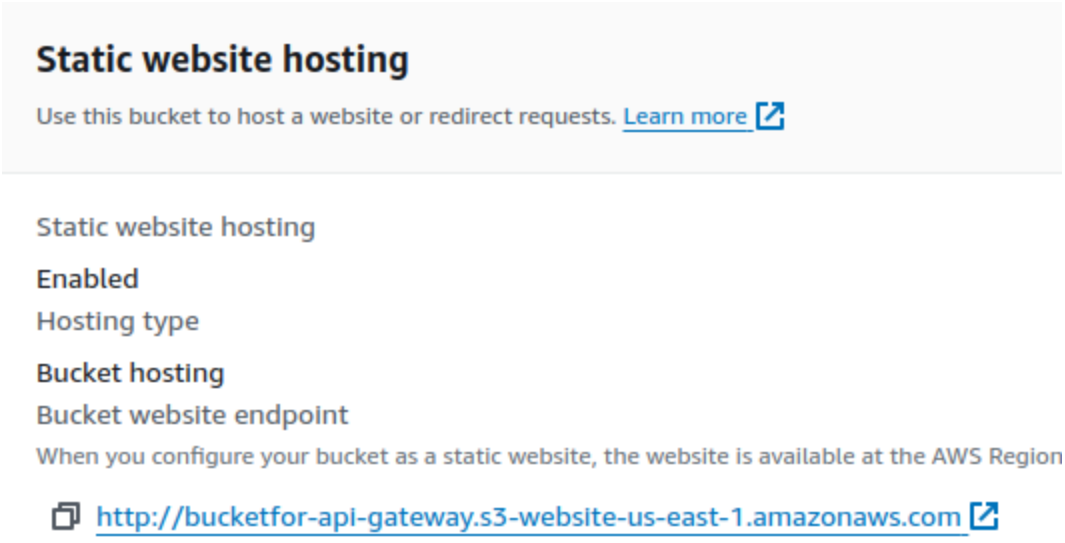
Create a serverless web application using AWS Lambda, API Gateway, S3, DynamoDB

##

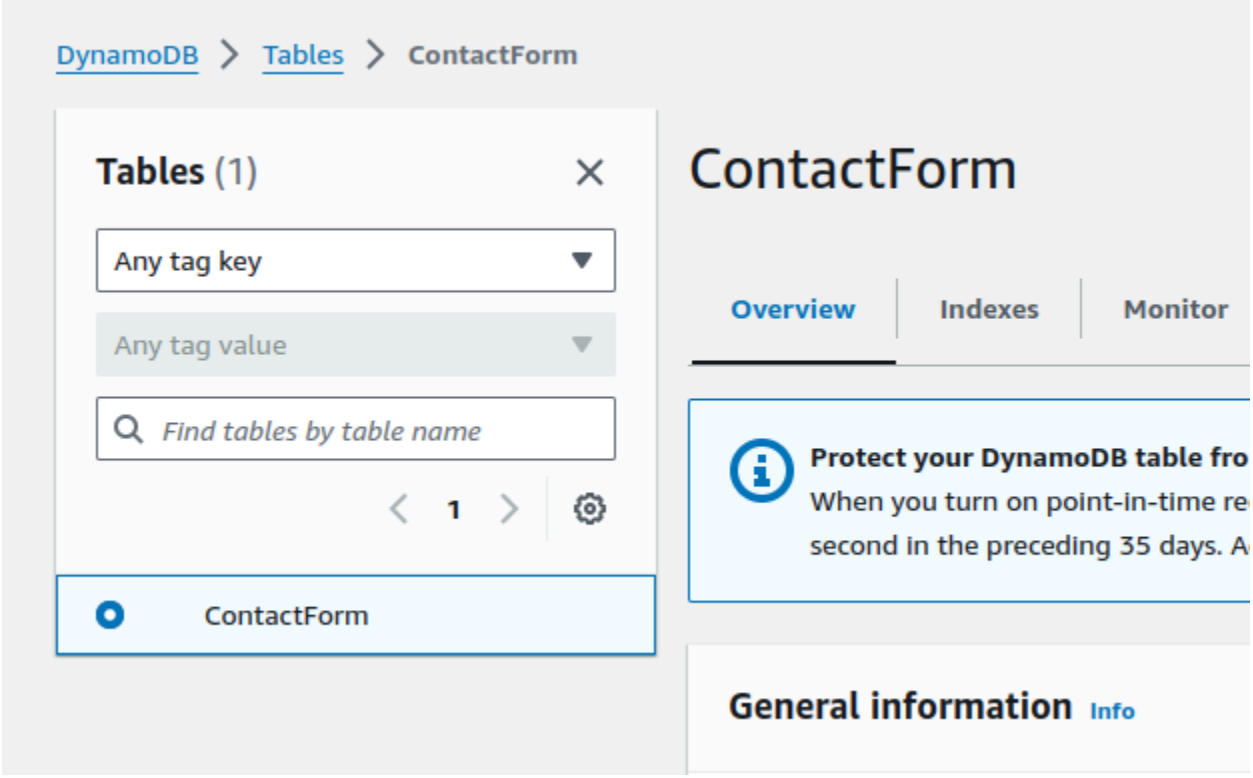
1. **Create S3 bucket with necessary information and policy, after the bucket is created upload the static page file and enable the static hosting to true under properties.**

```
{
  "Version": "2012-10-17",
  "Id": "Policy1706019486631",
  "Statement": [
    {
      "Sid": "Stmt1706019482372",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::bucketfor-api-gateway/*"
    }
  ]
}
```

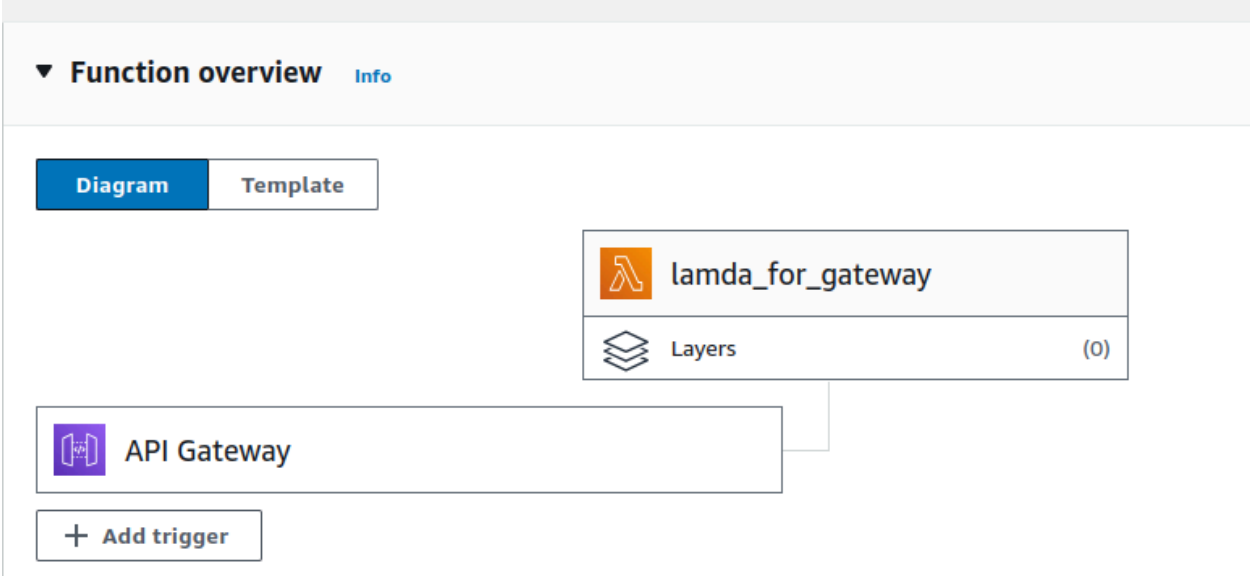




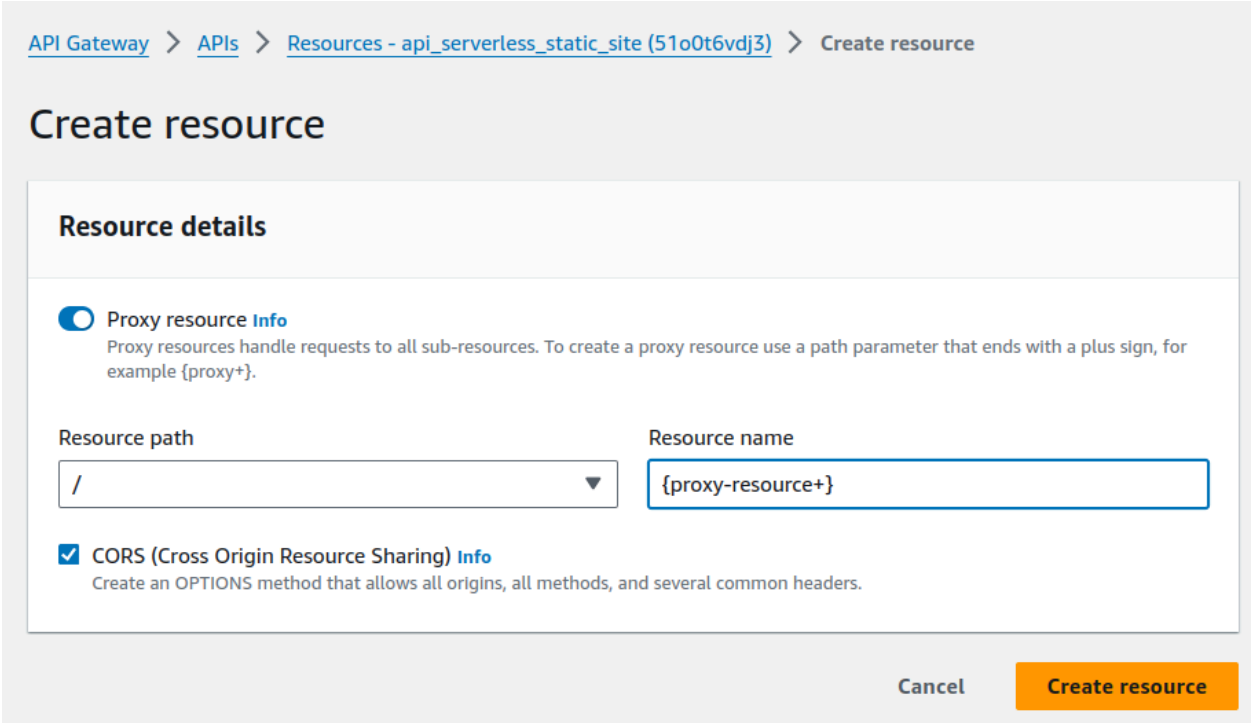
2. **Create table in dynamo db with the necessary database configuration**



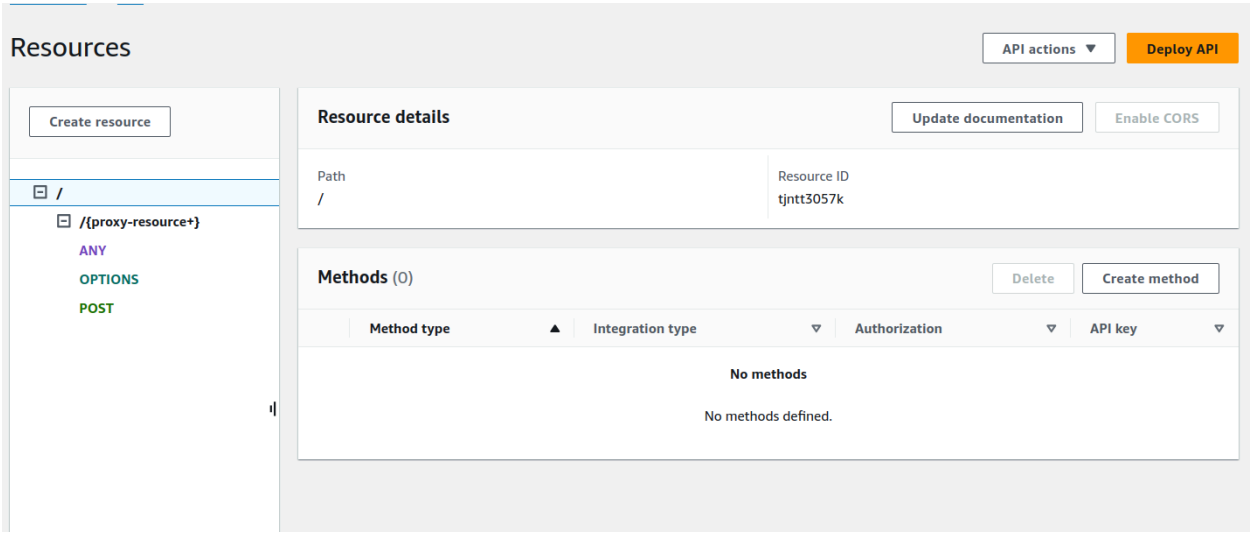
3. **Create lambda function with API Gateway with necessary configuration**



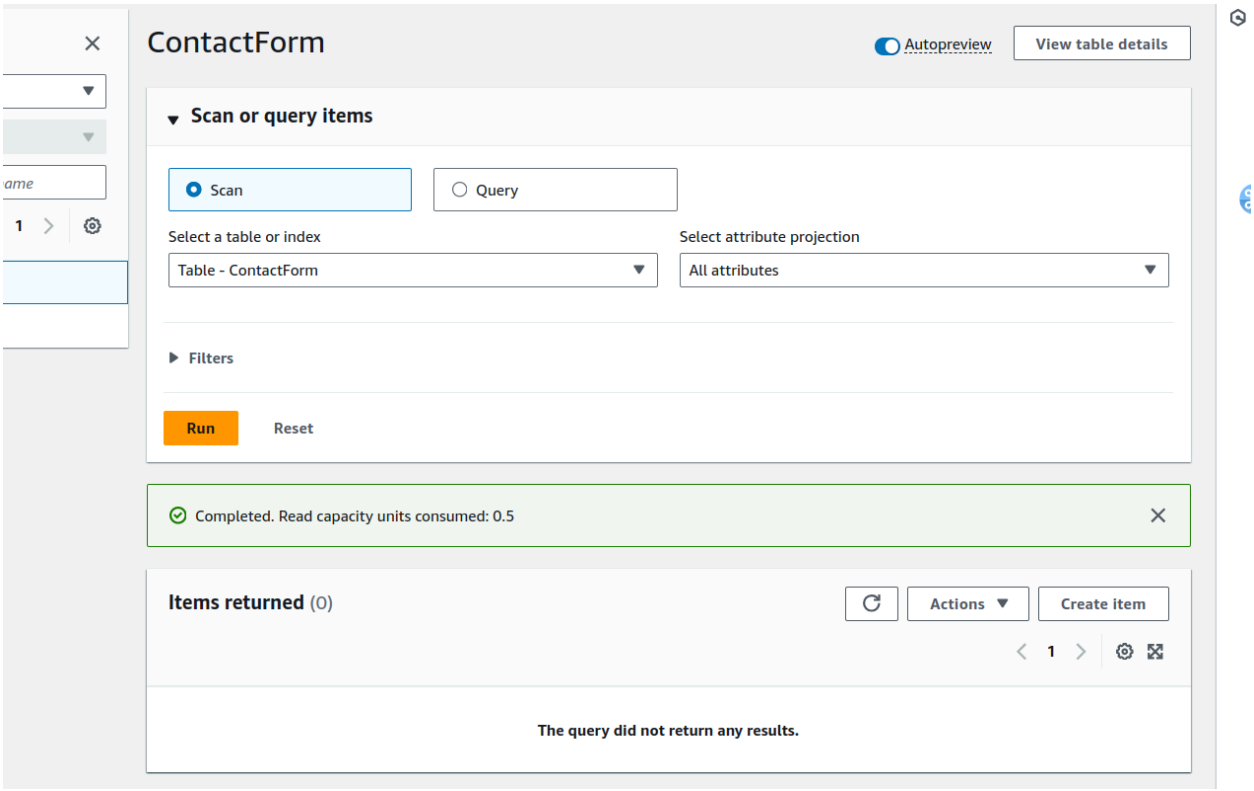
4. **Now create a Rest api with resource while creating resource enable the proxy resource(by default it will be non-proxy integration which will only get the request payload data in the event not other information such as header etc)**



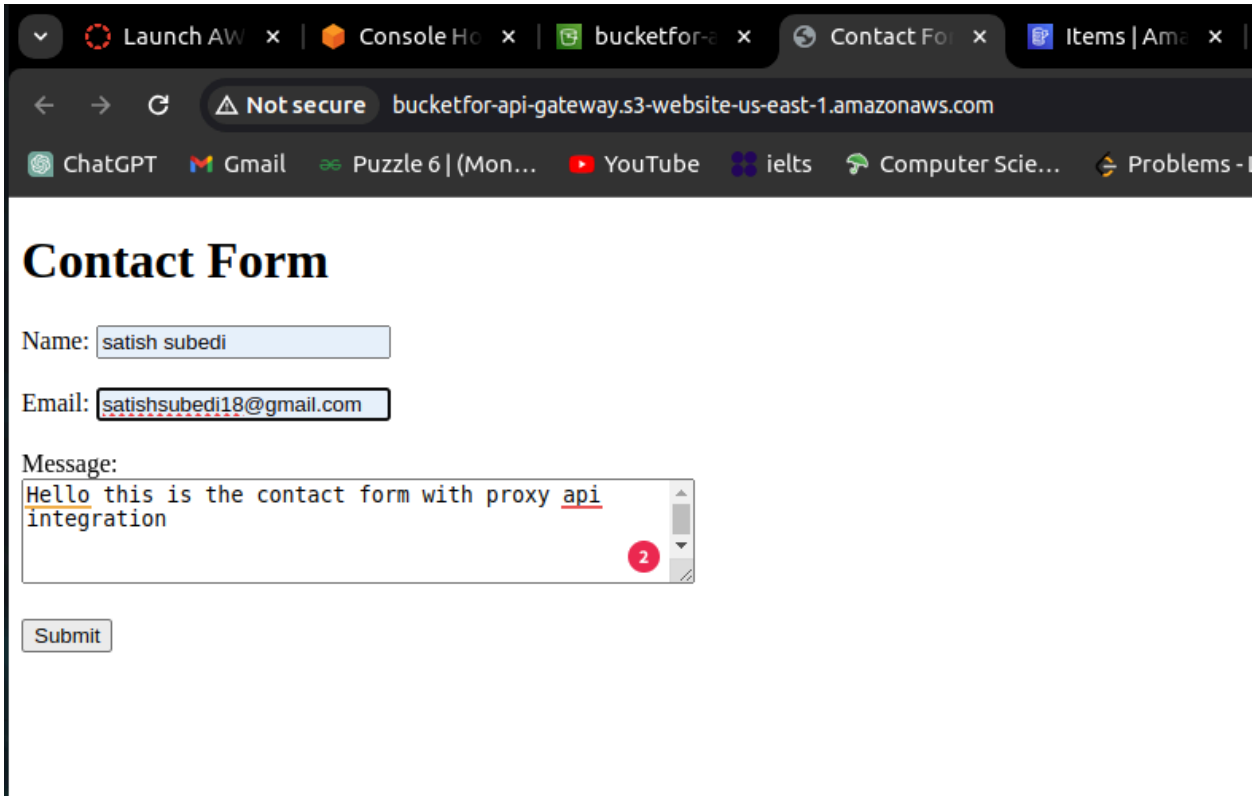
API Gateway



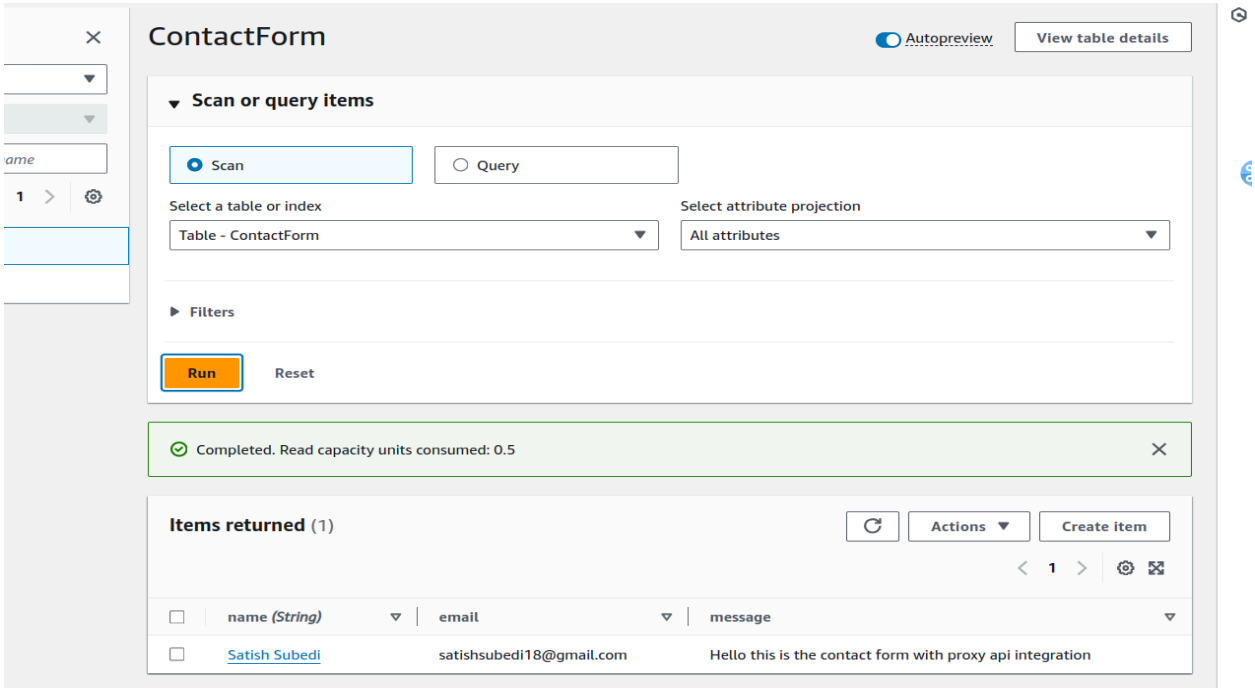
5. **There is no records recently in my dynamo DB recently**



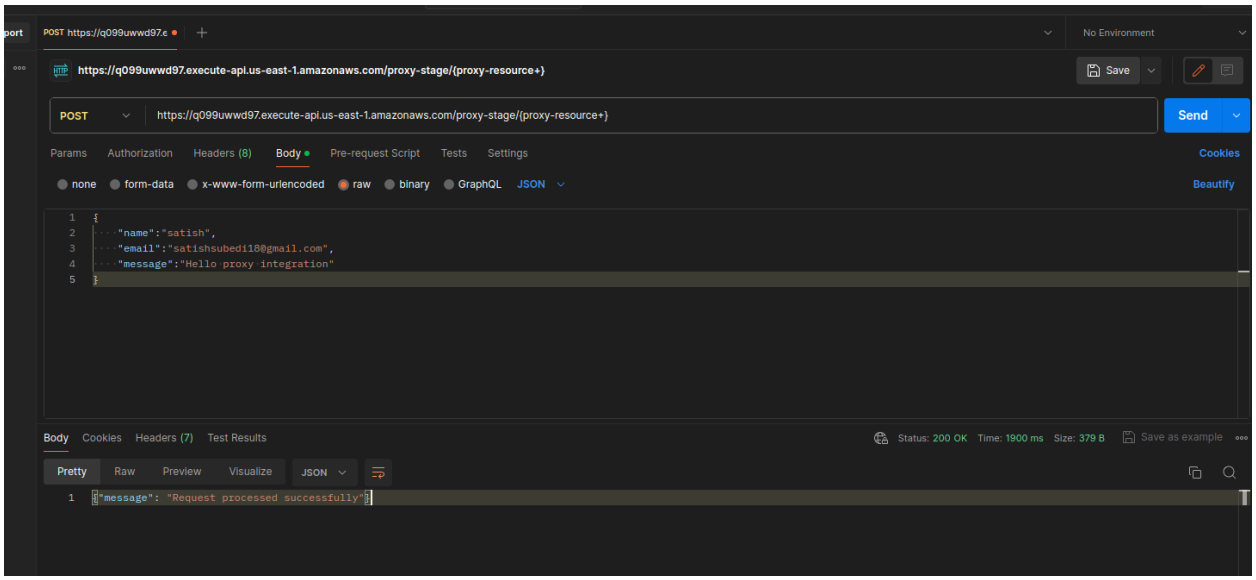
6. **Now go to the static site and fill the contact form and click submit button**



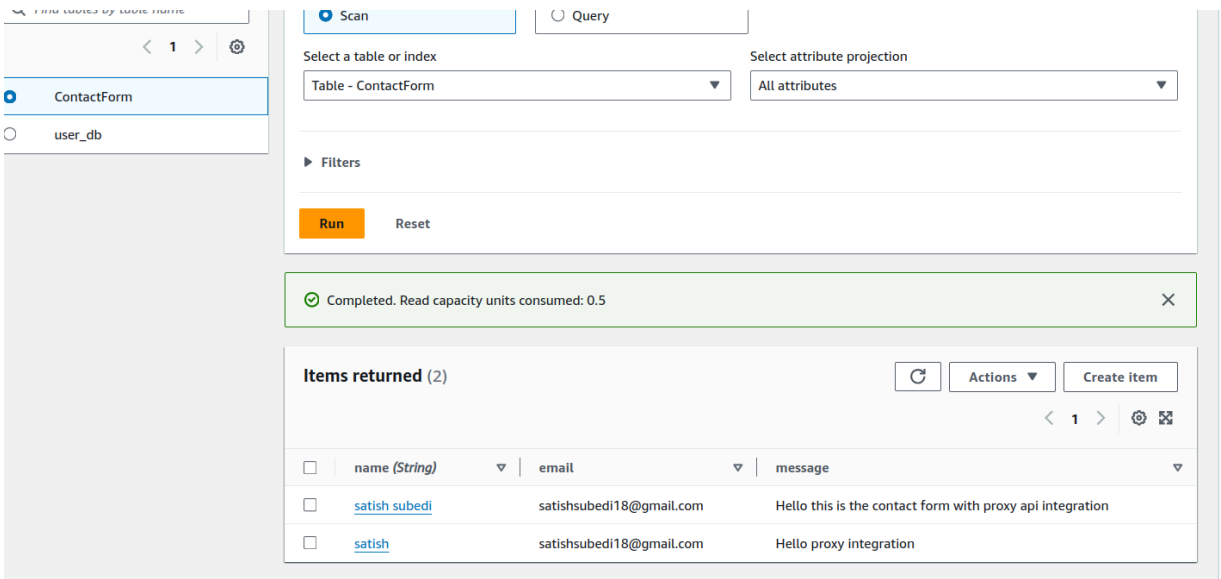
After the post call we can see the record is inserted in dynamodb



Postman test



Result



Backend code

```python
def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('ContactForm')

    try:
        http_method = event['httpMethod']

        if http_method == "POST":
            request_body = json.loads(event['body'])
            name = request_body.get('name', '')
            email = request_body.get('email', '')
            message = request_body.get('message', '')

            item ={
                'name':name,
                'email':email,
                'message':message
            }

            table.put_item(Item=item)
            response = {
                "statusCode": 200,
                "body": json.dumps({"message": "Request processed successfully"})
            }
    except Exception as e:
        # Handle any exceptions
        print("Error:", e)
        # Return error response
        response = {
            "statusCode": 500,
            "body": json.dumps({"message": "Internal Server Error"})
        }

    return response
```

2. Creating a Serverless API

Objective: Develop a serverless API using AWS Lambda and API Gateway.

Approach:

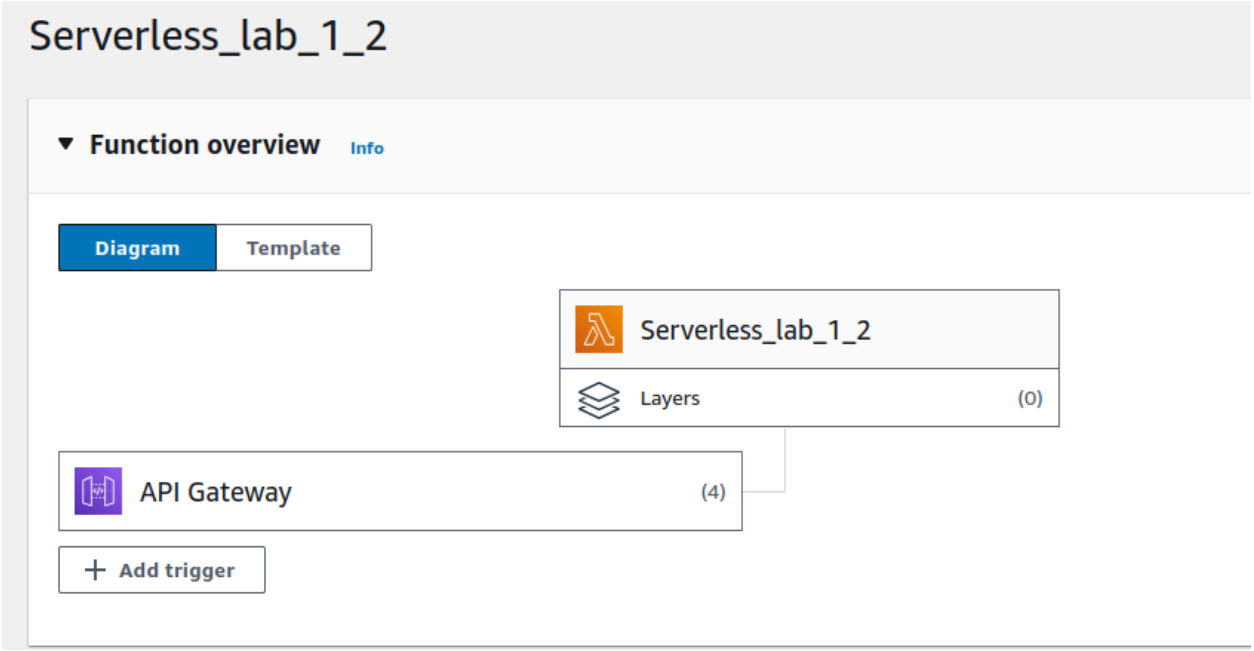
- **Define API:** Design a simple RESTful API (e.g., for a todo list application).
- **Lambda Functions:** Create Lambda functions for each API method (GET, POST, PUT, DELETE).
- **API Gateway Setup:** Use API Gateway to set up the API endpoints, connecting each endpoint to the corresponding Lambda function.
- **Testing:** Test the API using tools like Postman or AWS API Gateway test functionality.

Goal: Gain hands-on experience in building and deploying a serverless API, understanding the integration between Lambda and API Gateway.

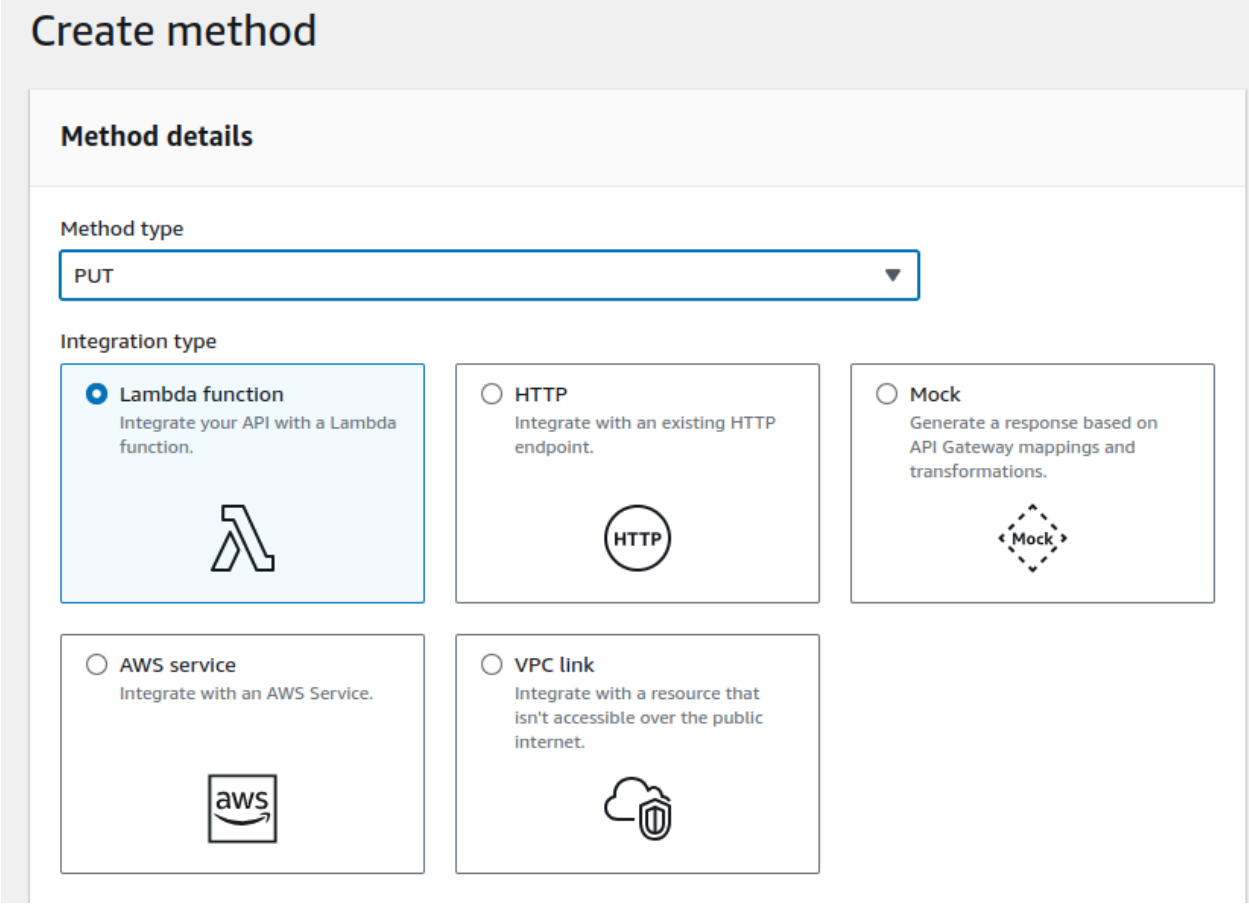
GENERAL STEPS

1. **Create the lambda function with the labrole IAM**

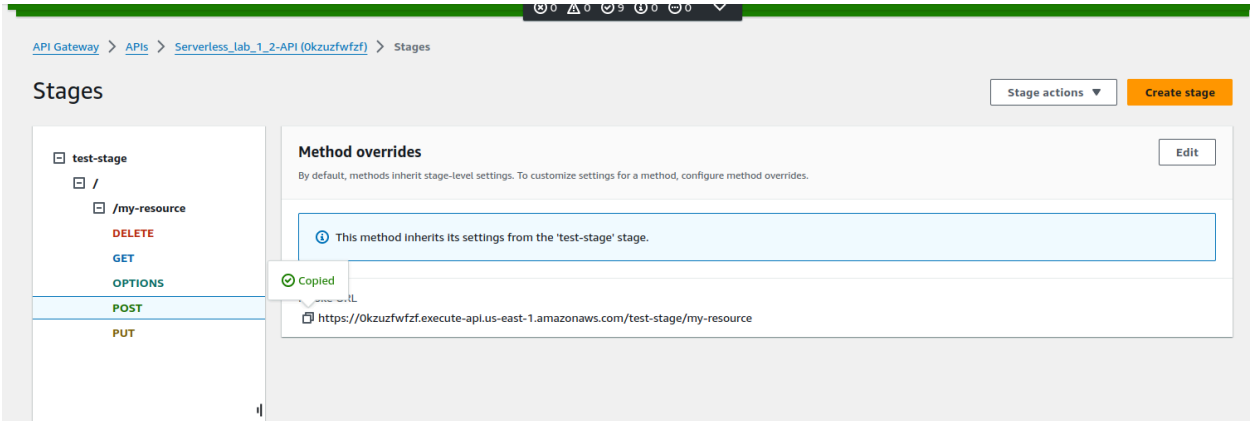
2. **Click the add trigger to add the api gateway**



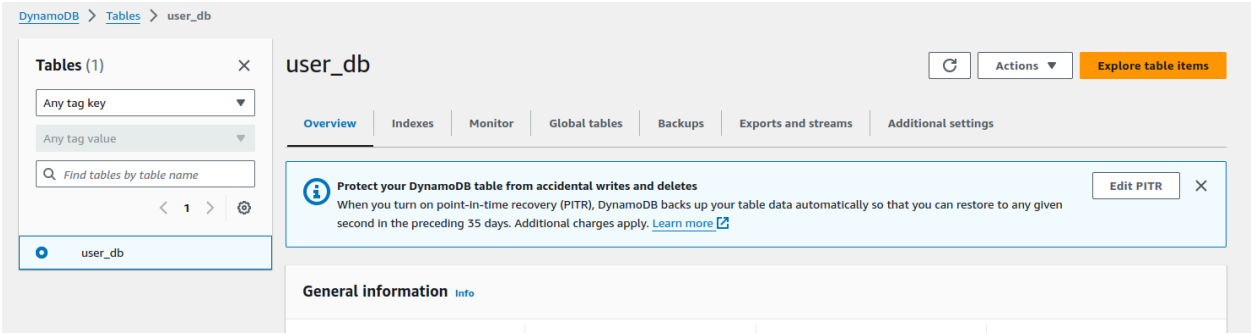
3. **Now go to api gateway and create the end point for the get,post,put and delete With resources and stage, while creating api end point choose the integration type to lambda and select arn of the lambda that you have created**



After the end point is created deploy the api



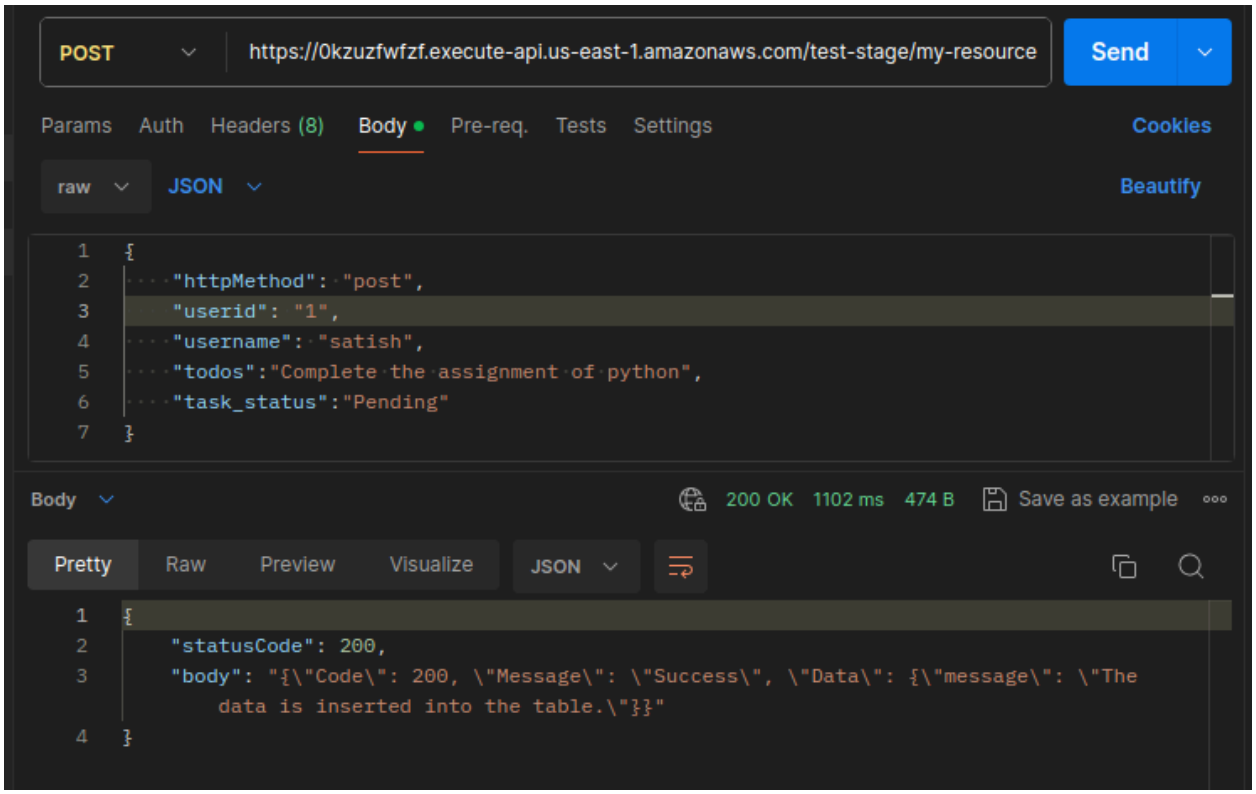
4. **Create DynamoDB with the necessary information to perform crud operation on it**



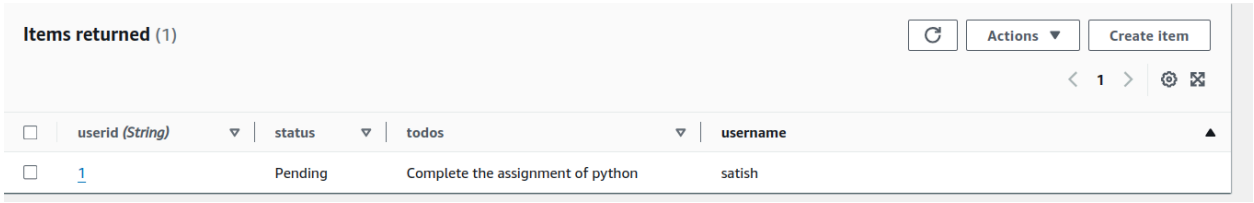
********** **TODO APPLICATION** **********

POST ACTION CALL VIA POSTMAN

Using the api endpoint like and making post request via postman to insert todo application details like userid, username,todo,status



Here data is inserted lets check in the dynamo db



Code for the post method

```python
def handle_post_request(event):
    try:
        userid = event.get('userid')
        username = event.get('username')
        todos = event.get('todos')
        task_status = event.get('task_status')
        item = {
            'userid': userid,
            'username': username,
            'todos':todos,
            'task_status':task_status
        }

        # Put item into DynamoDB
        table.put_item(Item=item)

        return build_response(200, "Success", {"message": "The data is inserted into the table."})
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))
```

GET ACTION CALL VIA POSTMAN

We had just three data for the todo application in our db so let's extract them all via api call

Items returned (3)

Actions ▼ | Create item

1

userid *(String)*	task_status	todos	username
1	Pending	Complete the assignment of python	satish
2	Completed	Complete the assignment of aws	shailav
3	Not Started	Complete the assignment of SQL	Sushant

Data for all users in postman

GET https://0kzuzfwfzf.execute-api.us-east-1.amazonaws.com/test-stage/my-resource Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
    "httpMethod":"get"
}
```

Body 200 OK 371 ms 797 B Save as example

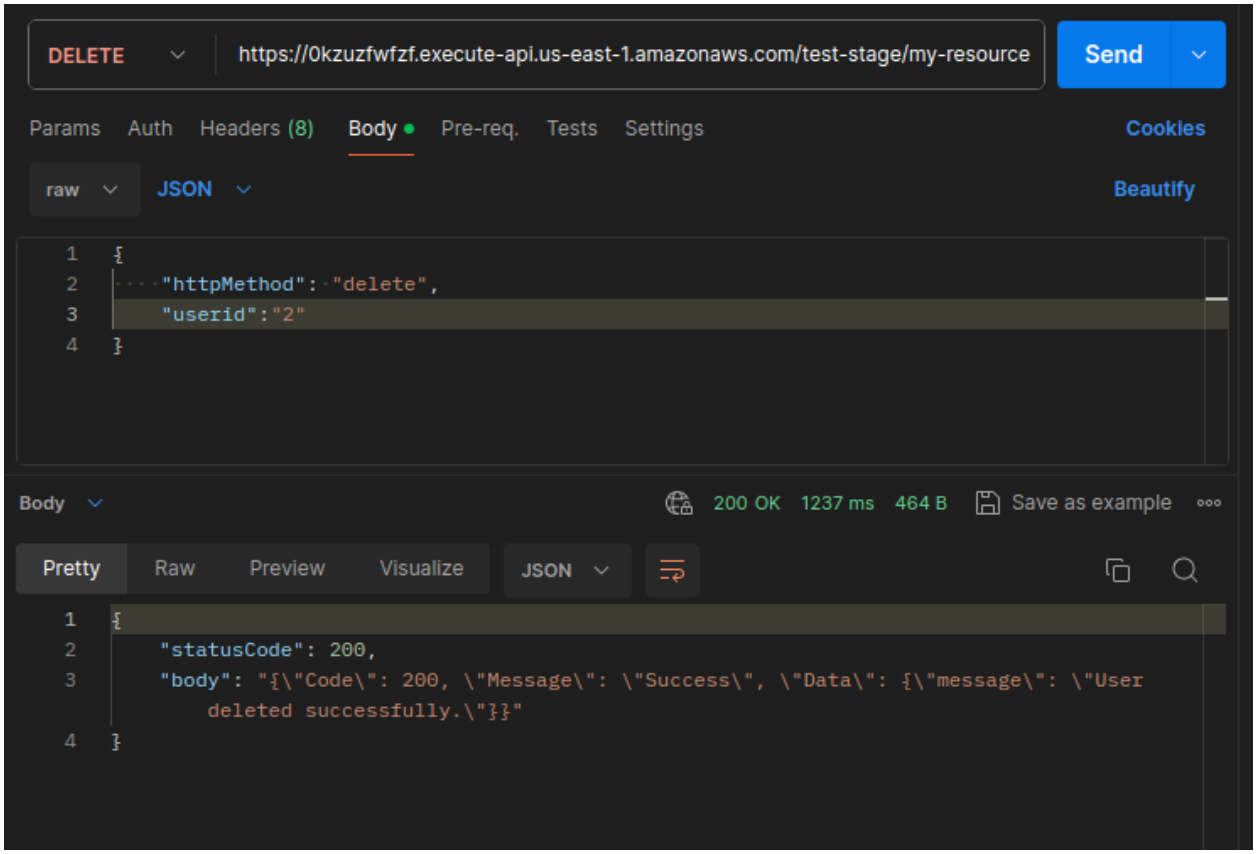
Pretty Raw Preview Visualize JSON


```
{
    "statusCode": 200,
    "body": "{\"Code\": 200, \"Message\": \"Success\", \"Data\": [{\"username\":
        \"shailav\", \"task_status\": \"Completed\", \"userid\": \"2\", \"todos\":
        \"Complete the assignment of aws\"}, {\"username\": \"satish\", \"task_status\":
        \"Pending\", \"userid\": \"1\", \"todos\": \"Complete the assignment of
        python\"}, {\"username\": \"Sushant\", \"task_status\": \"Not Started\",
        \"userid\": \"3\", \"todos\": \"Complete the assignment of SQL\"}]}"
}
```

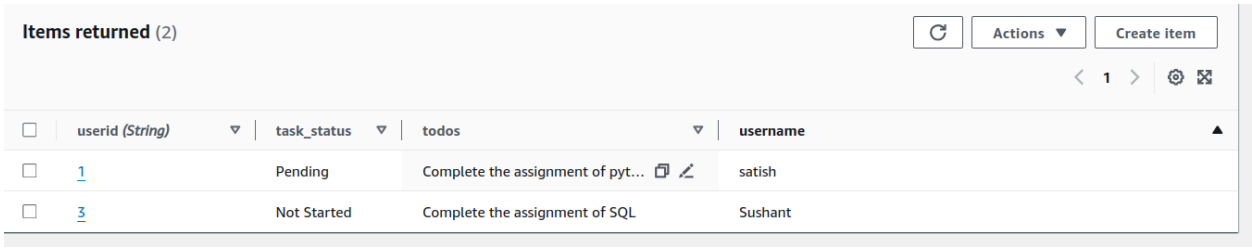

Code for get request

```python
def handle_get_request(event):
    try:
        result = table.scan()
        items = result.get("Items",[])
        return build_response(200, "Success", items)
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))
```

DELETING THE RECORD FOR THE USERNAME SHAILAV FROM DATABSE USING API



Lets check our database for the conformation for userid 2, as you can see in the database the data for userid is deleted



Code for delete

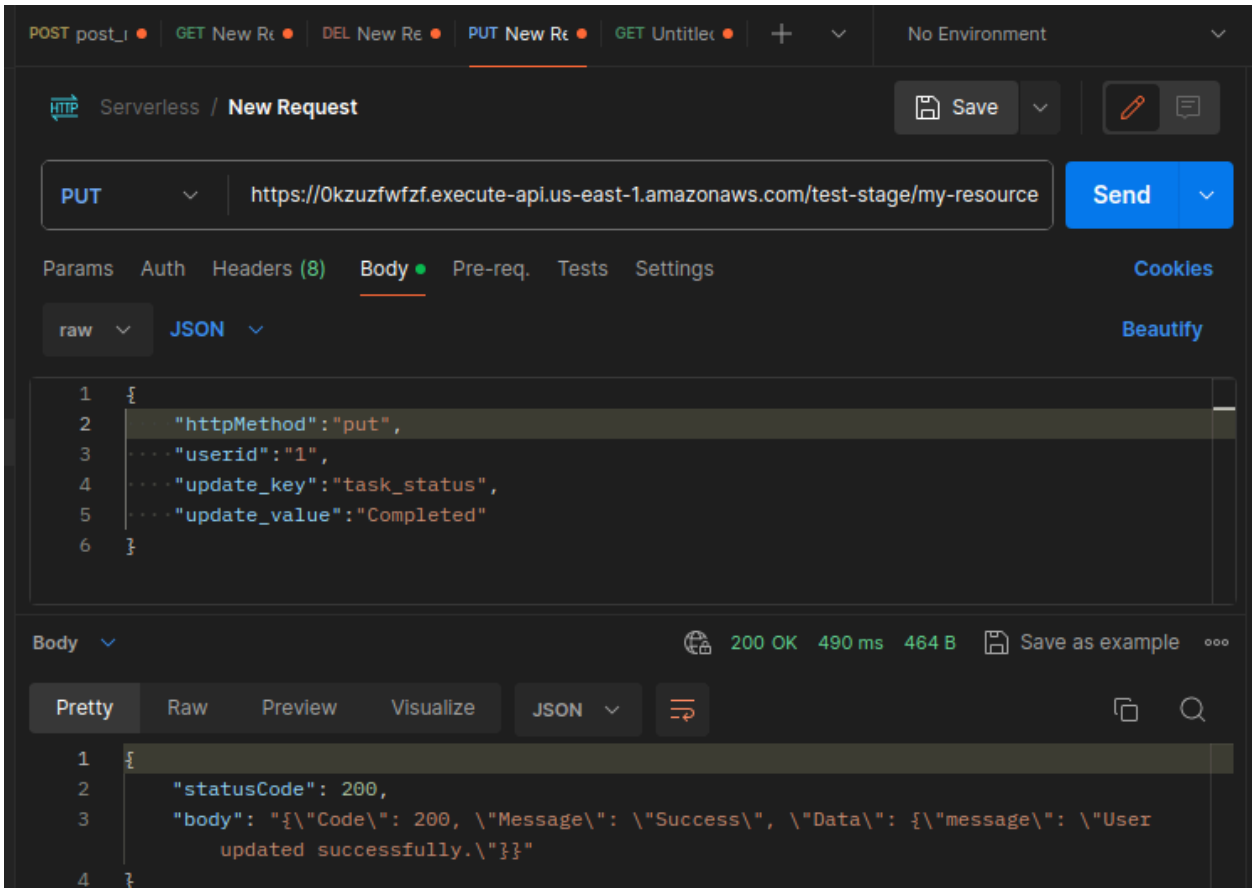
```
def handle_delete_request(event):
    try:
        userid = event.get('userid')
        table.delete_item(Key={'userid': userid})
        return build_response(200, "Success", {"message": "User deleted successfully."})
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))
```

UPDATING THE RECORD IN DATABASE BY API CALL

So we know we have a record for satish which status is pending so lets update that status to completed via postman

userid *(String)*	task_status	todos	username
1	Pending	Complete the assignment of pyt...	satish

Update call via api



Let's check the database for the conformation

userid *(String)*	task_status	todos	username
1	Completed	Complete the assignment of python	satish

Now you can see the status is changed to completed

Code

```python
def handle_put_request(event):
    try:
        userid = event.get('userid')
        update_key = event.get('update_key')
        update_value = event.get('update_value')

        table.update_item(
            Key={'userid': userid},
            UpdateExpression=f'SET {update_key} = :value',
            ExpressionAttributeValues={':value': update_value},
            ReturnValues='UPDATED_NEW'
        )
        return build_response(200, "Success", {"message": "User updated successfully."})
    except ClientError as e:
        return build_response(500, "Internal Server Error", str(e))
```

3. Serverless Data Processing Pipeline

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

Approach:

- **Data Ingestion**: Use AWS services like S3 or Kinesis to ingest data.
- **Processing**: Create Lambda functions to process the ingested data.
- **Storage**: Store the processed data in an appropriate AWS service, like S3 or DynamoDB.
- **Monitoring**: Set up CloudWatch to monitor the pipeline's performance and to log any issues.

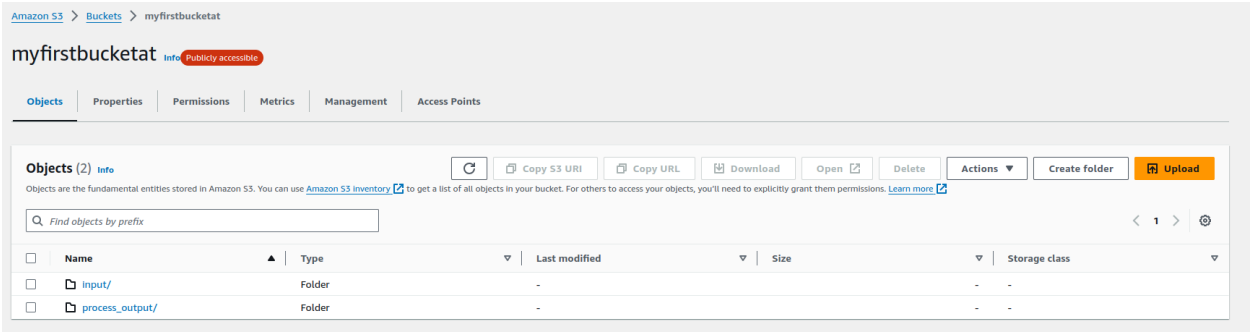
Goal: Learn to build a serverless data processing pipeline, understanding the flow of data through various AWS services.

GENERAL STEPS

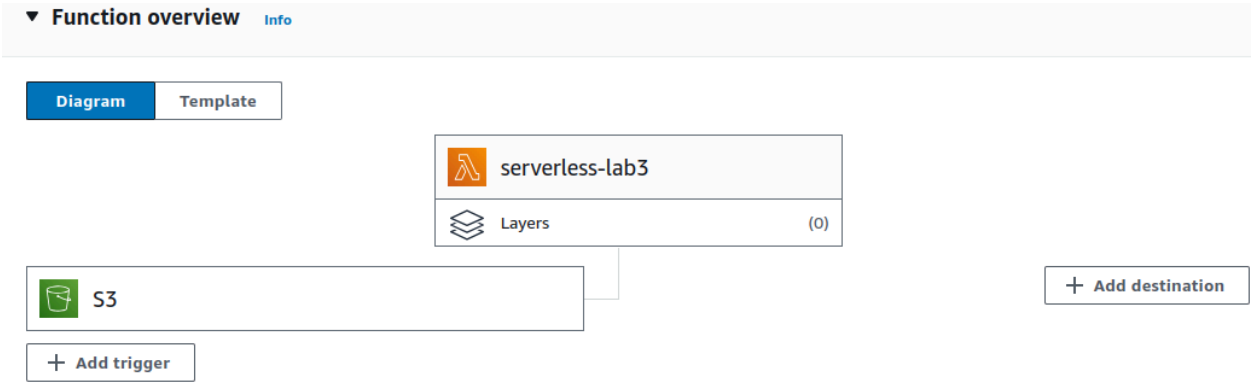
Lab to read the text file and process the file to convert in uppercase

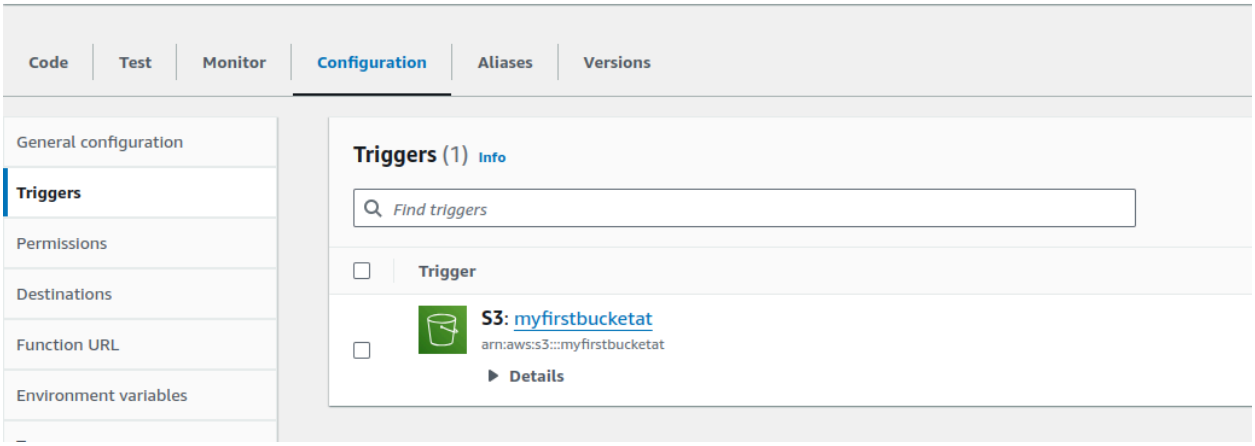
##

1. **Create the S3 bucket with the necessary information and assign the policy for the particular bucket and create two folder input and processed_output to put the process data.**



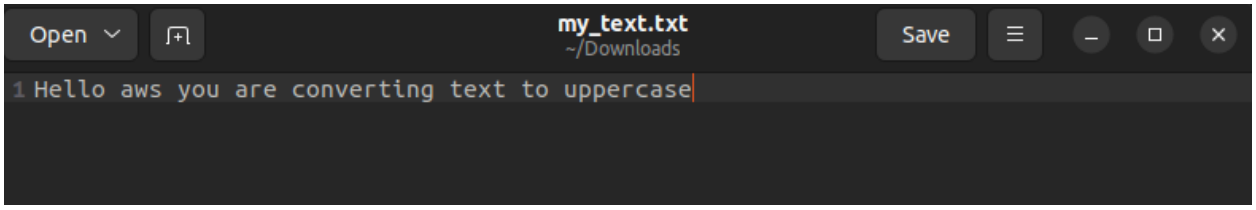
2. **Create a lambda function and add trigger as a S3 note add necessary details while creating the lambda triggers for avoiding any problems**



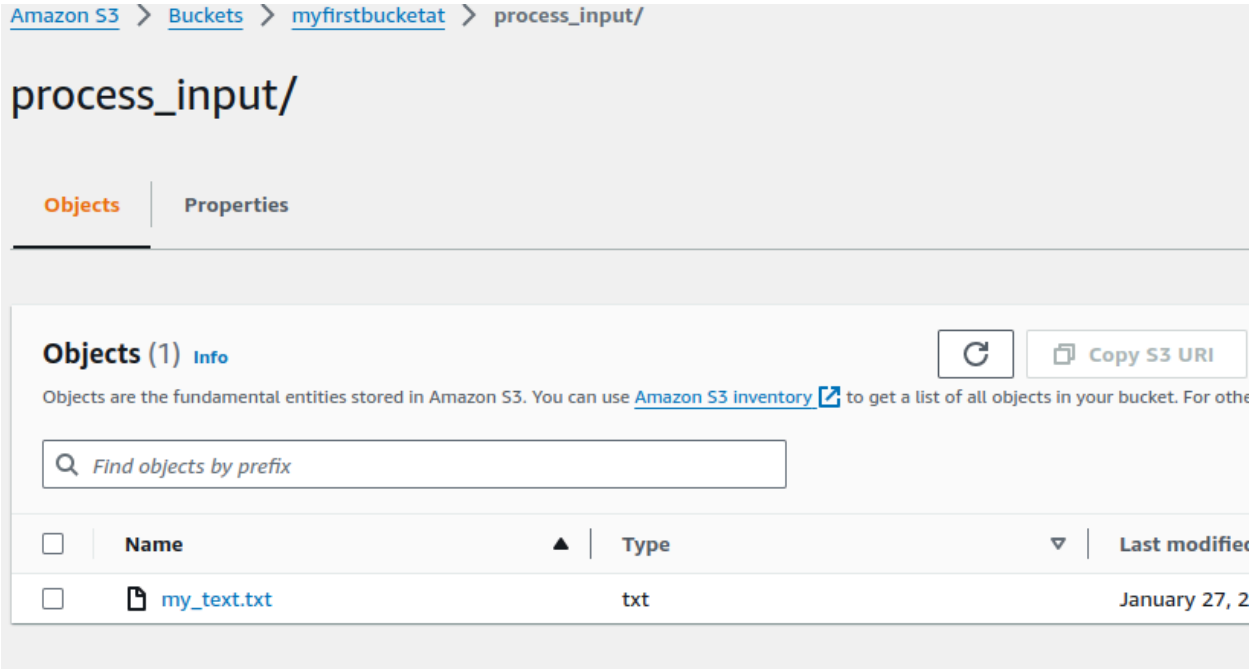


3. **Now write a code in lambda to read the input file and put the processed the text file to convert all the text in the file to upper case and put in process_input folder.**

Sample_text_file



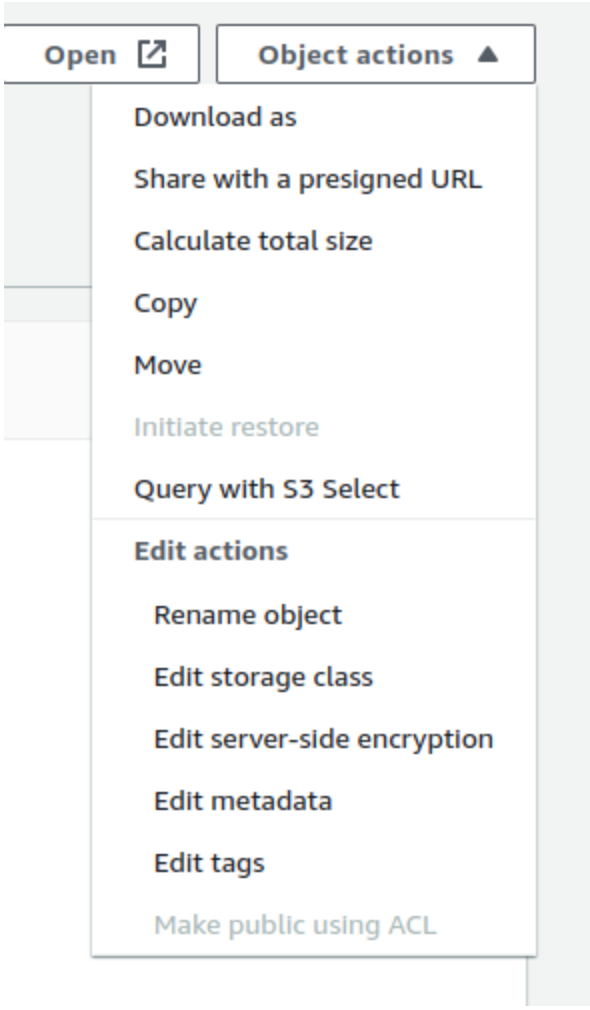
Output file



Sample code

```python
import boto3
import os
from io import BytesIO

def lambda_handler(event, context):
    s3_client = boto3.client('s3')

    # Retrieve the bucket and key from the S3 event
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Read the file from S3
    response = s3_client.get_object(Bucket=bucket, Key=key)
    content = response['Body'].read().decode('utf-8')

    # Convert the content to uppercase
    upper_content = content.upper()

    # Specify destination folder within the same bucket
    destination_folder = 'process_input'


    destination_key = os.path.join(destination_folder, os.path.basename(key))

    s3_client.put_object(Body=upper_content.encode('utf-8'), Bucket=bucket, Key=destination_key)

    print(f"File {key} converted to uppercase and saved to {destination_key}")
```

4. **Select the processed file and click object action in your right side and select query with s3 to get the result**


Open
Object actions
Download as
Share with a presigned URL
Calculate total size
Copy
Move
Initiate restore
Query with S3 Select
Edit actions
Rename object
Edit storage class
Edit server-side encryption
Edit metadata
Edit tags
Make public using ACL

Output

SQL query

Amazon S3 Select supports only the SELECT SQL command. Using the S3 console, you can extract up to 40 MB of records from Amazon Athena

```
1  /* To create reference point for writing SQL queries, you can display the fi
2  SELECT * FROM s3object
```

SQL Ln 2, Col 23 Errors: 0 Warnings: 0

Query results

Query results are not available after you choose **Close** or navigate away. Choose **Download results** to download a copy of the fo

Status

Successfully returned 1 record in 828 ms

Bytes returned: 47 B

```
1  HELLO AWS YOU ARE CONVERTING TEXT TO UPPERCASE
2
```

5. **Goto AWS console and select CloudWatch to monitor the log from lambda function**

Select to log for the appropriate lambda

/aws/lambda/serverless-lab3	Standard

Log file

No more records within selected time range *Retry*

2024-01-27T18:32:37.680+05:45	INIT_START Runtime Version: python:3.12.v18 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:776a3759221679a634181f858871d5514dc74a176f78bc535f822a932845…
2024-01-27T18:32:37.984+05:45	START RequestId: 79d9cfe4-e830-4e28-9ab5-a2e2836b836d Version: $LATEST
2024-01-27T18:32:40.705+05:45	File input/my_text.txt converted to uppercase and saved to process_input/my_text.txt
2024-01-27T18:32:40.731+05:45	END RequestId: 79d9cfe4-e830-4e28-9ab5-a2e2836b836d
2024-01-27T18:32:40.731+05:45	REPORT RequestId: 79d9cfe4-e830-4e28-9ab5-a2e2836b836d Duration: 2746.62 ms Billed Duration: 2747 ms Memory Size: 128 MB Max Memory Used: 81 MB Init Duration: …

No more records within selected time range *Auto retry paused.* ***Resume***