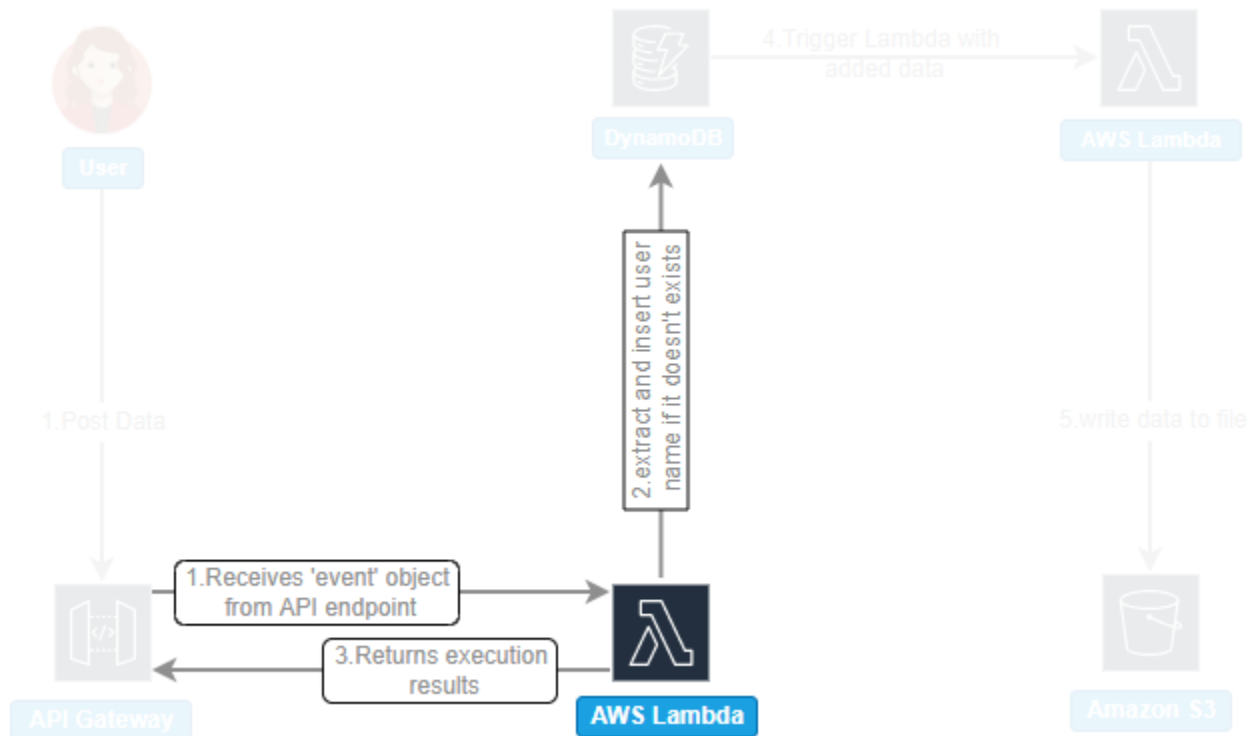


3. API lambda setup

Now let's start targeting individual parts of the solution one by one. We will first start with lambda development for API endpoint !!

This lambda should mainly perform the following operations

1. Receives input from API endpoint in 'event' parameter.
2. Extracts and validates and insert 'name' provided by user in a query string into DynamoDB table
3. Returns success or failure to API endpoint depending on results of execution.



To start with point 1 we should know what is the structure of event object. Since we have not yet started using AWS platform we need to first get understanding of how event object from API endpoint looks like

There are several ways we can get event object struct in AWS as documented [here](#).

Create a new folder under 'src' with name 'events' and add sample event for apigateway in apigateway-aws-proxy.json file so that it will be easy to refer them later.

The screenshot shows the VS Code interface. On the left, the Explorer pane shows the project structure with a folder named 'events' under 'src'. The file 'apigateway-aws-proxy.json' is selected. On the right, the editor shows the content of this file, which is a JSON object representing an API Gateway event. The JSON structure includes 'body', 'resource', 'path', 'httpMethod', 'isBase64Encoded', 'queryStringParameters', and 'multiValueQueryStringParameters'.



```
{
  "body": "eyJ0ZXN0Ijo1Ym9keS9",
  "resource": "/{proxy+}",
  "path": "/path/to/resource",
  "httpMethod": "POST",
  "isBase64Encoded": true,
  "queryStringParameters": {
    "foo": "bar"
  },
  "multiValueQueryStringParameters": {
    "foo": [
      "bar"
    ]
  },
  "pathParameters": {
```


add following default lambda code in api_lambda.py

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

Now lets start processing 'event' from API gateway.

  Do we see hardcoding in logic here ? YES we do !!!

Please keep in mind this part is focusing on getting hands on the development process around AWS services and how to effectively write the code logic. We are deliberately keeping few things simple for ease of understanding. Trust me...there are lot of things ahead for a beginner to understand, so be patient 

To start with the code don't focus on things like modular architecture, standards, clean code, error handling scenarios etc. First pass should be focused on writing basic working code. Later passes can focus on remaining things.

Replace lambda handler code with following code. This is only dummy code to extract user provided name from query parameter.

```

import json

def lambda_handler(event, context):
    # Extract required details from event object.
    # Remember structure of event object stored in apigateway-aws-proxy.
    json
    user_name = event['queryStringParameters']['name']
    request_method = event['httpMethod']
    request_path= event['path']

    # check for POST operation on user endpoint
    if request_path == "\user":
        if request_method == 'POST':
            #call post handler for user endpoint
            post_handler_return = user_post_handler(user_name)

    # create a response object to return as API endpoint response
    response_object = {}
    response_object['statusCode'] = 200 if post_handler_return else 400
    response_object['headers'] = {}
    response_object['headers']['Content-Type'] = 'application/json'
    response_object['body'] = 'success' if post_handler_return else
'failure'

    # return from lambda
    return response_object

# dummy post handler for user endpoint
# whihc always return true to start with
def user_post_handler(user_name):
    # TODO add user name to DynamoDB

    return True

```

Now lets add first operation to add user name to DynamoDB table. Since we have a separate function 'user_post_handler' for managing this we will start with the function.

Updated code looks like this. Don't panic !!! we will go though things in details.

```

import json
import os          # new
import boto3       # new
import uuid        # new
from botocore.exceptions import ClientError      # new

# Read environment variables

```

```

table_region = os.environ["REGION"]      # new
table_name = os.environ["TABLE_NAME"]    # new

def lambda_handler(event, context):

    # Extract required details from event object.
    # Remember structure of event object stored in apigateway-aws-proxy.
    json
    user_name = event['queryStringParameters']['name']
    request_method = event['httpMethod']
    request_path= event['path']

    # check for POST operation on user endpoint
    if request_path == "\user":
        if request_method == 'POST':
            #call post handler for user endpoint
            post_handler_return = user_post_handler(user_name)

    # create a response object to return as API endpoint response
    response_object = {}
    response_object['statusCode'] = 200 if post_handler_return else 400
    response_object['headers'] = {}
    response_object['headers']['Content-Type'] = 'application/json'
    response_object['body'] = 'success' if post_handler_return else
'failure'

    # return from lambda
    return response_object

# POST handler for user API endpoint
# DynamoDB table schema PK-UserID (string), SK-UserName (string)
# DynamoDB table should be in-place as a prerequisite
def user_post_handler(user_name):

    # create dynamodb table object
    ddb_resource = boto3.resource("dynamodb", region_name=table_region)
# new
    ddb_table = ddb_resource.Table(table_name) # new

    # Generate JSON to add user name in DynamoDB table.
    # we are adding UserID and unique value using uuid library
    insert_item = {
        'UserID': uuid.uuid1(),
        'UserName' : user_name
    }

    try:
        ddb_response = ddb_table.put_item(

```

```
        TableName=table_name,  
        Item=insert_item  
    )  
  
except ClientError as e:  
    raise e  
    #TODO better error handling
```

This code extracts 'Name' (user name) from query string parameter and adds it to DynamoDB table using uuid as unique PK.