**AWS API GATEWAY to Lambda Function - Create API Gateway Endpoint that will invoke a lambda function & return some JSON back to the client.**

**Requirement: Client has asked us to stubs/mock an API service with request & response, so that when that lambda is triggered a response is given, the response has a dynamic time and state and customer ID, which depends on request.**

****

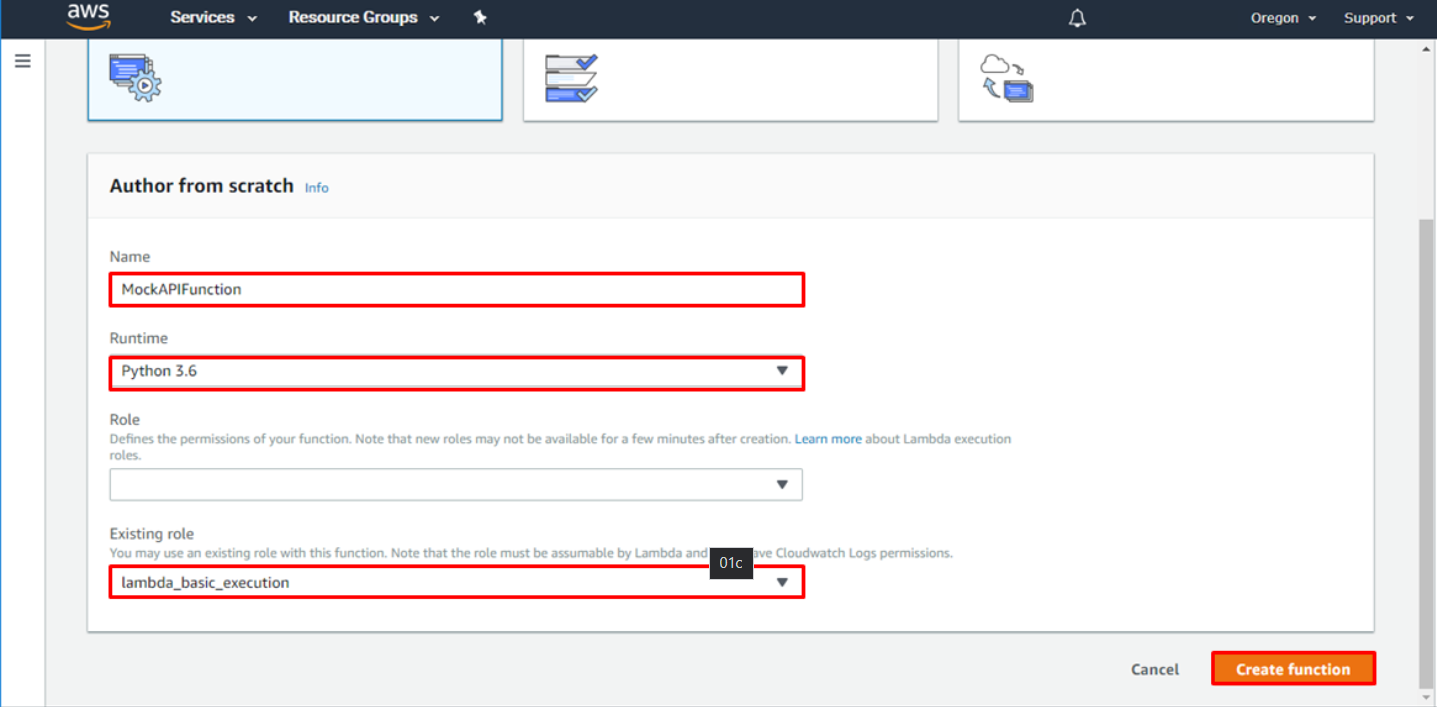
**Step 1: Create a Lambda Function that returns a hardcoded or dynamic response back to the Client.**

**Step 2: Setup a Get API, which will forward requests to the Lambda Function**

**Step 3 : Invoke the Get API from Postman or Browser**

**Step 4: Verify the Logs from Cloudwatch**

Lambda>> Create a function.



In the. Py or .js file, enter the code. Save & Deploy.

**Python Code**:

import json

print ('Loading function')

// The Lambda function handler is the method that processes events. When your function is invoked, Lambda runs the handler method.

// You can use the following general syntax when creating a function handler in Python:

**### def handler\_name(event, context):**

**##return some\_value**

def lambda\_handler(event, context):

#1. **Parse out query string parameters**

transactionId = event['queryStringParameters']['transactionId']

transactionType = event['queryStringParameters']['type']

transactionAmount = event['queryStringParameters']['amount']

print('transactionId=' + transactionId)

print('transactionType=' + transactionType)

print('transactionAmount=' + transactionAmount)

#2. **Construct the body of the response object**

transactionResponse = {} // Create an empty object & start filling the values

transactionResponse['transactionId'] = transactionId

transactionResponse['type'] = transactionType

transactionResponse['amount'] = transactionAmount

transactionResponse['message'] = 'Hello from Lambda AWS'

#3. **Construct a http response object**

responseObject = {} // Create an empty object & start filling the values

responseObject['statusCode'] = 200

responseObject['headers'] = {}

responseObject['headers']['Content-Type'] = 'application/json'

responseObject['body'] = json.dumps(transactionResponse)

// The Actual Body. We need to convert our object into a string, so we use the JSON library & set our response object that we created up.

#4. **Return the response object**

return responseObject

**Node JS Code:**

If there are variables in the response which needs to be fetched from request, pass it using **req.<name>**

#1 **Declare the Handler Function**

exports.handler = function(event,context,callback){

// function handler is the method in your function code that processes events. When your function is invoked, Lambda runs the handler method.

#2 **Declare the variables and conditions to capture the current time & current time + 30 mins**

var d = new Date();

var n = d.toISOString();

console.log("UTC time \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",n); // To print

var estDeliveryTime;

var utcTime = new Date();

// utcTime.setHours(utcTime.getHours() + utcTime.getTimezoneOffset()/60 - 0);

var year = utcTime.getFullYear()

var month = addZero(utcTime.getMonth()+1);

var day = addZero(utcTime.getDate());

var hours = addZero(utcTime.getHours());

var minutes = addZero(utcTime.getMinutes()+30);

var seconds = addZero(utcTime.getSeconds());

var miliSeconds = addZero(utcTime.getMilliseconds());

if(minutes>59){

var hours1 = addZero(utcTime.getHours()+1);

if (hours1>23){

hours1=00;

day=addZero(utcTime.getDate()+1);

}

var minutes1 = addZero(5);

console.log("hour from if minute >59 condition",hours1);

console.log("minutes from if minute >59 condition",minutes1);

estDeliveryTime = year + "-" + month + "-"+ day + "T" + hours1 + ":" + minutes1+ ":" + seconds + "." + miliSeconds+"000Z";

console.log("estimate DeliveryTime is ",estDeliveryTime);

}

else{

estDeliveryTime = year + "-" + month + "-"+ day + "T" + hours + ":" + minutes+ ":"+ seconds + "."+ miliSeconds +"000Z";

console.log("estimate DeliveryTime is ",estDeliveryTime); //"2020-10-01T18:20:42.395Z"

}

function addZero(n) {

// To add a leading zero if time digit only has one digit

return ( n < 0 || n > 9 ? "" : "0" ) + n;

};

#3 **Enter the variables of Response which will fetch data from Request with req.<name>**

let req = event;

var dropoff\_address = **req**.dropoff\_address;

var dropoff\_address\_city = dropoff\_address.city;

var dropoff\_address\_state = dropoff\_address.state;

var dropoff\_address\_street = dropoff\_address.street;

var dropoff\_address\_zip = dropoff\_address.zip\_code;

var dropoff\_address\_unit = dropoff\_address.unit;

#4 **Enter the Response inside a try catch, parametrize the response like jmeter ${name}**

try{

console.log("event is",event);

const estimates = `{

"delivery\_time": "${estDeliveryTime}",

"pickup\_window\_start\_time": null,

"fee": 549,

"dropoff\_address": {

"city": "${dropoff\_address\_city}",

"dasher\_parking\_details": null,

"state": "${dropoff\_address\_state}",

"street": "${dropoff\_address\_street}",

"unit": "${dropoff\_address\_unit}",

"zip\_code": "${dropoff\_address\_zip}"

},

"pickup\_time": "${n}",

"currency": "USD",

"pickup\_window\_end\_time": null,

"id": 1

}`;

callback(null,JSON.parse(estimates)) ;

}

catch(e){

callback(new Error("Error: failed to get response!"));

}

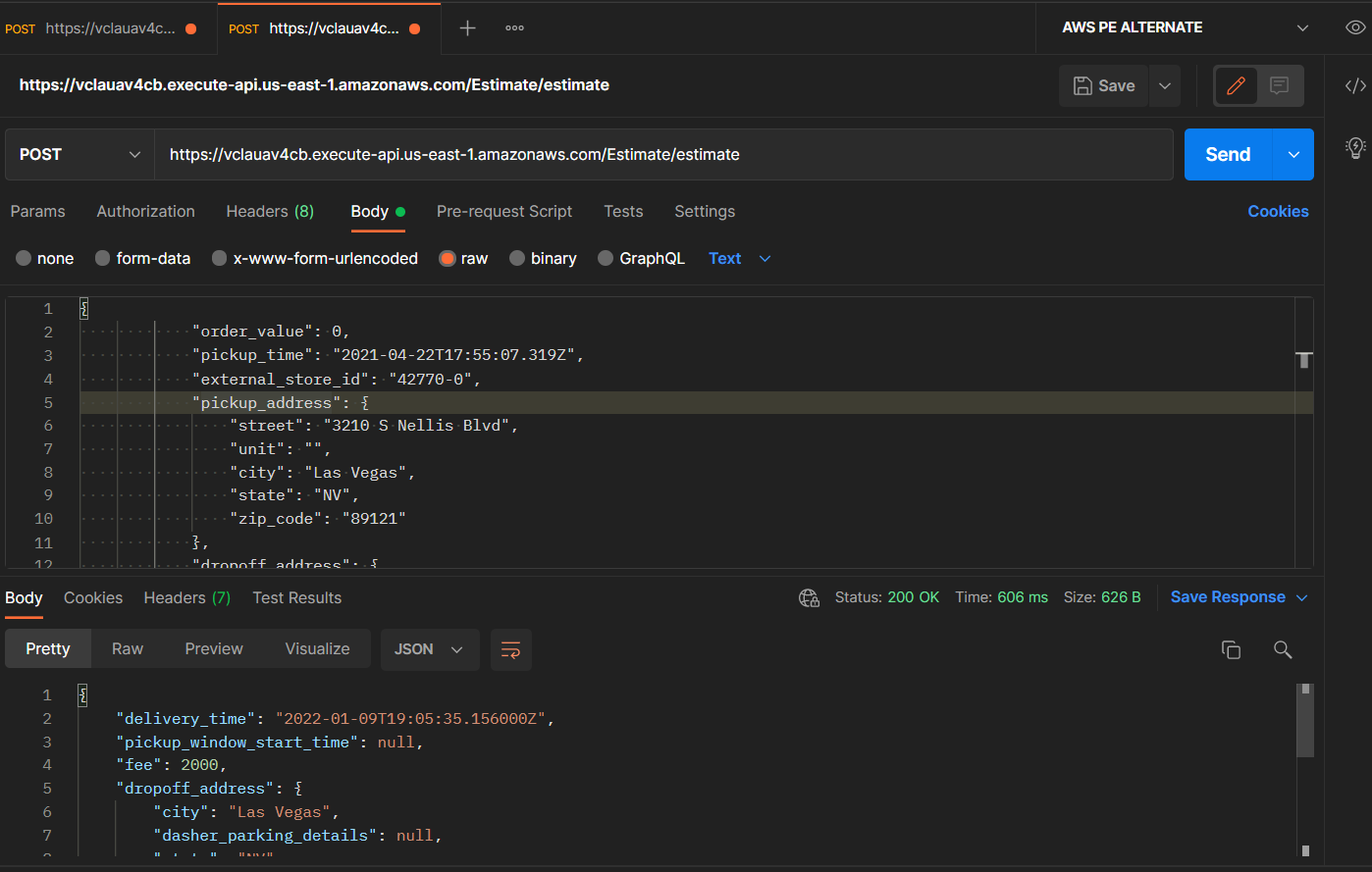
};

After this, save and deploy the lambda.

**Step 2: Setup a Get API, which will forward requests to the Lambda Function**

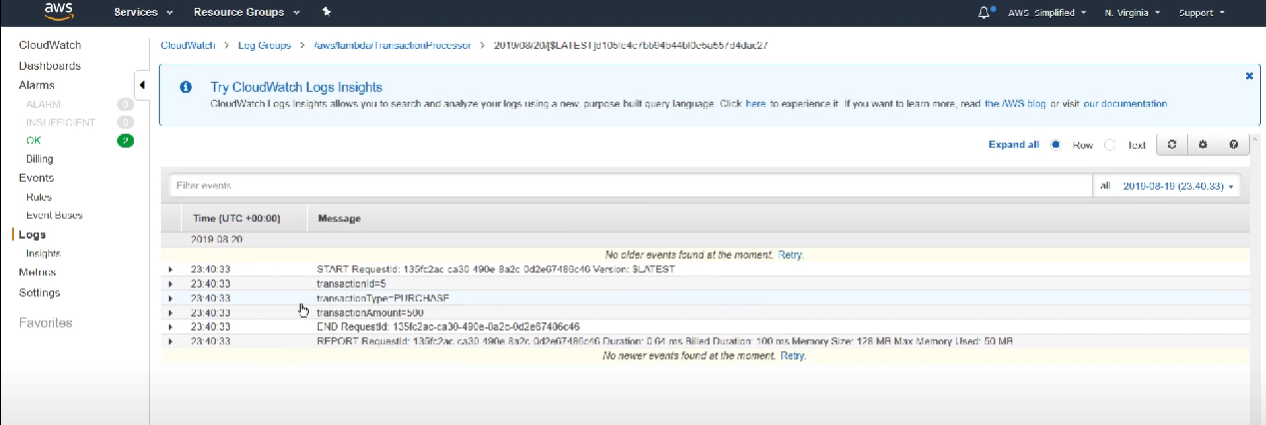
API Gateway Section>>Create API>>Create Resource>>Create Method>> GET or POST Method>> Map the Lambda Function with it>> Deploy API>>Create a Stage(in Test Environment)>> Get the URL Auto Generated

**Step3 : Invoke the Get API from Postman or Browser.**

****

**Step 4: Verify the Logs from Cloudwatch to check if everything is as expected.**

Cloudwatch>>Logs>>LambdaName>>Logstream

****