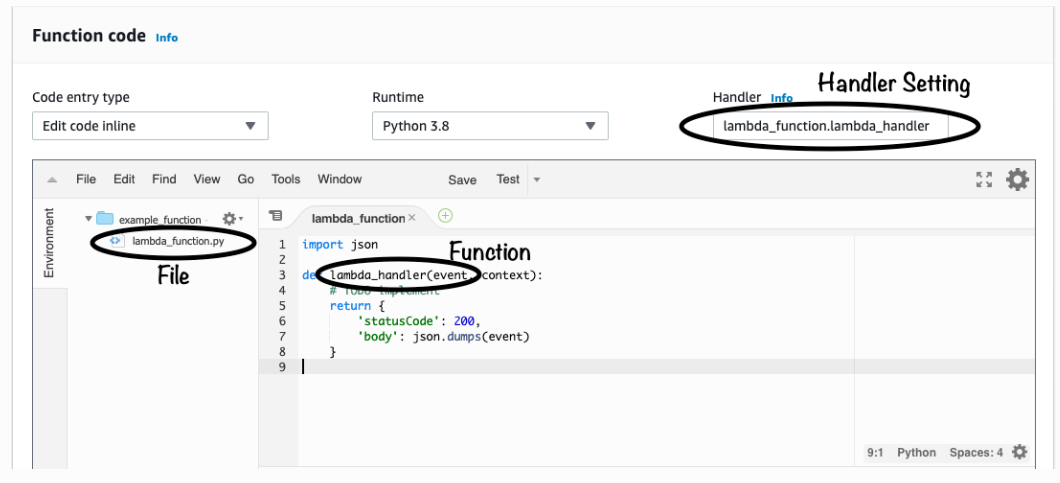
AWS Lambda is an Amazon serverless, full managed computing service that let's you to run the code. It is an event-driven computing service.

Let’s take a look of screenshot of lambda function from console.



**Handler/Handler function:**

The handler function is the starting point of your code. It's the python function that is executed when your lambda function runs.

By default handler will be --- lambda\_handler(event, context) but we can create our own handler.

**Creating Custom Handler:**

Open the lambda function and go to 'CODE' table then scroll down and go to EDIT.

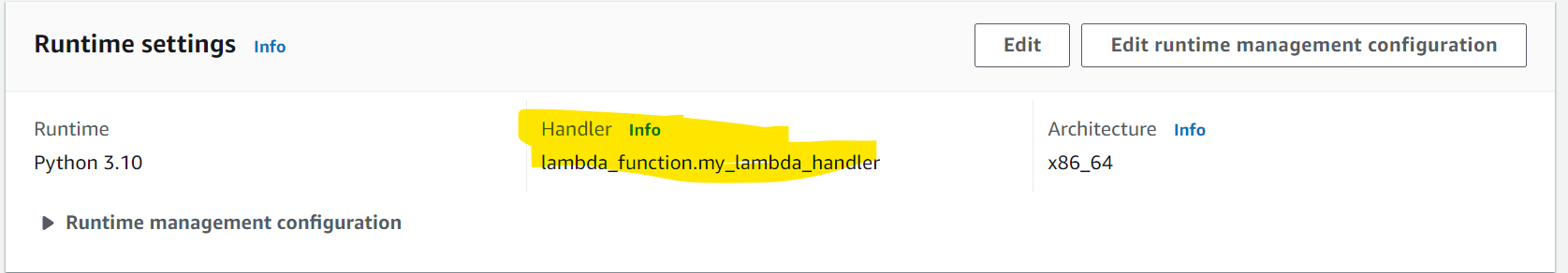
Here we can define our custom handler as --- handler\_file\_name.handler\_name

**Note:**

When we create custom handler/function, then make sure that handler/function is present in that file.

**Example**:

let say we have below handler configuration-



According to this configuration-

* Starting/entry point of aws lambda function will be ----my\_lambda\_handler(event, context ) function from lambda\_function.py file.
* Make sure that my\_lambda\_handler() is present in lambda\_function.py file.

**Context in AWS lambda**

Context object provides methods and properties that provide information about the invocation, function, and execution environment

**Creating lambda**

We can create AWS lambda many way as-

1. Using console
2. Using Cloud shell
3. Using CloudFormation **(CFT)**

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html#cfn-lambda-function-ephemeralstorage>

**#############################################**

**# Creating Lambda using CloudFormation #--- Method 3rd**

**#############################################**

While creating lambda using CloudFormation template we can split the lambda creation in two part-

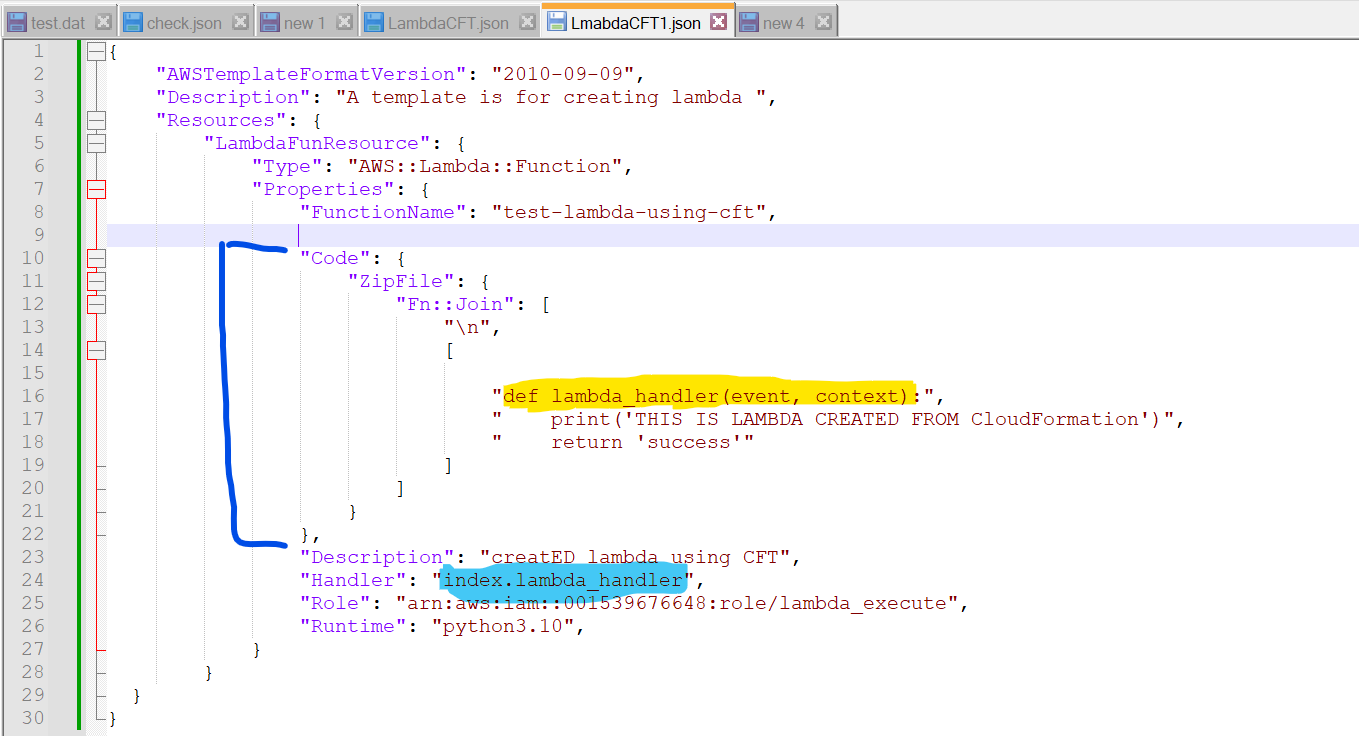
1. When code and dependency are packed in zip file and placed s3 bucket
2. Providing lambda function code in CloudFormation template

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html#cfn-lambda-function-ephemeralstorage>

**Creating Lambda using CFT and code also in template -- Method 3rd (a)**

In below sample template code, it has lambda function code and lambda creation code both.





**Parameter description**

Runtime – programming language with version

Handler – name of file and method which will be entry point of our lambda (file\_name.method\_name).

Default file name – lambda\_function

Default function name – lambda\_handler(always give function name as lambda\_handler)

FunctionName – Name of lambda function by which it will be created

Code --- Code for the function

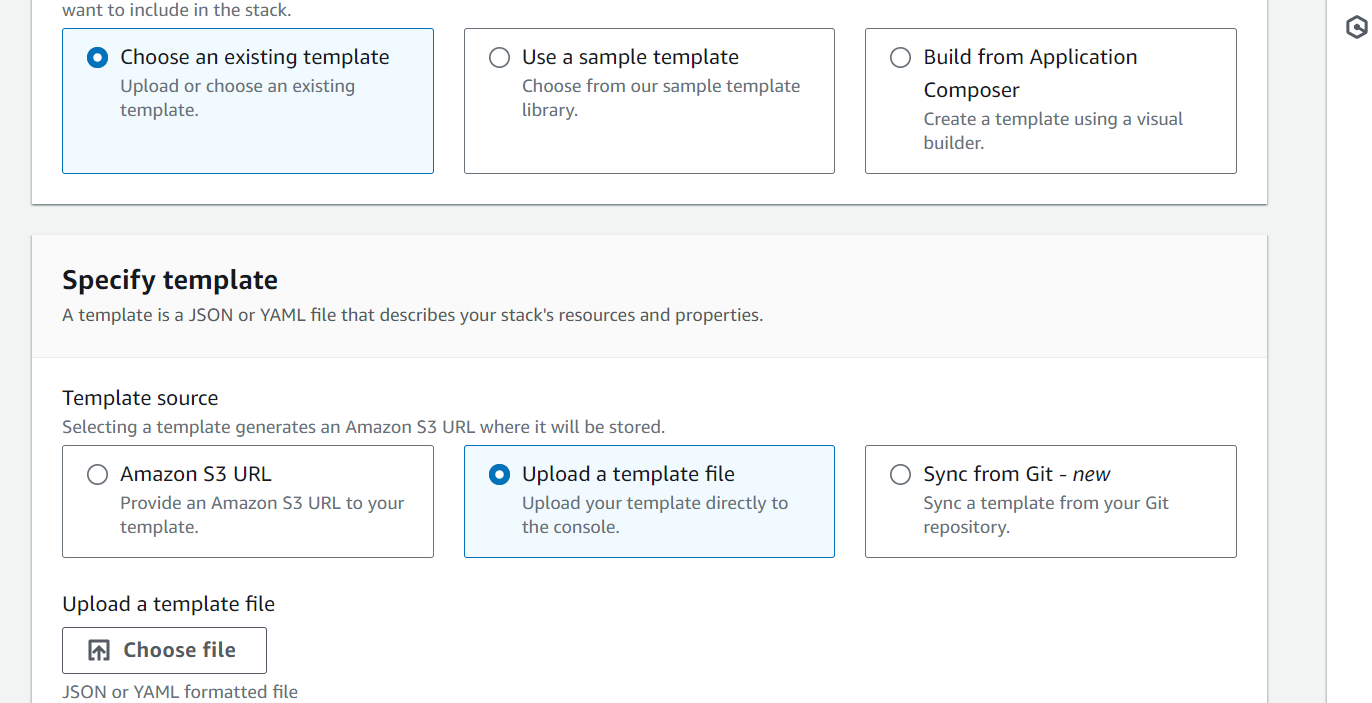
**Note:**

Based on above template file-

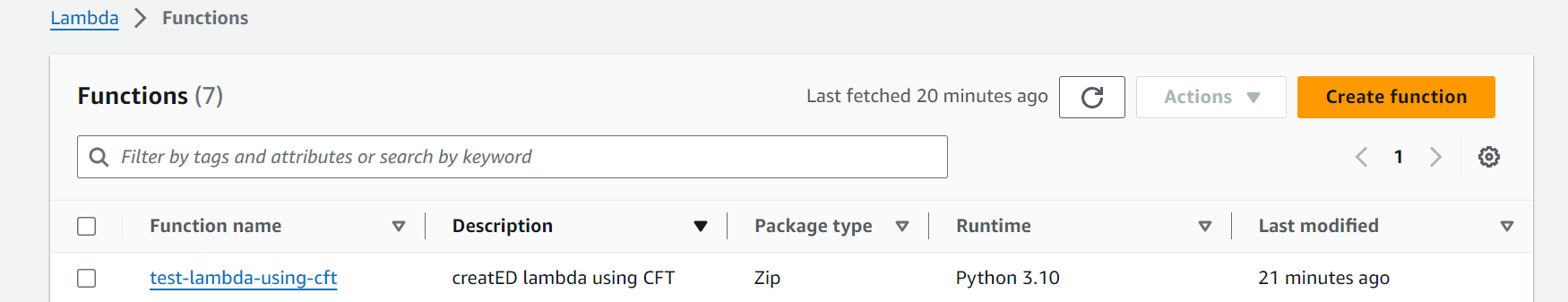
File name will be – index.py

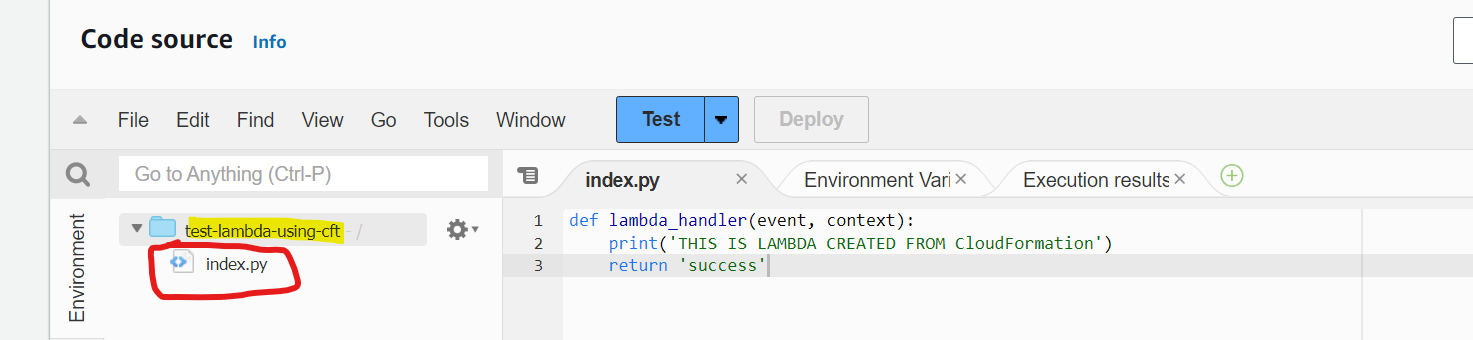
Function/metho name will be – lambda\_handler

Now upload this json file in cloud formation template and lambda function will get created.



Below screenshot shows the lambda created.





**Creating Lambda using CFT (code and dependency packed into zip ) --- Method 3 (b)**

**Step:**

* Create a folder locally on laptop / on cloud shell
* Create the lambda handler file you want, default name lambda\_function.py
* Create the lambda handler function in handler file

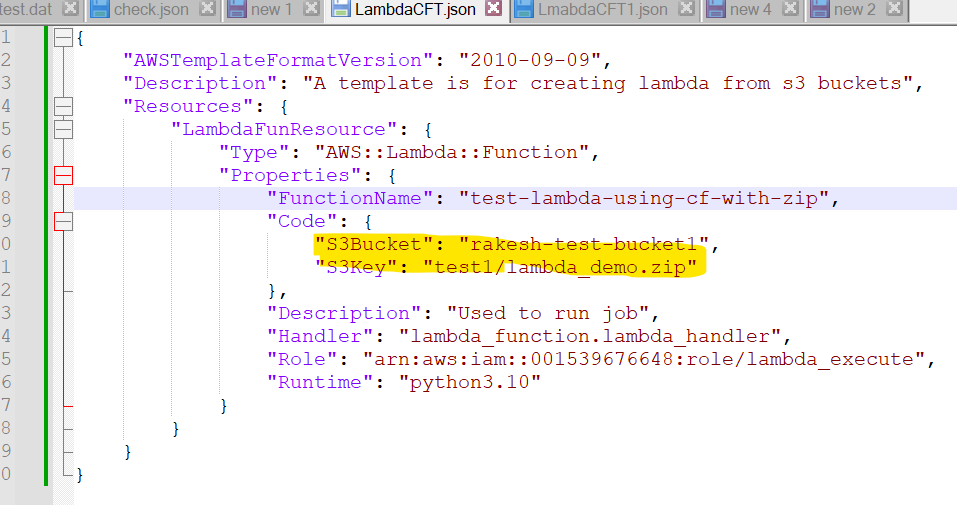
( optional – can be created after lambda is created using console)

* Install reqrered library using pip command (any of below method )

1. pip install --target \_path\_to\_install <library\_name>
2. Go to path where want to install then --- **pip install –target . <library\_name>**

* Zip the file and place that zip file in any bucket
* Now create the CFT json file and upload to CFT to create the lambda

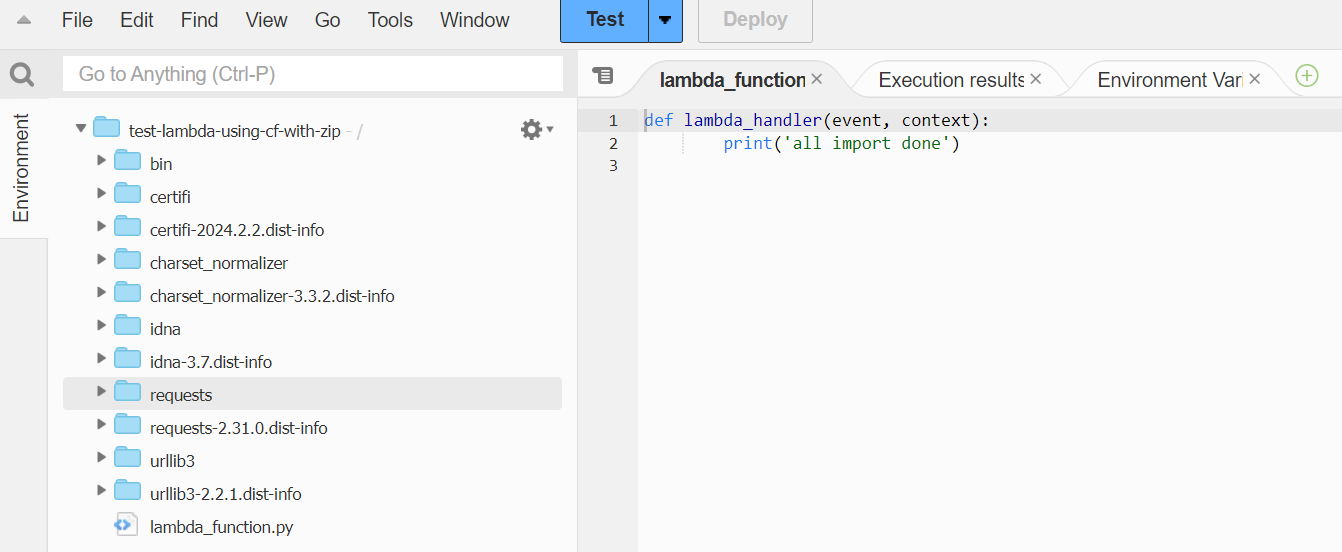




Here for learning purpose, I have zipped lambda\_demo folder which contains some libraries also which have lambda\_handler method in lambda\_function.py file as entry point as usual and placed in my bucket.

Add more detail how and what was in .zip folder and how it was created

Now upload this template and lambda function will be created as below-



**##############################**

**# Creating lambda using clous shell #**

**##############################**

This is exactly same as above one (Method 3rd(b)

**##############################**

**#Calling one lambda from another #**

**##############################**

We can one lambda from another lambda using boto3 library, but before calling we should make sure that calling lambda have access /permission to call another labda.

response = client.invoke( FunctionName='string'| labda arn, InvocationType='Event'|'RequestResponse'|'DryRun', LogType='None'|'Tail', ClientContext='string', Payload=b'bytes'|file | json\_data , Qualifier='string' ) ----- Syntax

**#######################################################################################**

**# AWS CodeBuild #**

**#######################################################################################**

**What is CodeBuild ?**

AWS CodeBuild is a fully managed build service that compiles source code, runs tests/unit tests, and produces software packages that are ready to deploy. With CodeBuild, you don’t need to worry about provisioning and managing your own build infrastructure. You simply provide your build project’s source code and build settings, and CodeBuild handles the rest.

**Some keywords used in CodeBuild**

Below are some common naming terminologies that we used in codebuild.

**Build project**

A build project includes information about how to run a build, including where to get the source code, which build environment to use, which build commands to run, and where to store the build output.

**Build environment**

A build environment represents a combination of operating system, programming language runtime, and tools that CodeBuild uses to run a build

**buildspec.yml**

A buildspec is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build. You can include a buildspec as part of the source code or you can define a buildspec when you create a build project

For build syntax and phased check below weblink-

<https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html#build-spec-ref-name-storage>

**Buildspec file name and storage location**

1. If you include a buildspec as part of the source code, by default, the buildspec file must be named buildspec.yml and placed in the root of your source directory.
2. You can specify only one buildspec for a build project, regardless of the buildspec file's name.
3. You can override the default buildspec file name and location. For example, you can:

* Use a different buildspec file for different builds in the same repository, such as buildspec\_debug.yml and buildspec\_release.yml.

Store a buildspec file somewhere other than the root of your source directory, such as config/buildspec.yml or in an S3 bucket.

The S3 bucket must be in the same AWS Region as your build project. Specify the buildspec file using its ARN (for example, arn:aws:s3:::<my-codebuild-sample2>/buildspec.yml).

Check more on below link-

<https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html>

**#######################################################################################**

**# AWS CodeBuild #**

**#######################################################################################**

**What is CodeDeploy**

* AWS CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.
* CodeDeploy can deploy application content that runs on a server and is stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories.
* CodeDeploy can also deploy a serverless Lambda function. You do not need to make changes to your existing code before you can use CodeDeploy.

<https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html>

**#######################################################################################**

**# boto3 #**

**#######################################################################################**

Boto3 is python library for working with AWS services.

Boto is a software development kit (SDK) designed to improve the use of the Python programming language in Amazon Web Services.

**Client in boto3:**

Clients provide a low-level interface to AWS whose methods map close to 1:1 (each AWS services can be used/accessed using boto3) with service APIs. All service operations are supported by clients.

For each AWS resource we can created client in below way –

**client = boto3.client( service\_name )** ---- Creating boto3 client for any service.

* Typically **yields primitive, non-marshalled data** (e.g. DynamoDB attributes are dicts representing primitive DynamoDB values)

**Resource in boto3:**

Resources represent an object-oriented interface to Amazon Web Services (AWS). They provide a higher-level abstraction than the raw, low-level calls made by service clients. Not all AWS services can be exposed/used by resource.

We can create boto3 resource in below way for supportable AWS resource-

**resource = boto3.resource ( service\_name )** ----- Creating boyo3 resource

* Typically **yields marshalled data, not primitive AWS data** (e.g. DynamoDB attributes are native Python values representing primitive DynamoDB values)

**Paginators**

Some AWS operations return results that are incomplete and require subsequent requests in order to attain the entire result set. In such case we need to use paginator.

Paginator can be **created by using boto3 client or resource as** -

**paginator= client.get\_paginator(operation\_name)**

**paginator= resource.get\_paginator(operation\_name)**

* **operation\_name**

This is the same name as the method name of the client/resource.

If method of client/resource is 'foo()' then it will be accessed like –

* + **resource.get\_paginator(foo)**
  + **clinet.get\_paginator(foo)**

**################################ AWS S3 client #########################################**

For boto3 clinet of S3 object we have below methods-



**Commonly used methods are:**

1. **S3.Client.get\_object(\*\*kwargs)**

Retrieves objects from Amazon S3. To use GET, you must have READ access to the objects.

Return type is python dictionary.

**kwarg** – these are keyword argument **of request format**.

**Request Syntax/Format**

Request/syntax accepts below parameters**-**

response = client.get\_object(

Bucket='string',

IfMatch='string',

IfModifiedSince=datetime(2015, 1, 1),

IfNoneMatch='string',

IfUnmodifiedSince=datetime(2015, 1, 1),

Key='string',

Range='string',

ResponseCacheControl='string',

ResponseContentDisposition='string',

ResponseContentEncoding='string',

ResponseContentLanguage='string',

ResponseContentType='string',

ResponseExpires=datetime(2015, 1, 1),

VersionId='string',

SSECustomerAlgorithm='string',

SSECustomerKey='string',

RequestPayer='requester',

PartNumber=123,

ExpectedBucketOwner='string',

ChecksumMode='ENABLED'

)

**Description of request parameter:**

* **Bucket**=string -- Name of bucket in which file is present
* **Key** = string -- key/full path of file (excluding bucket name) in bucket

{

'Body': StreamingBody(),

'DeleteMarker': True|False,

'AcceptRanges': 'string',

'Expiration': 'string',

'Restore': 'string',

'LastModified': datetime(2015, 1, 1),

'ContentLength': 123,

'ETag': 'string',

'ChecksumCRC32': 'string',

'ChecksumCRC32C': 'string',

'ChecksumSHA1': 'string',

'ChecksumSHA256': 'string',

'MissingMeta': 123,

'VersionId': 'string',

'CacheControl': 'string',

'ContentDisposition': 'string',

'ContentEncoding': 'string',

'ContentLanguage': 'string',

'ContentRange': 'string',

'ContentType': 'string',

'Expires': datetime(2015, 1, 1),

'WebsiteRedirectLocation': 'string',

'ServerSideEncryption': 'AES256'|'aws:kms'|'aws:kms:dsse',

'Metadata': {

'string': 'string'

},

'SSECustomerAlgorithm': 'string',

'SSECustomerKeyMD5': 'string',

'SSEKMSKeyId': 'string',

'BucketKeyEnabled': True|False,

'StorageClass': 'STANDARD'|'REDUCED\_REDUNDANCY'|'STANDARD\_IA'|'ONEZONE\_IA'|'INTELLIGENT\_TIERING'|'GLACIER'|'DEEP\_ARCHIVE'|'OUTPOSTS'|'GLACIER\_IR'|'SNOW'|'EXPRESS\_ONEZONE',

'RequestCharged': 'requester',

'ReplicationStatus': 'COMPLETE'|'PENDING'|'FAILED'|'REPLICA'|'COMPLETED',

'PartsCount': 123,

'TagCount': 123,

'ObjectLockMode': 'GOVERNANCE'|'COMPLIANCE',

'ObjectLockRetainUntilDate': datetime(2015, 1, 1),

'ObjectLockLegalHoldStatus': 'ON'|'OFF'

}

**Response Syntax**

Response is **always a dict type**, having below parameters-

**########################### botocore.response #############################**

class botocore.response.StreamingBody(raw\_stream, content\_length)

Wrapper class for an http response body.

It have some methods to work with StreamingBody data.

Streaming Body class methods are:-

* **read(amt=None)**

Read at most amt bytes from the stream.If the amt argument is omitted, read all data.

* **close()**

Close the underlying http response stream

* **readable()**

Return whether object was opened for reading.If False, read() will raise OSError.

**########################### s3 client methods #############################**

1. **client.get\_paginator(operation\_name)**

Creates a paginator for operation.

Parameters are:

**operation\_name** -- this will be boto3 client method name for which we want to create pagination.

1. client.list\_objects(\*\*kwargs)

Returns some or all (up to 1,000) of the objects in a bucket. You can use the request parameters as selection criteria to return a subset of the objects in a bucket. It’s return type is dictionary.

**Request (\*\*kwargs) parameters are:**

response = client.list\_objects(

Bucket='string',

Delimiter='string',

EncodingType='url',

Marker='string',

MaxKeys=123,

Prefix='string',

RequestPayer='requester',

ExpectedBucketOwner='string',

OptionalObjectAttributes=[

'RestoreStatus',

]

)

**Return type/format:**

This method returns the result in dict format having below parameter-

{

'IsTruncated': True|False,

'Marker': 'string',

'NextMarker': 'string',

'Contents': [

{

'Key': 'string',

'LastModified': datetime(2015, 1, 1),

'ETag': 'string',

'ChecksumAlgorithm': [

'CRC32'|'CRC32C'|'SHA1'|'SHA256',

],

'Size': 123,

'StorageClass': 'STANDARD'|'REDUCED\_REDUNDANCY'|'GLACIER'|'STANDARD\_IA'|'ONEZONE\_IA'|'INTELLIGENT\_TIERING'|'DEEP\_ARCHIVE'|'OUTPOSTS'|'GLACIER\_IR'|'SNOW',

'Owner': {

'DisplayName': 'string',

'ID': 'string'

},

'RestoreStatus': {

'IsRestoreInProgress': True|False,

'RestoreExpiryDate': datetime(2015, 1, 1)

}

},

],

'Name': 'string',

'Prefix': 'string',

'Delimiter': 'string',

'MaxKeys': 123,

'CommonPrefixes': [

{

'Prefix': 'string'

},

],

'EncodingType': 'url',

'RequestCharged': 'requester'

}

**Contents (list)**

Metadata about each object returned. it will be list of dictionary-

An object consists of data and its descriptive metadata.

* + Key (string) –

The name that you assign to an object. You use the object key to retrieve the object.

* + LastModified (datetime) –

Creation date of the object.

1. **clinet.list\_objects\_v2(\*\*kwargs)**

this is exactly same as list\_objects(\*\*kwargs) expect list\_objects\_v2(\*\*kwargs) takes some addition parameter in input-

|  |  |
| --- | --- |
| list\_objects\_v2 | list\_objects |
| response = client.list\_objects\_v2(  Bucket='string',  Delimiter='string',  EncodingType='url',  MaxKeys=123,  Prefix='string',  # Replace marker to list continuous page  ContinuationToken='string',  # set to True to fetch key owner info. Default is False.  FetchOwner=True|False,  # This is similar to the Marker in list\_object()  StartAfter='string'  ) | response = client.list\_objects(  Bucket='string',  Delimiter='string',  EncodingType='url',  #Marker to list continuous page  Marker='string',  MaxKeys=123,  Prefix='string'  ) |

**#####################################################################################**

**########################### boto3 AWS Lambda #######################################**

**#####################################################################################**

Using the aws boto3 library is same as other service (S3, Glue etc). To work with lambda using boto3 library we need to first create the client object same as other.

lambda\_client = boto3.client( 'lambda' ) ------------ creating client object for lambda using boto3

**Methods of boto3 lambda's client object-**



**response = client.invoke( FunctionName='string'| labda arn, InvocationType='Event'|'RequestResponse'|'DryRun', LogType='None'|'Tail', ClientContext='string', Payload=b'bytes'|file | json\_data , Qualifier='string' )**

This function is used to **Invoke a Lambda function**. **This operation requires permission for the lambda:InvokeFunction action**

Response of this method is dict type.

**Function parameters are:**

* **FunctionName**: Lambda functioan name/ARN to invoke. ( mandatory )
* **InvocationType**: string), acceptable value - RequestResponse, Event , DryRun
* **Payload**: bytes or seekable file-like object or json object, Input/Event to sent to called function

**Return/Response type of this method is dict type.**

**Response Syntax**

{

'StatusCode': 123,

'FunctionError': 'string',

'LogResult': 'string',

'Payload': StreamingBody(),

'ExecutedVersion': 'string'

}

'StatusCode' ------ Ineteger type

'Payload' ---- this is the data return by called function. Type of this will be StreamingBody.

**Example**:

Let say we have one lambda – "test\_labda\_2" which is suppose to call "test\_lambda\_end". Write lambda code in test\_labda\_2 to call test\_lambda\_end and print the data return by test\_lambda\_end.

**Solution:**

**# test\_lambda\_2 code #**

import json

import boto3

#Using this lambda and boto3 library

#will call another lambda

def lambda\_handler(event, context):

    print('test\_lambda2 started...')

    print('even is : ',event)

    print('data types of event is: ',type(event))

    client=boto3.client('lambda')

    #Now try to invoke the lambda

    lambda\_ARN = 'arn:aws:lambda:ap-south-1:001539676648:function:test\_lambda\_end'

    event\_to\_send=json.dumps({'key1':'value1'})

    response=client.invoke(FunctionName=lambda\_ARN,InvocationType='RequestResponse',Payload=event\_to\_send)

    print('type of repsonse is: ',type(response)) #dict

    print('payload is: ',response['Payload'])     #<botocore.response.StreamingBody

    print('value of payload is: ',response['Payload'].read()) #{"return data": "test\_lambda\_end comppleted"}

#test\_lambda\_end #

import json

def lambda\_handler(event, context):

    # TODO implement

    print('started test\_lambda\_end with below details: ')

    print('event is: ',event) # event is: {'key1': 'value1'}

    print()

    print('context is: ',context)

    return {'return data' :"test\_lambda\_end comppleted"}