AWS Lambda

Traditional way of deploying servers was to create and manage servers and deploy those applications on those servers with lambda we just have to not maintain any servers but rather just focus on the application code.

AWS manages the underlying compute resources like storage

You can define specific functions

-> Triggered by events - Ex: When there is an object in the S3 bucket execute this function

-> Request/Response - When you expect a response from the request Ex : When integrating with AWS API Gateway

Supported Languages

node.js

Python

Java

C#

Ruby

The code in the lambda functions can be created in any of these languages.

Why use Lambda :

--> Serverless computing - It eliminates the need to manage the infrastruture so you just need to create the function and Amazon can handle everything.

--> Function as a service - We can build funtionality without the complexity of building and managing the infrastructure.

--> Scalable capacity - It can dynamically scale capacity in response to increased traffic.

--> Pay per use -

--> Event driver workflows - You can say when you upload the file to execute the function

--> Integration with other AWS Products - We can use API gateway to provide a secure and scalable gateway for web apis that routes http request to Lambda functions or we can define S3 events that invoke lambda functions.

We can also use Lambda functions to process the SQS messages

Amazon S3, DynamoDB, Amazon SNS, CloudTrail, CloudFront

AWS Lambda Costs

- Amount of requests

- Duration for Work undertaken

Free Tier :

First 1M free requests per month with 400,000 GB - seconds compute

AWS Lambda Limitations :

- Max Execution Duration < 15 min

- Memory Allocation 128MB -3008 MB

You cannot allocate CPU to your Lambda functions unlike in EC2 but available cpu for your lambda function will be allocated.

- Ephemeral Storage < 500MB - The capacity which is under /tmp folder

- Compressed function package < 50 MB - Deployment package size which is 250 MB for unzipped and 50MB for Zip

- Function package in a region < 75 GB

Lambda Scenarios

How Lambda works:

-> First upload AWS Lambda code in any supported language.

-> Lambda can run your functions when triggered by events from different AWS services.

-> Lambda executes the code when it is triggered and executes the code to a container.

-> When a function is created lambda packages it into a new container and then executes the container. You will be charged only when the code executes.

Popular Triggers that can happen

- File change in S3 bucket

- Update Dynamo DB tables

- Push notifications

- Email sending

- Hosting a website

In a Serverless model user needs to - Organize code into lambda functions

- Each lambda function runs in its own container

- Each container is allocated its necessary RAM and CPU.

- Invoked with lambda API or in response to events from AWS service.

Lambda function structure

---> It starts with Handler- entry point, accepts JSON-formatted input

Ex :

def handler\_name (event,context)

### your code

return result;

event - events has details of the event

context - context provides runtime information to handler

---> Trigger : Invoked or triggered everytime there is an upload to S3 or DynamoDB table change.

Respond to request to API gateway or based on AWS Cloudwatch trigger

Example :

def lambda\_handler (event, context):

print(event)

return 'Hello from me'

Create Lambda function

From Scratch

Use existing Blueprint

Container Images - You can package lambda function code and dependencies as a container image rather than updating as a zip so when triggered the functions deployed as container images are run as is.

Python example for parsing a response in this case it will show the region.

When we use this function script below :

import json

import os

def lambda\_handler(event, context):

json\_region = os.environ['AWS\_REGION']

return {

"statusCode": 200,

"headers": {

"Content-Type": "application/json"

},

"body": json.dumps({

"Region ": json\_region

})

}

When we invoke the below data

{

"key1": "value1",

"key2": "value2",

"key3": "value3"

}

We get the below response

{

"statusCode": 200,

"headers": {

"Content-Type": "application/json"

},

"body": "{\"Region \": \"us-east-1\"}"

}

200 OK: the request was successful.

Content-Type - Indicates the content type that is used in the body of the request.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated