LAMBDA

compute = processing power (CPU+RAM+Storage)

Server/Instance - EC2

- OS
- Security
- The application
- Infrastructure (CPU, memory, network, storage)

-> 'Server- based Computing'

-> AWS Lambda is a serverless compute service.

-> do not need to manage any infrastructure/servers, AWS will automatically manage them for you.

-> You write & run a code to let AWS manage the server infrastructure.

-> Serverless Compute service allows developers to build & run the application w/o handling the server's directly.

-> Using AWS Lambda, every resources is created by writing a code, but the infrastructure is managed by AWS itself.

-> AWS Lambda is used to run the codes w/o managing the servers. It is a serverless computer service, that provides a platform to run the programs.

-> It supports multiple programming languages.

Difference b/w EC2 & Lambda

EC2	Lambda
-> Server-based compute service.	-> Serverless compute service.
-> You need to create & manage the servers.	-> No need to manage the servers OR to interact with the service portal.
-> You pay on the basis of running instances and attached resources.	-> You have to pay only for the computation.

Functions:

-> A piece of code that performs a specific task/operation.

-> To perform a task in AWS Lambda, you need to create a function, inside which you will write the code.

```
num1 = 10
num2 = 5
vsum = num1+num2
print(sum)
```

```
def add(a,b,c):
    return a+b+c

print(add(4,7,1))
```

Types of Functions:

1. Event Source (Triggers)
2. Destination (Actions)

1. Event Source

-> Event: It is an input that triggers the execution of a Lambda function.

-> Lambda is a serverless comput service. You write the code & upload to AWS, and it automatically runs the code 'when it is triggered by an event'.

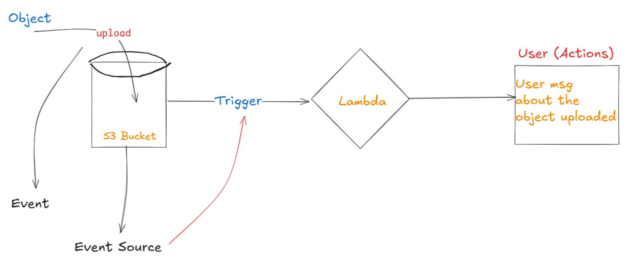
-> Event source: It is a type of service/request that triggers Lambda.

Types of Events:

1. Push-based event:

-> In push-based event, the event source actively triggers the Lambda function, whenever an occurs.

Here, whenever something occurs within the source, the event source will trigger the Lambda.

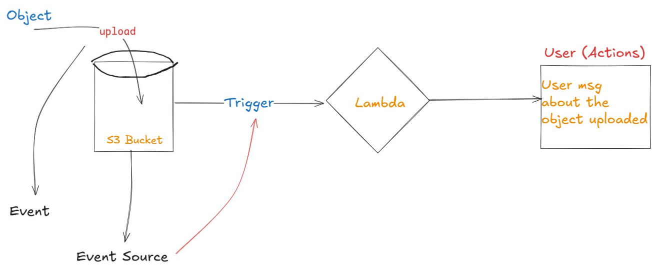


Types of Events:

1. Push-based event:

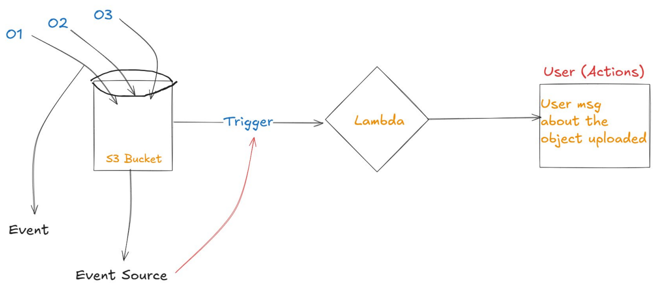
-> In push-based event, the event source actively triggers the Lambda function, whenever an occurs.

Here, whenever something occurs within the source, the event source will trigger the Lambda.



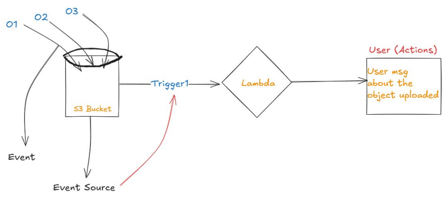
Types of Push-based event:

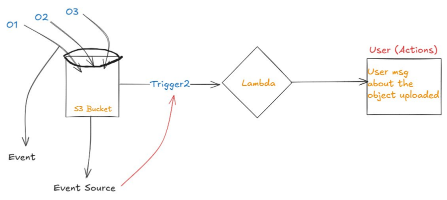
i. Synchronous Push-based event:



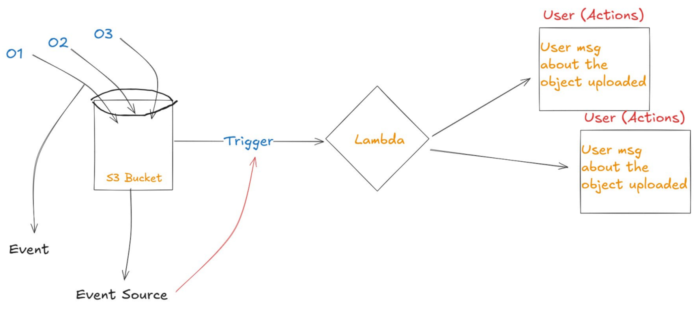
-> here, Lambda will take the requests of triggers, one-by-one & after only the 1st process if completed, then only it will request the trigger again.

-> The event source will wait for the Lambda function to execute & return the response, before continuing.





ii. Asynchronous Push-based event:



-> THe event source sends the request to Lambda & does not wait for the function completely and Lambda processed these event independently.

-> It will not wait for the whole process to complete. Here, each of the event will have their own triggers.

Steps to manage EC2 instances with the help of Lambda

1. Go to EC2 and create 2 instances and keep them in running state.
2. Go to IAM & select 'Roles' and then click on 'Create Roles'.
3. From Trusted entity type, select 'AWS Service' & in 'USe case', select 'Lambda'.
4. Under 'Permission policies', search for 2 permissions:
    i. AmazonEC2FullAccess
    ii. EC2InstanceConnect
5. Click on Next & then provide a a name to the role and click on 'Create Role'.
6. Search for 'Lambda' service and click on it.
7. Click on 'Create a function' & select 'Author from scratch' option.
8. Under 'basic information', give a name to function and select the runtime as 'Python'.

- Create a python script to list all the running instances.
- Create a python script to stop all the running instances.
- Create a python script to send an email if an object is uploaded in S3 bucket.

```
import json
 import boto3

def lambda_handler(event, context):
        ec2 = boto3.client('ec2')

        response = ec2.describe_instances()

        instances_info=[]

        for resevation in response['Reservations']:

                    for instance in resevation['Instances']:

                     info = {

                            "InstanceID": instance['InstanceId'],
                              "State": instance['State']['Name']

                       }

                    print(f"Instance ID: {info['InstanceID']} |
                    State: {info['State']}")

                     instances_info.append(info)

           return {
                 "Instances": instances_info
             }
```