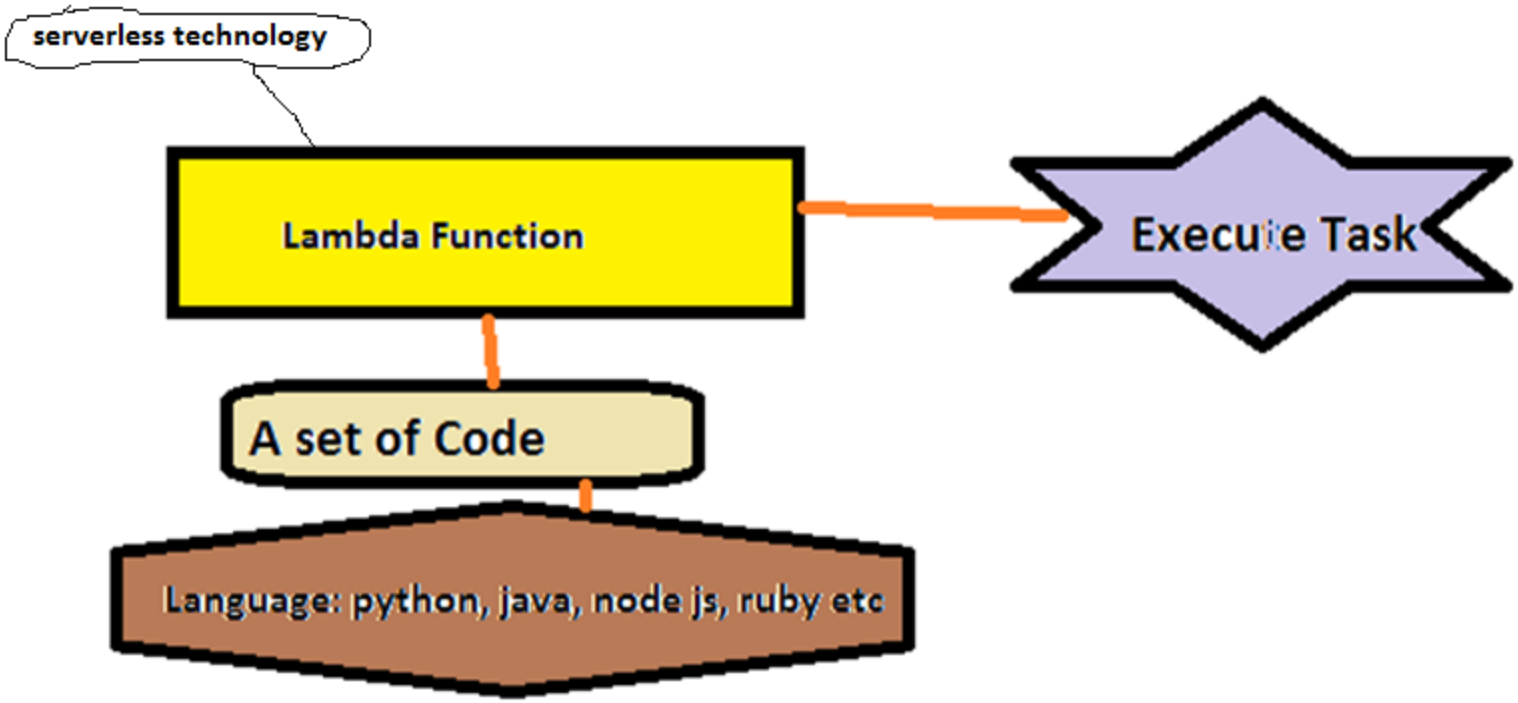
AWS Lambda


serverless technology
Lambda Function
Execute Task
A set of Code
Language: python, java, node js, ruby etc

Topics to be covered--Lambda

1) Introduction to serverless technology - Lambda

2) Create two Lambda function to start and stop EC2 Instance

3) Create three Lambda function to start, stop and terminate EC2 Instance and Add with Cloudwatch to trigger it after a fixed duration of creation—Additionally add SNS topic to get notification

4) Create Lambda function to start and stop EC2 Instance and Add with Cloudwatch ---use UTC time

5)) Create Lambda function to get trigger from s3 bucket and store the data in dynamodb.

What is AWS Lambda

- AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner.

- The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services.

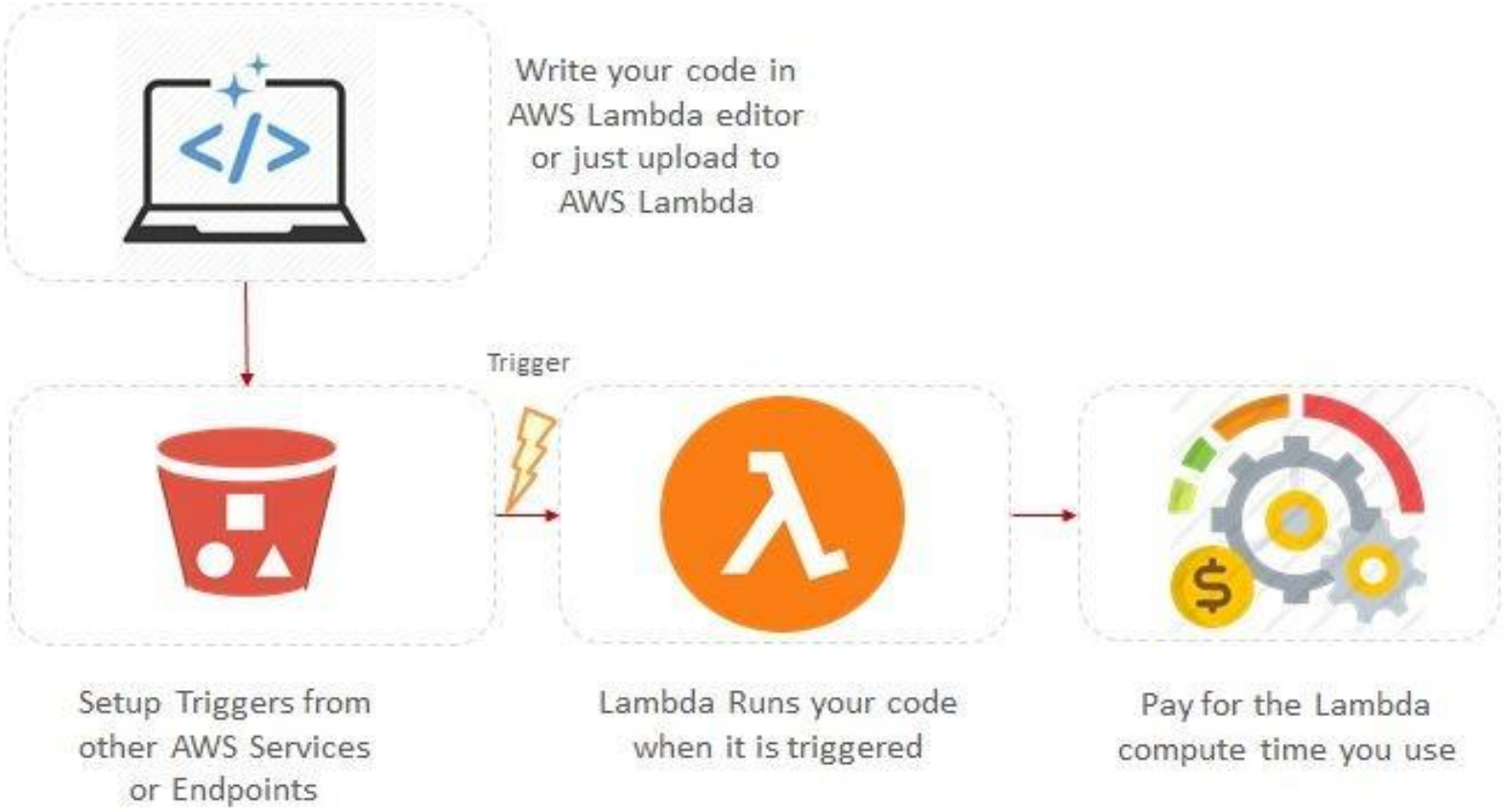
- The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of, so that you can focus on writing application code.

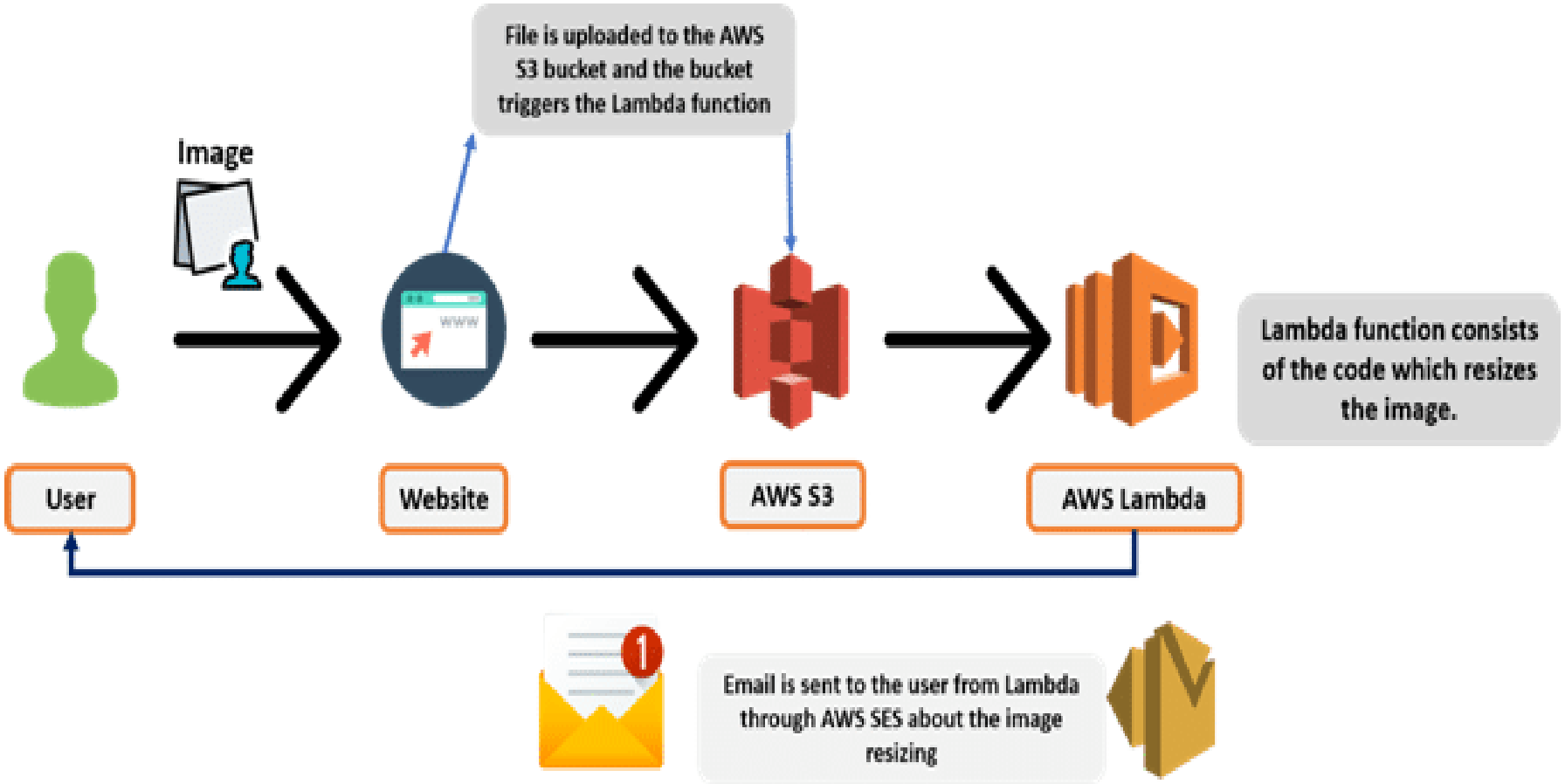
How does AWS Lambda work

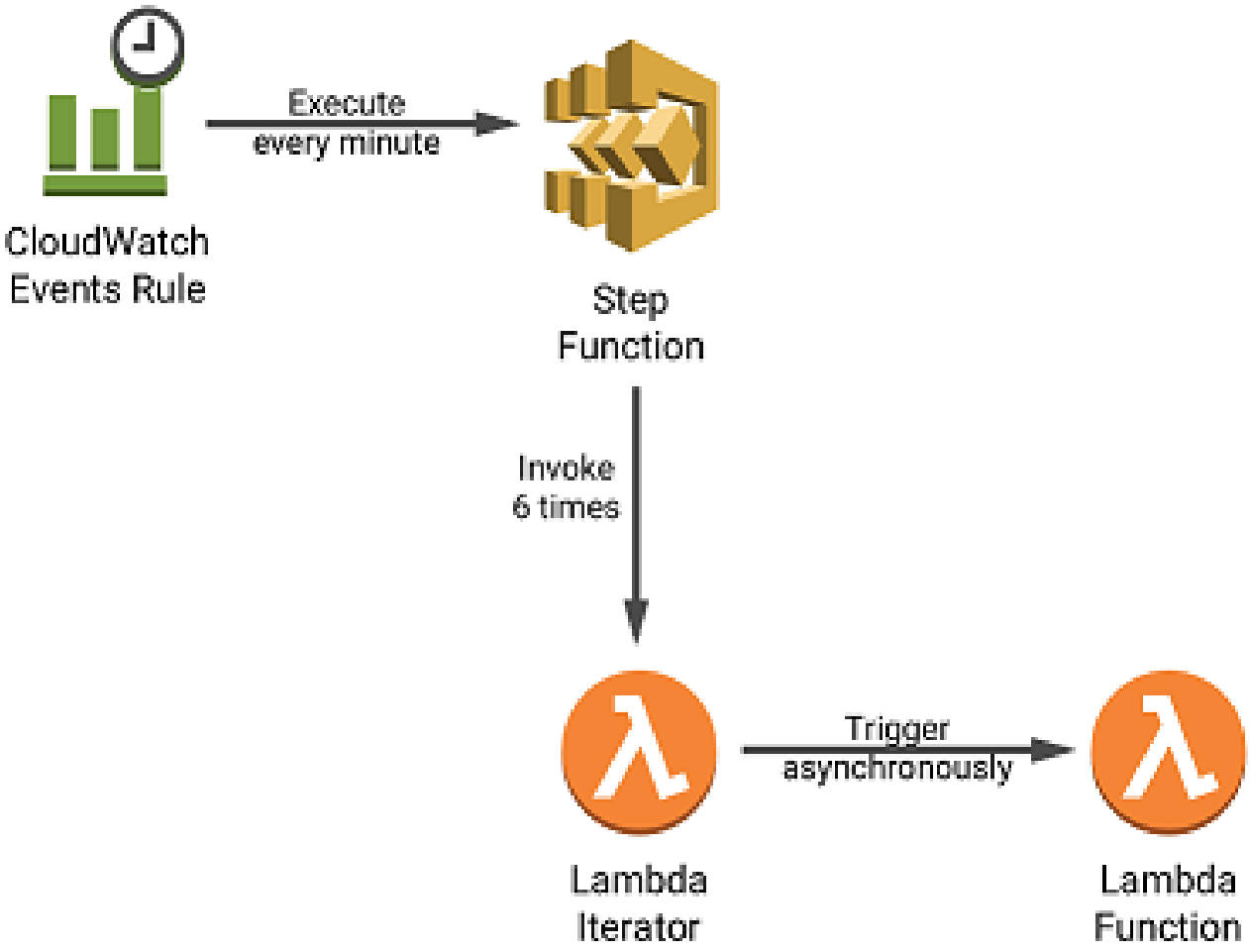
Each Lambda function runs in its own container. When a function is created, Lambda packages it into a new container and then executes that container on a multi-tenant cluster of machines managed by AWS. Before the functions start running, each function's container is allocated its necessary RAM and CPU capacity. Once the functions finish running, the RAM allocated at the beginning is multiplied by the amount of time the function spent running. The customers then get charged based on the allocated memory and the amount of run time the function took to complete.

One of the distinctive architectural properties of AWS Lambda is that many instances of the same function, or of different functions from the same AWS account, can be executed concurrently. Moreover, the concurrency can vary according to the time of day or the day of the week, and such variation makes no difference to Lambda—you only get charged for the compute your functions use. This makes AWS Lambda a good fit for deploying highly scalable cloud computing solutions.

How Lambda Works ???




File is uploaded to the AWS S3 bucket and the bucket triggers the Lambda function
Image
www
Lambda function consists of the code which resizes the image.
User
Website
AWS S3
AWS Lambda
1
Email is sent to the user from Lambda through AWS SES about the image resizing


Execute
every minute
CloudWatch
Events Rule
Step
Function
Invoke
6 times
Trigger
asynchronously
Lambda
Iterator
Lambda
Function

Supported languages and runtimes

- ✓ Node.js
- ✓ Python
- ✓ Ruby
- ✓ Java
- ✓ C#
- ✓ PowerShell

Most common use cases for AWS Lambda

- **Scalable APIs.** When building APIs using AWS Lambda, one execution of a Lambda function can serve a single HTTP request. Different parts of the API can be routed to different Lambda functions via Amazon API Gateway. AWS Lambda automatically scales individual functions according to the demand for them, so different parts of your API can scale differently according to current usage levels. This allows for cost-effective and flexible API setups.
- **Data processing.** Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with datasources like Amazon DynamoDB and trigger a Lambda function for specific kinds of data events. For example, you could employ Lambda to do some work every time an item in DynamoDB is created or updated, thus making it a good fit for things like notifications, counters and analytics.
- **Task automation.** With its event-driven model and flexibility, AWS Lambda is a great fit for automating various business tasks that don’t require an entire server at all times. This might include running scheduled jobs that perform cleanup in your infrastructure, processing data from forms submitted on your website, or moving data around between different datastores on demand.

Benefits of using AWS Lambda

- **Pay per use.** In AWS Lambda, you pay only for the compute your functions use, plus any network traffic generated. For workloads that scale significantly according to time of day, this type of billing is generally more cost-effective.
- **Fully managed infrastructure.** Now that your functions run on the managed AWS infrastructure, you don't need to think about the underlying servers—AWS takes care of this for you. This can result in significant savings on operational tasks such as upgrading the operating system or managing the network layer.
- **Automatic scaling.** AWS Lambda creates the instances of your function as they are requested. There is no pre-scaled pool, no scale levels to worry about, no settings to tune—and at the same time your functions are available whenever the load increases or decreases. You only pay for each function's run time.
- **Tight integration with other AWS products.** AWS Lambda integrates with services like DynamoDB, S3 and API Gateway, allowing you to build functionally complete applications within your Lambda functions.

Example cost calculations

- **Scheduled jobs.** Imagine that you run cron jobs in your AWS infrastructure that perform database rollovers. The job runs every night and the average run time is 10 minutes. The function uses 1GB of memory while running. There is no outbound network traffic generated by the function as it connects to your RDS database in the same availability zone.

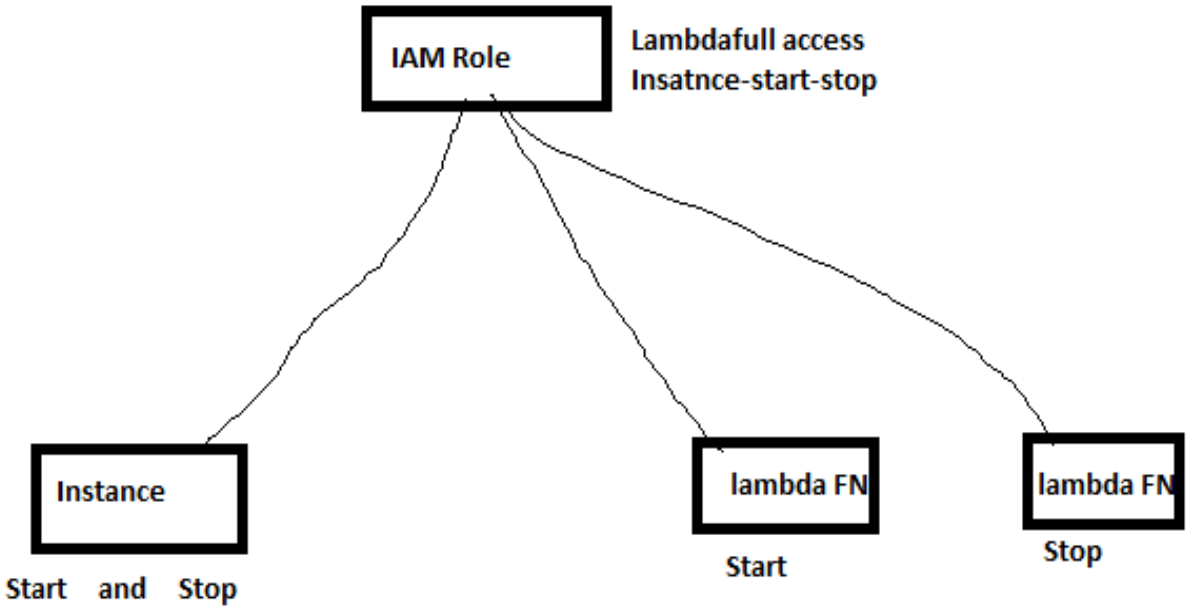
- Total compute: 30 days x 600 GB-seconds (10 minutes) = 18,000 GB-seconds
  Total requests: 30 days x 1 request per day = 30 requests

- This usage is within the AWS free tier, so you'll be charged $0 for AWS Lambda.

- **Note:** You'll still pay for the RDS usage according to the RDS price list.

Lambda Lab --1

1) Create two Lambda function to start and stop EC2 Instance



Lab—1--Steps

1) IAM – open IAM –Create a custom policy and add ec2 start and stop permission.

2) IAM – create a role—select lambda –next –select instance start stop policy – next—create

3) Open google –type ec2 instance start stop lambda –open first link and copy start and stop python code.

4) Open Lambda –new Function – Name :start-instance—Select runtime : Python –select existing lambda role –next –scroll down –Paste the start instance python code –Modify region and instance id. –save

5) In the same way create stop instance function.

6) To check –select created lambda function –test—write some name:test1—create—test -----Now open EC2 console and check the instance

Lab –1 --Code

1) To start Instance

```
import boto3
region = 'us-west-1'
instances = ['i-12345cb6de4f78g9h', 'i-08ce9b2d7eccf6d26']
ec2 = boto3.client('ec2', region_name=region)

def lambda_handler(event, context):
    ec2.start_instances(InstanceIds=instances)
    print('started your instances: ' + str(instances))
```

Lab –1 --Code

1) To stop Instance

```
import boto3
region = 'us-west-1'
instances = ['i-12345cb6de4f78g9h', 'i-08ce9b2d7eccf6d26']
ec2 = boto3.client('ec2', region_name=region)

def lambda_handler(event, context):
    ec2.stop_instances(InstanceIds=instances)
    print('stopped your instances: ' + str(instances))
```

Lab –1 --Code

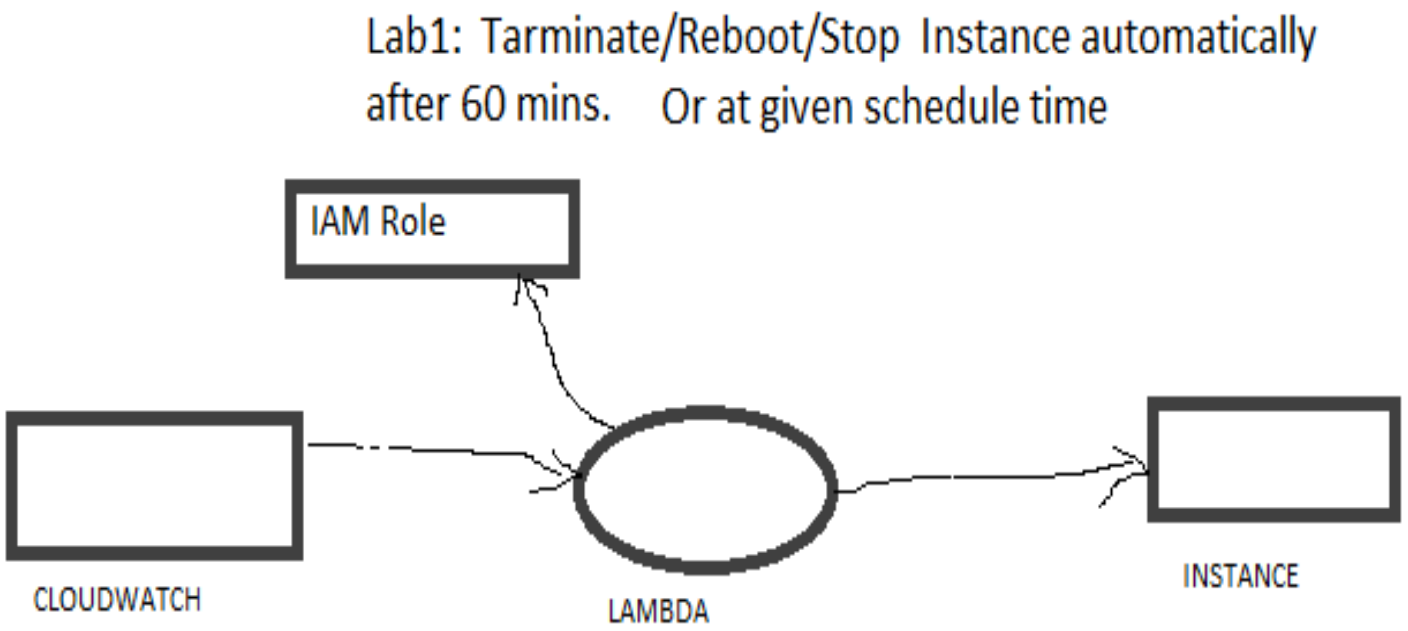
1) To Terminate Instance

```
import boto3
region = 'us-west-1'
instances = ['i-12345cb6de4f78g9h', 'i-08ce9b2d7eccf6d26']
ec2 = boto3.client('ec2', region_name=region)

def lambda_handler(event, context):
    ec2.terminate_instances(InstanceIds=instances)
    print('terminated your instances: ' + str(instances))
```

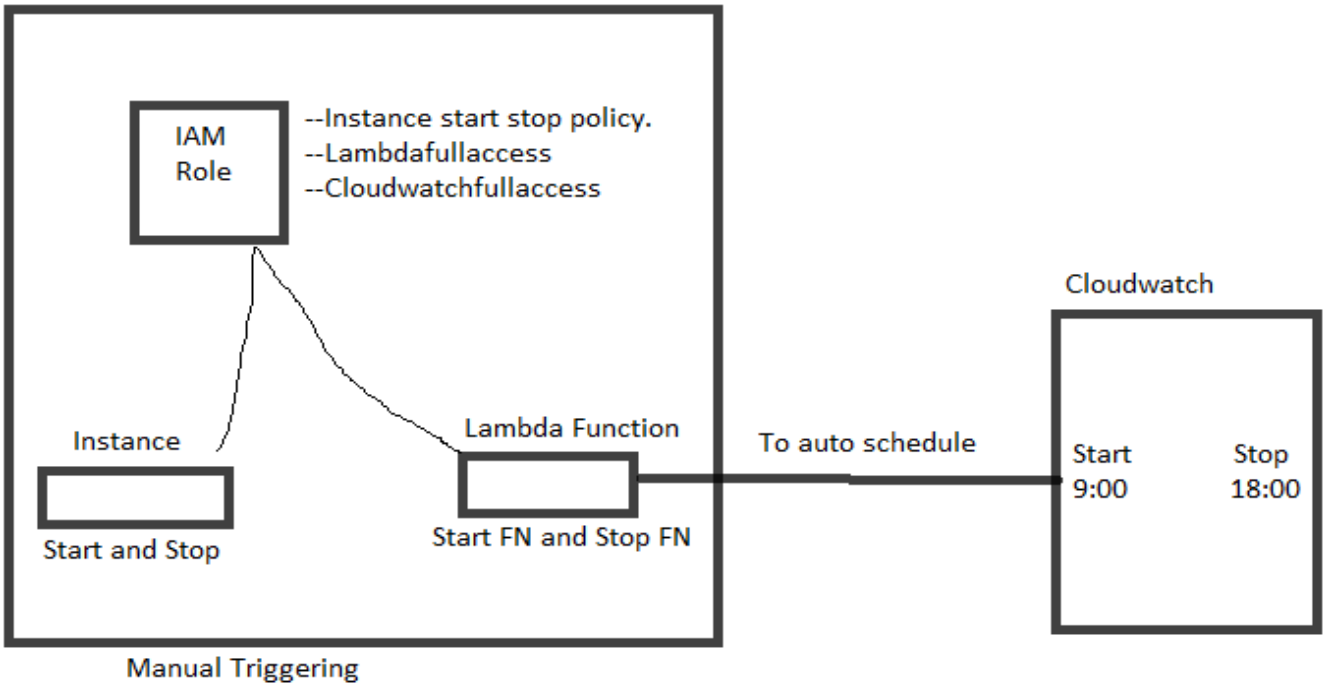
Lambda Lab --2

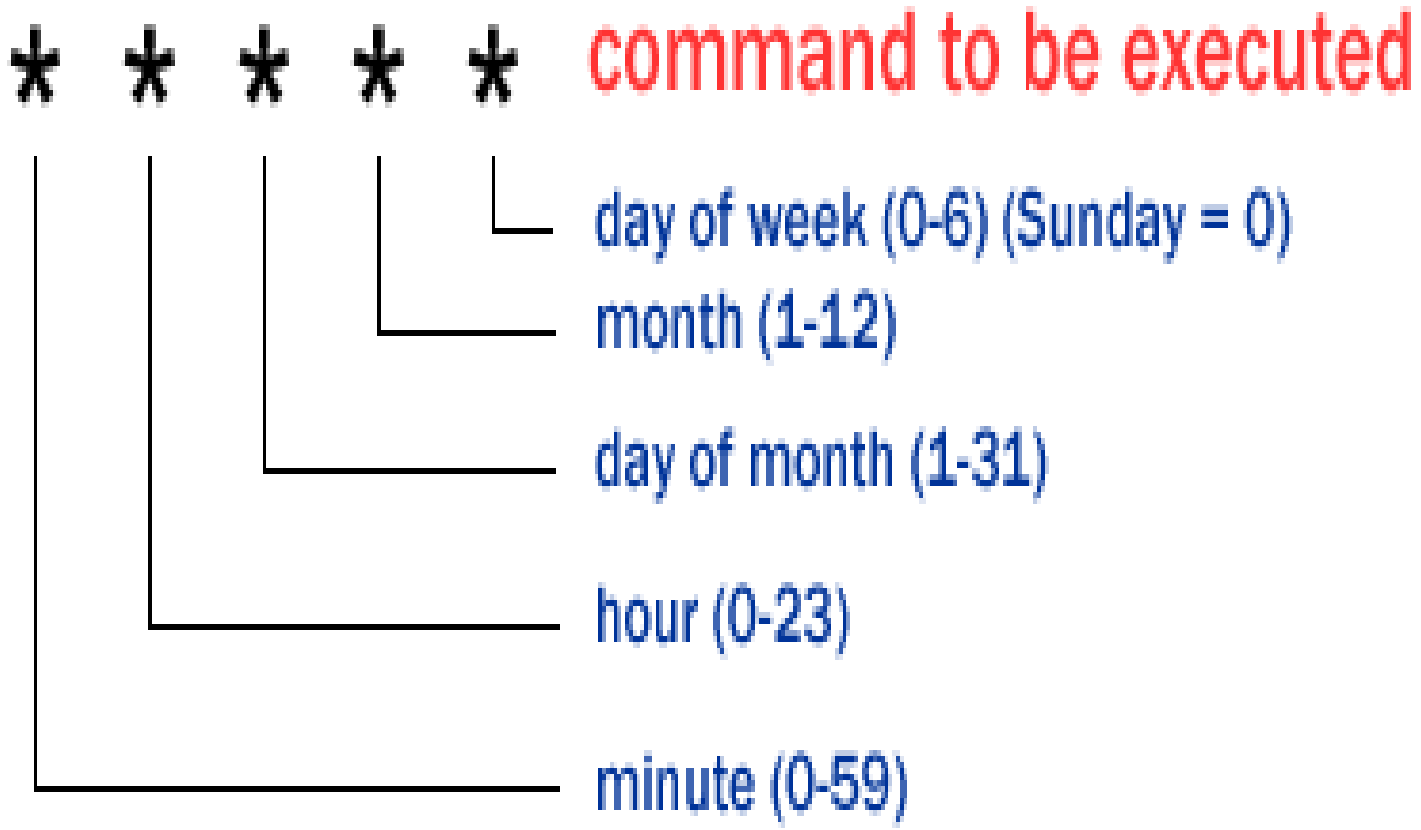
2) Create three Lambda function to start, stop and terminate EC2 Instance and Add with Cloudwatch—Additionally add SNS topic to get notification



Lambda Lab --3

3) Create Lambda function to start and stop EC2 Instance and Add with Cloudwatch ---use UTC time




* * * * * command to be executed
day of week (0-6) (Sunday = 0)
month (1-12)
day of month (1-31)
hour (0-23)
minute (0-59)

Step 1: Create rule

Create rules to invoke Targets based on Events happening in your AWS environment.

Event Source

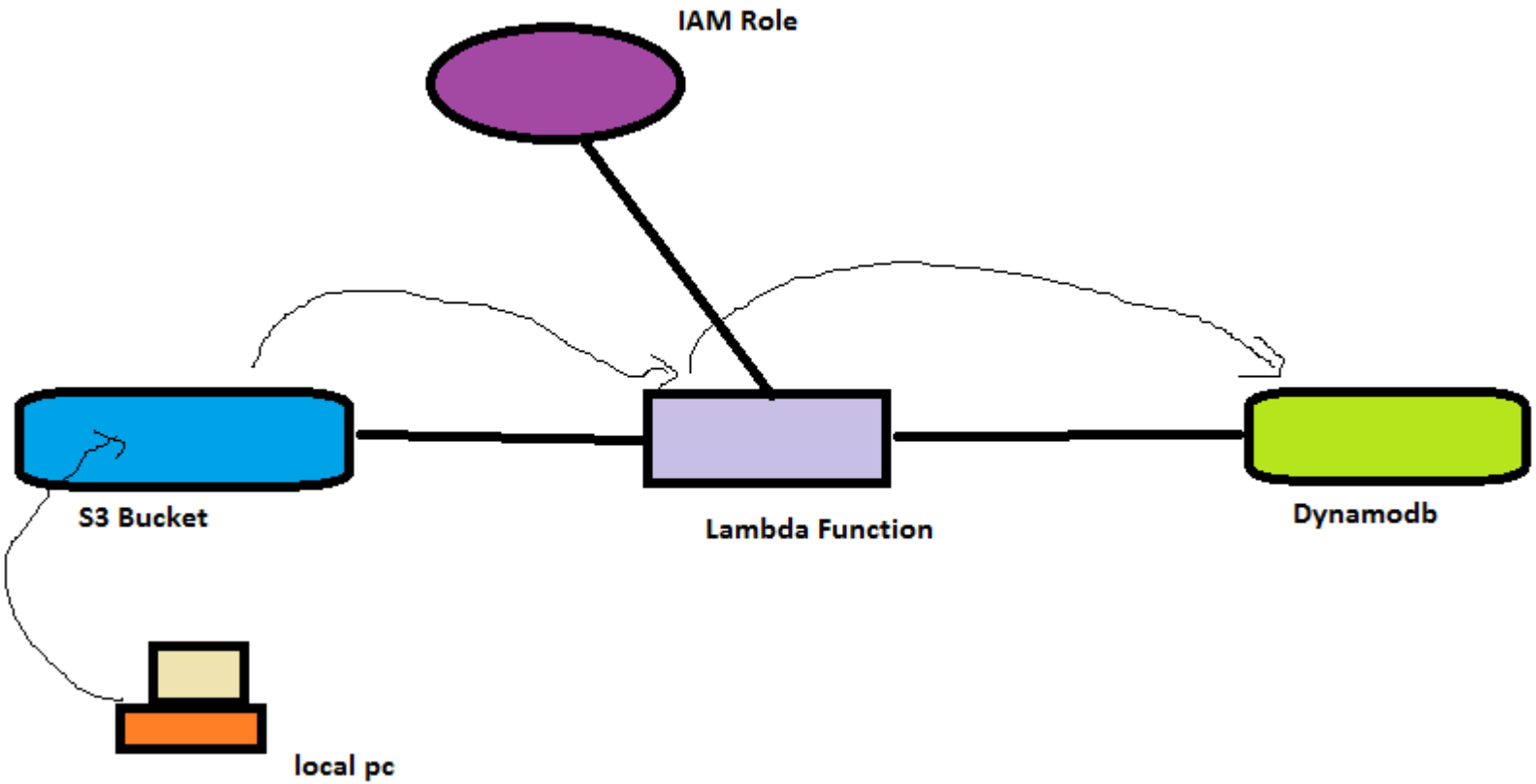
Build or customize an Event Pattern or set a Schedule to invoke Targets.

▸ Show sample event(s)

*** Required**

Lambda Lab --4

4) Create Lambda function to get trigger from s3 bucket and store the data in dynamodb.



Lambda Lab –4 --Steps

4) Create Lambda function to get trigger from s3 bucket and store the data in dynamodb.

1) Create one IAM role having Dynamodbfull access role.

2) Create one S3 bucket

4) Open Lambda –new Function –Author from scratch – Name: s3todynamodb--- Select runtime: Python 3.6 –select existing lambda role : iamrole–next –scroll down –Paste the python code --Configuration--Add Trigger --S3 --fill detail – save ---save

4) Create Dynamodb table with name: newtable and Primary key-Unique --Go to items

5) Upload some files in S3 and check in Dynamodb

Lambda Lab –4--Code

```
import boto3
from uuid import uuid4
def lambda_handler(event, context):
    s3 = boto3.client("s3")
    dynamodb = boto3.resource('dynamodb')
    for record in event['Records']:
        bucket_name = record['s3']['bucket']['name']
        object_key = record['s3']['object']['key']
        size = record['s3']['object'].get('size', -1)
        event_name = record ['eventName']
        event_time = record['eventTime']
        dynamoTable = dynamodb.Table('newtable')
        dynamoTable.put_item(
            Item={'unique': str(uuid4()), 'Bucket': bucket_name, 'Object':
object_key,'Size': size, 'Event': event_name, 'EventTime': event_time})
```