AWS Lambda

Analytics Tensor
Mahesh KC
mahesh.kc@analyticstensor.com
https://analyticstensor.com

AWS Lambda

- AWS Lambda is a serverless infrastructure. Serverless mean we don't need to launch the EC2 instance and deploy the code in order to get out function to be executed.
- AWS Lambda is a fully managed services that take care of all the infrastructure.
- We just have to create or upload function in lambda via:
  - Inline editor in Lambda
  - Upload zip file containing all the code
- The pay is based on compute time per 100ms.
- The code written in Lambda is known as Lambda Function.
- Once the function is on Lambda we can invoke those functions:
  - Manually
  - Through API
  - CLI
  - Events: Create the event on different services such as CloudWatch, S3 etc.
- 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 with 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 cost is based on the allocated memory and the amount of run time the function took to complete.
- The entire infrastructure layer of AWS Lambda is managed by AWS. We don't get visibility into how the system operated and don't need to worry about underlying machine, network connection etc.

AWS Lambda (cont.)

- AWS Lambda is used when building serverless application. AWS Lambda is one of the resources for running the application code.
- It can be integrated with other AWS services such as S3, DynamoDB, RDS, API Gateway etc. for creating serverless application.
- Some of the use case for AWS Lambda are:
  - Scalable APIs: When we are building APIs using AWS Lambda, one execution of Lambda function can server a single HTTP request. Different API can be routed to different Lambda functions via Amazon API Gateway.
  - Data Processing: It is used for event-based data processing. For example, if some event happened on S3 or DynamoDB, we ca trigger Lambda functions for specific kinds of data events.
  - Task Automation: Since it is event-driven mode, it fit for automation of various tasks. We don't have to need server every time. Some examples includes: running schedule job for cleaning infrastructure, processing data when forms are submitted in the website, moving data between different datastores on demand.
- Lambda support most of the popular programming languages and runtimes such as: Node.js, Python, Java, Go, C#, PowerShell etc.
- The benefit of AWS Lambda are:
  - Pay per use:
  - Fully managed infrastructure:
  - Automatic scaling
  - Integration with other AWS services:

AWS Lambda (cont.)

- Limitation of AWS Lambda:
  - Execution time/run time: Lambda function will timeout after running for 15 minutes. We can't change the limit. Lambda is not right choice if the functions runs more than 15 minutes.
  - Memory Limit: The available memory (RAM) for functions range form 128MB to 3088MB with a 64MB step.
  - Code package size: The zipped lambda code should not exceed 500MB and unzipped shouldn't be larger then 250MB.
  - Concurrency: Default concurrent execution for all Lambda functions within a single AWS account are limited to 1000. But we can increase by creating ticket to AWS support.
  - Payload size: In Amazon API Gateway, when triggering Lambda functions in response to HTTP request, the maximum payload size is 10MB.
  - Cold start time: When function starts based on the event, there can be small latency between the event and the functions runs. If the functions hasn't been used in last 15 minutes then the latency can be up to 5-10 seconds.
- Lambda function can be created:
  - Author from scratch: Creating by our self.
  - Use a blueprint: Choose from existing blueprint.
  - Browse serverless app repository: Choose from other repository.

https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html

AWS Lambda Function in Python

- When we create a Lambda function, we specify a Handler function. It is a function in our code that AWS Lambda invoke when the service executes our code.

```
def handler_name(event, context):
    # todo
    return value
```

- Handler: Handler is a function which calls to start AWS lambda function. The structure of handler function is
- event: AWS Lambda uses this parameter to pass in event data to the handler. This parameter is usually Python dict type. It can be list, str, int, float, or NoneType type. AWS Lambda passes any event data to handler functions as the first parameter. The handler should process the event data and it can invoke any other functions or method in our code. We need to determine the content and structure of the event. The event structure varies based upon services. Let's see event data: https://docs.aws.amazon.com/lambda/latest/dg/lambda-services.html
- context: Context is the second parameter in a handler function. AWS Lambda uses this parameter to provide runtime information to the handler. Our code can interact with AWS Lambda using this context object. For e.g. we can find the execution time remaining before AWS Lambda terminated on our lambda function. Let's see the context object: https://docs.aws.amazon.com/lambda/latest/dg/python-context.html

AWS Lambda Function in Python (cont.)

- The handler can return a value. The action of return value depend on the invocation type to use when invoking the Lambda function:
  - If we use *RequestResponse* invocation type (synchronous execution), then AWS Lambda returns the result of the Python function call to the client invoking the Lambda function. In HTTP response to the invocation request is serialized into JSON. For e.g AWS Lambda console use the *RequestResponse* invocation type, so when we invoke the function using the console, the console will display the returned value.
  - If the handler returns objects that can't be serialize by *json.dumps*, then it will return an error.
  - If the handle returns *None*, as Python functions without a *return* statement implicitly do, the runtime returns *null*.
  - If we use *Event* invocation type (asynchronous execution), the value is discarded.

```
def lambda_handler(event, context):
    msg = 'Hello {} {}!'.format(event['YourFirstName'], event['YourLastName'])
    return {'msg' : msg }
```

- The Lambda function has one function lambda_handler. The function will return a message containing data from the event it received as input.

AWS Lambda Function in Python (cont.)

Logs

- Lambda function comes with a CloudWatch Logs for each instance the function. We can view the Lambda function logs in CloudWatch Log Groups at the console. It will be in /aws/lambda/function-name.
- The logs can also be printed in console using print method or any logging library to write to stdout or stderr. It logs the START, END and REPORT line for each invocation.
- For more info about Logging: https://docs.aws.amazon.com/lambda/latest/dg/python-logging.html

Errors

- If we have error on the code then it raises an error. Lambda catches the error and generate a JSON documents with fields for error message, type, and stack trace.

Demo: Create AWS Lambda Function in Python

- Choose Lambda from Compute service category.
- Click Create function.
- Choose Author from scratch.
- Type function name: copy_file_fromto_s3_bucket
- Choose Python Latest version i.e. (Python 3.8) as Runtime.
- Choose create a new role with basic Lambda permissions, AmazonS3FullAccess, AWSOpsWorksCloudWatchLogs, AWSLambdaFullAccess
- Click create function.
- Under code entry type:
  - Edit code inline. (Choose) Copy the copy from the chat box. The code is on the note below.
  - Upload a zip file.
  - Upload a file from Amazon S3.
- Click Add trigger. Choose S3.
- Under Bucket: Choose logfile-src.
- Under Event Type: Choose All object create.
- Keep Prefix and Suffix empty.
- Choose Enable trigger.
- Prerequisites: Open s3 console in other tab in the browser. Create bucket ”logfile-src” and “logfile-trg”. Create a role lambda_fun_s3. Choose policy : AWSOpsWorksCloudWatchLogs and AmazonS3FullAccess.
- Click Add.

Demo: Create AWS Lambda Function in Python

- Load some files to logfile-src bucket.
- You should see the same file uploaded on logfile-src into logfile-trg bucket.
- Check out the logs on CloudWatch under Log Groups.