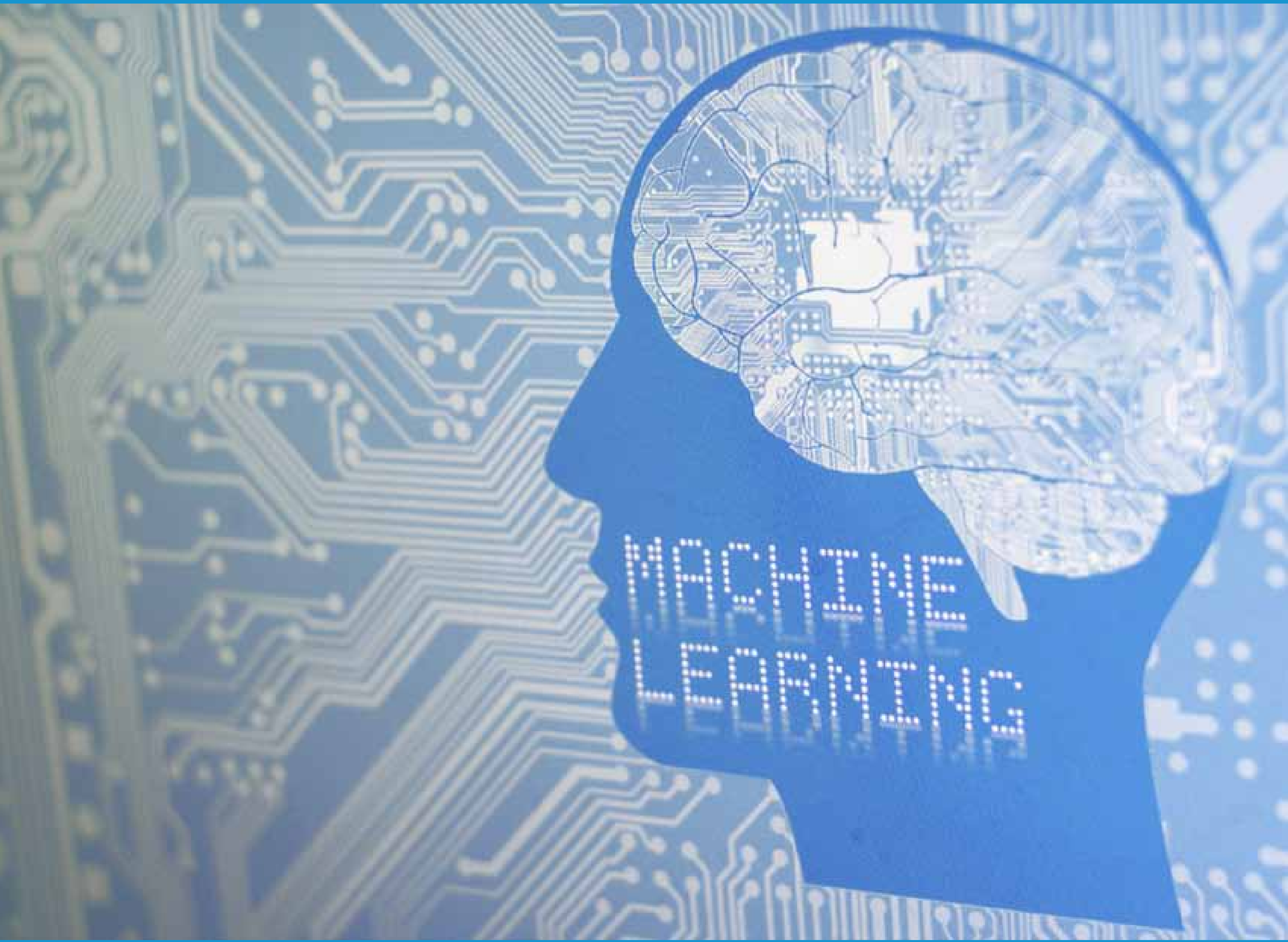
Mphasis
The Next Applied



Deploying Machine Learning Model on AWS Lambda

A Whitepaper by

Eshwar R
Associate Software Engineer, Mphasis NEXTlabs

Ramaraj Periasamy
Cloud Architect, Mphasis NEXTlabs

Sachin Kannan
Associate Software Engineer, Mphasis NEXTlabs

Contents

1. Introduction

1.1 Overview

We live in a digital world, surrounded by huge amount of data, and that's why it is called the data age. The data available to us today has the DNA of the details of our day-to-day activities. Information technology is currently focussed on mining this data from various data sources to gain meaningful insights.

One of the most significant industry trends to gain insight into the vast amount of data is to leverage Artificial Intelligence (AI). AI uses machine learning and deep learning algorithms to analyse the data to make real-time decisions and to trigger subsequent actions.

The AI-based machine learning and deep learning models can broadly be deployed in the following ways:

a) Server-based architecture (which is in practice since long time)

b) Serverless architecture

c) Microservices architecture

Cloud environment provides the required services to deploy the AI algorithms in serverless and in microservices architecture. This document highlights the design and implementation steps for deploying a machine learning model in AWS cloud environment as an AWS Lambda (serverless) function. It also provides the library and package details required to create, deploy and execute the AI model as Lambda function.

1.2 About this team

This document is the result of work done by the engineering team of NEXTlabs on serverless computing in cloud environment. NEXTlabs is an R&D unit of Mphasis, which focusses on developing in-house AI-based (machine learning and deep learning algorithms) solutions to solve complex customer problems.

1.3 Approach

Serverless computing, a service offered by almost all cloud providers, does not need users to manage the underlying resources that run your applications. Hence, the users are free from the tasks involving provisioning, scaling, and managing servers. This helps them channel their time and energy into the development of products that ensure scalability and reliability.

AWS Cloud offers AWS Lambda as their serverless computing platform. Software developers write actionable functions to realize the business logic and run the function as Lambda function. The users can then configure events to trigger it. In this manner, AWS Lambda makes it possible to run event-driven applications in a "serverless" computing model.

2. Deployment Architecture for Machine Learning Model in AWS

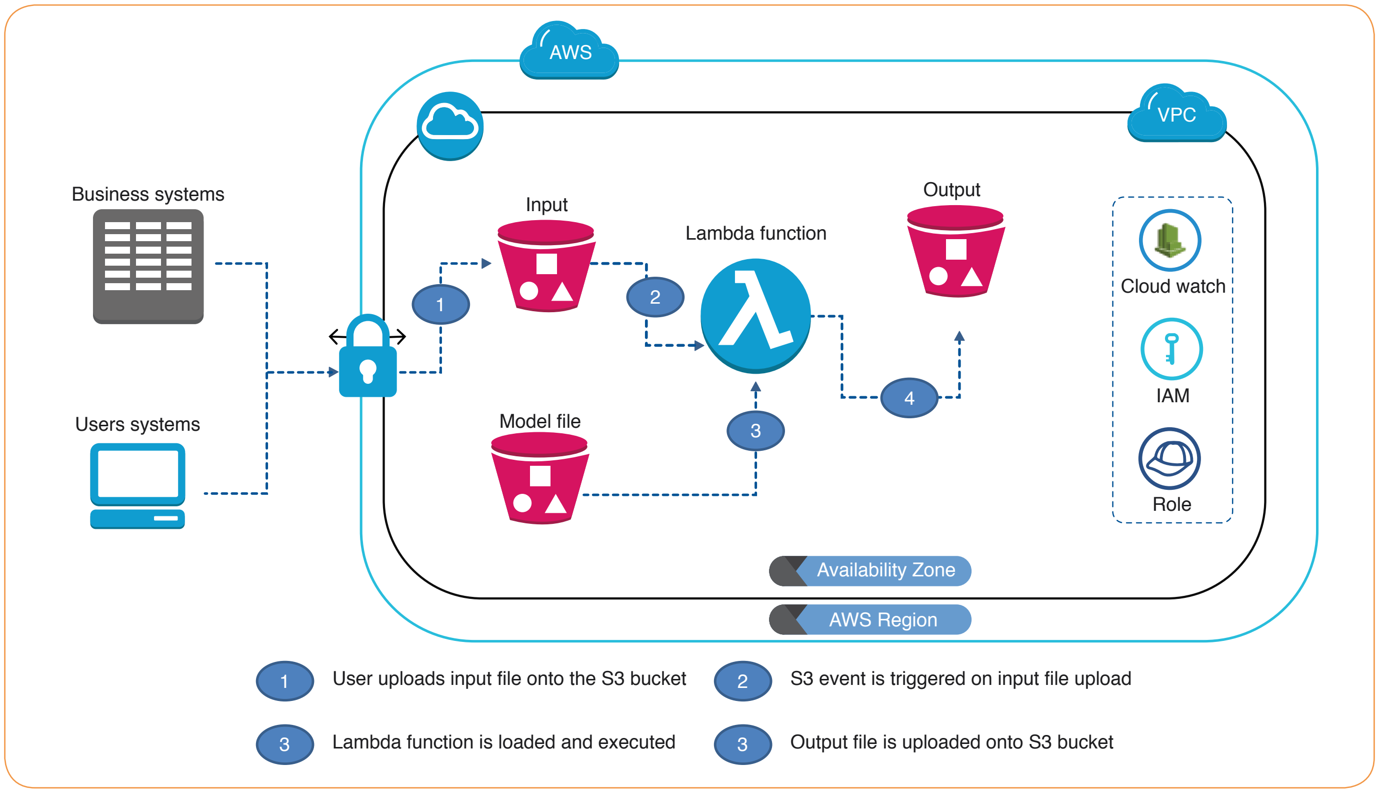
Deployment architecture for machine learning model includes components to:

- Store the input data
- Execute the Lambda function
- Store the packaged model file and the Lambda function code

Kinesis streams, DynamoDB and S3 buckets can be the source of inputs. In our case, the input file, which is a .CSV file, is uploaded into a S3 bucket and it triggers an Amazon S3 event. AWS Lambda executes the Lambda functions in response to the event triggered. The Lambda function then uploads the output to another S3 bucket.

Users can download the output file from S3 bucket and use a visualizing tool to analyze the output. Amazon QuickSight is one such tool from AWS, but the user is free to choose his own tool based on his requirements. Amazon CloudWatch monitors the resource usage and provides the metrics based on the logs uploaded by the Lambda functions at runtime. IAM roles control the access for the services used in the architecture.

Below is the detailed architecture of the serverless deployment on AWS.


AWS
VPC
Business systems
Users systems
Input
Lambda function
Output
Model file
Cloud watch
IAM
Role
1
2
3
4
Availability Zone
AWS Region
1 User uploads input file onto the S3 bucket
2 S3 event is triggered on input file upload
3 Lambda function is loaded and executed
3 Output file is uploaded onto S3 bucket

2.1 Implementation of machine learning model as a Lambda function

Lambda functions are implemented in the following two ways:

1. Implementation without advanced packages
2. Implementation with advanced packages

2.1.1 Implementation without advanced packages

You can use AWS Console to implement Lambda functions when the source code of the function does not require any special packages other than AWS SDK. Please note that Lambda functions come pre-installed with AWS SDK files. You can use the online editor in AWS console to develop the source code for the Lambda function. This method is useful to deploy small workloads as Lambda functions, which do not require complex packages.

2.1.2 Implementation with advanced packages

Lambda does not support adding packages through AWS Console editor. Therefore, when the source code of the Lambda functions requires additional packages, such as a graphics library for image processing, you cannot use AWS Console editor. In that case, you will need to create Lambda function deployment package separately. This method has to be used even when Lambda needs to be created programmatically using AWS CLI. You can deploy the package into Lambda function using AWS console or upload the package into an S3 bucket, and configure the Lambda function to execute it.

The deployment package has been created separately for this research work as the model used (sentiment analysis) requires complex packages. The package file was then created as zip and uploaded to an S3 bucket. The Lambda function was configured to execute it.

Note: We recommend that users follow this method as it gives complete programmatic access.

2.2 Specific Considerations

AWS Lambda has the following limitations. It will be good to keep this in mind while designing server architecture using AWS Lambda.

- "hard" limitations for the runtime environment.
  - ♦ The disk space is limited to 512 MB
  - ♦ Memory can vary from 128 to 1536 MB
  - ♦ Maximum execution timeout for a function is 5 minutes
  - ♦ The default package size is 50 MB
- Limitations on the requests served by Lambda
  - ♦ Request and response body payload size is maximized to 6 MB
  - ♦ Event request body can be up to 128 KB

You can approach AWS support centre to increase these limits. Refer to the below AWS URL for more details on AWS Lambda limits.
https://docs.aws.amazon.com/lambda/latest/dg/limits.html

3. Hardware/Software Requirements

3.1 Hardware requirements

3.1.1 Machines

AWS does not allow console access for the instances on which Lambda functions run. Only the size of the memory needed to run the Lambda functions need to be specified. Memory can be configured between 128 MB to 3008 MB.

3.2 Software requirements

3.2.1 Product version

The following software packages are required to run Lambda functions in Python environment.
These Python packages are specific to Amazon Linux.

√ *_pycache_*
√ *certifi-2018.1.18.dist-info*
√ *chardet-3.0.4.dist-info*
√ *colorama-0.3.9.dist-info*
√ *Dateutil*
√ *docutils-0.14.dist-info*
√ *idna-2.6.dist-info*
√ *jmespath-0.9.3.dist-info*
√ *numpy-1.13.3.dist-info*
√ *pandas-0.20.3.dist-info*
√ *pkg-resources*
√ *pyasn1-0.4.2.dist-info*
√ *pytz-2017.2.dist-info*
√ *PyYAML-3.12-py3.6.egg-info*
√ *requests-2.18.4.dist-info*
√ *rsa-3.4.2.dist-info*
√ *setuptools-38.4.0.dist-info*
√ *six-1.11.0.dist-info*
√ *vaderSentiment-2.5.dist-info*

3.3 Machine learning model requirements

This section articulates the important steps that one needs to follow while choosing algorithms to create and train machine learning models.

- Choose the appropriate algorithm to create the model. Identify required training data for the model.
- Consider the following factors while deciding on infrastructure requirements to train machine learning models
  - ♦ Algorithm complexity and parallelism requirements to decide CPU requirements
  - ♦ Data size (length of the array and data type) to decide the memory requirements
- Make sure the input data format matches with the attributes that model trains on. The input data can be a JSON/text/xml file as well as streaming data.
- The input data must be pre-processed before the models consume it, to see if there are any missing values and data mismatch present in the input. Design of pre-processing stage should be as per the input data format.
- Machine learning models consume data in data frames, which are the standard data structures used in machine learning models. The input data should be provided in CSV format, as it is structurally similar to the data frame structure.
- Stop training the model when it provides the required accuracy consistently over a period.
- Retrain the algorithm if the testing accuracy deviates drastically from the training accuracy (bias-variance trade-off). In such cases, relook at the input data as well.

The model used in the research work for this document is on sentiment analysis, which takes CSV file as input and provides formatted CSV file as output. The model file is in Python language.

3.4 AWS resource requirements

The following resources are to be created in AWS environments to deploy and execute the machine learning model.

- **Input bucket** – Create an S3 bucket that holds the raw input CSV files. Uploading a file onto this bucket will trigger the Lambda function.
- **Output bucket** – Create an S3 bucket to store the data processed by the Lambda function.
- **Lambda function** - Write an AWS Lambda function that loads the model file and calculates the sentiment scores. (As mentioned earlier, the model deployment package is kept in an S3 bucket and the path is configured in Lambda function. The example path can be S3://[CodeBucket]/[CodeKeyPrefix]/data-processor.zip).
- **Lambda execution role** – Create an AWS Identity and Access Management (IAM) role to execute the Lambda function. This role will give the Lambda function the required access to S3 buckets and for writing logs to CloudWatch.
- **S3 Access policy** - Create an IAM policy to be associated with Lambda Execution Role. This policy will provide read and write access to S3 buckets.
- **S3 event creation** – Create a trigger configuration to invoke Lambda function when input file is uploaded onto a specific S3 bucket.

4. Configuration and Deployment

Following configurations need to be done on the AWS resources used in deployment:

- Configure S3 buckets to allow read and write operations from Lambda function
- Configure the Lambda function to access the S3 buckets for reading the input data and writing the output data
- Configure the Lambda function to retrieve the function to be executed from S3 bucket
- Configure the Lambda function to write execution logs to CloudWatch to generate CloudWatch metrics

The following sections detail the activities that need to be carried out to complete the configuration.

4.1 S3 bucket settings

S3 buckets store the Lambda function, which executes at runtime, the input data and the output data. We need to apply the policies (as detailed below) on these buckets for the read, write and execution operations, to ensure the buckets are accessible from the Lambda function.

S3 policy for the bucket to store Lambda function – As mentioned in section 2.1, the Lambda function deployment package is uploaded onto S3 bucket. When the deployment package is uploaded, the Lambda function is triggered, which then retrieves the package from the S3 bucket. This policy allows access only to the content of the bucket that has the Lambda assigned roles. Granting public access is not recommended.

Given below is the demonstration of creating and assigning an S3 policy using AWS CLI.

To create an S3 policy

```
aws iam create-policy --policy-name <policy name> --policy-document file://<input to the policy>
```

To assign the policy to an S3 bucket

```
aws s3api put-bucket-policy --bucket <bucket name> --policy file://<policy name>
```

Sample policy statement

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": [
        "s3:DeleteObject",
        "s3:PutObject",
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::MyBucket/*"
    }
  ]
}
```

S3 policy for the bucket to store input data

User uploads the input file to the bucket, which is then processed by the Lambda function. Steps to create and assign the policy remain the same as above.

S3 policy for the bucket to store input data

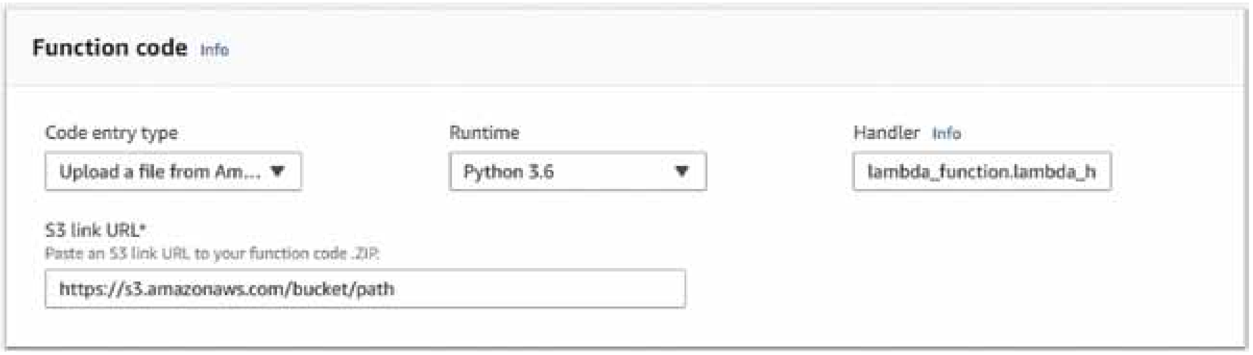
Lambda function uploads the output file once the input file is processed completely. It is recommended to give this bucket public read access unless you can specify specific IP ranges as the source of request.

Please find the sample policy that we used in our output S3 bucket.

```
{
  "Version":"2012-10-17",
  "Statement":[
   {
     "Sid":"AddPerm",
     "Effect":"Allow",
     "Principal": "*",
     "Action":["s3:PutObject"],
     "Resource":["arn:aws:s3:::examplebucket/*"]
   }
 ]
}
```

4.2 AWS Lambda configuration

- Choose a runtime environment for the AWS Lambda function to execute. AWS Lambda supports the following runtime environments.
  - *Python – Versions 2.7 and 3.6*
  - *Node.js – Versions 4.8, 6.10 and 8.10*
  - *Java 8*
  - *Go 1.x*
  - *C# - .NET core 1.0 and .NET core 2.0*
- Configure basic settings like max execution time, function handler and specify the URL of the deployment package. For the research work, we have chosen the runtime environment as Python 3.6 and defined the Lambda function handler as below. The function handler will bootstrap the function once the file upload event triggers.


Function code Info

Code entry type: Upload a file from Am...

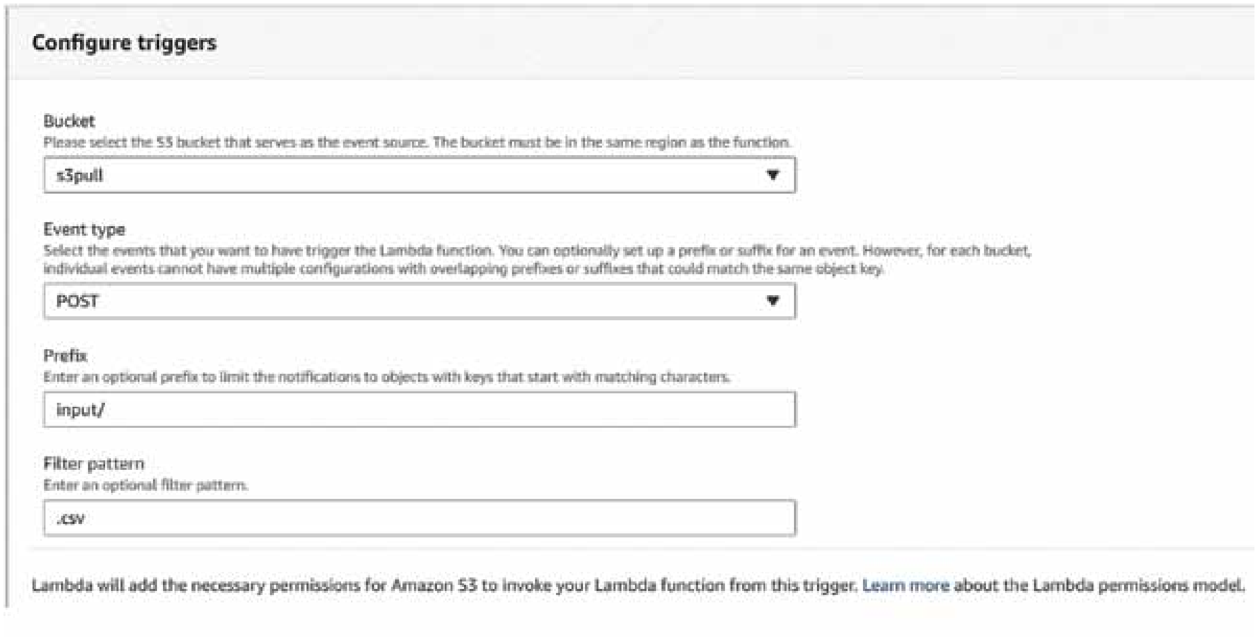
Runtime: Python 3.6

Handler Info: lambda_function.lambda_h

S3 link URL*
Paste an S3 link URL to your function code .ZIP.
https://s3.amazonaws.com/bucket/path

- Configure trigger events to use S3 as a trigger and specify the trigger type (for example, PUT action) along with any prefix, to define the folder structure.

  Applying this will result in Lambda automatically allocating the appropriate policies to access the configured S3 resources.



Configure triggers

Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.
s3pull

Event type
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.
POST

Prefix
Enter an optional prefix to limit the notifications to objects with keys that start with matching characters.
input/

Filter pattern
Enter an optional filter pattern.
.csv

Lambda will add the necessary permissions for Amazon S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

- Assign an execution role to Lambda function, which uses this role to upload the processed file to the output S3 bucket and to write execution logs to CloudWatch. S3 bucket policy remains the same as defined in the previous sections.

  Here is an example of the policy that allows write to CloudWatch.

```
{
  "Version": "2012-10-17",
  "Statement":[{
     "Effect":"Allow",
     "Action":["cloudwatch:GetMetricStatistics","cloudwatch:ListMetrics"],
     "Resource":"*",
     "Condition":{
        "Bool":{
           "aws:SecureTransport":"true"
           }
        }
     }
  ]
}
```

4.3 Building deployment package

Building the deployment package involves carrying out the following steps:

- Create a deployment package folder as per your choice.
- Identify dependent libraries and packages to execute the machine learning model created in Python.
  In our case, we used libraries and packages compatible with Amazon Linux since the Lambda functions are run on AWS Linux.
- Install all the packages in a deployment folder inside the root directory.

  ```
  pip install module-name -t /path/to/project-dir
  ```

  Note: Section 3.2 - Software discusses the package requirements in detail.
- Include the source code of the Lambda function in the deployment package folder and zip.
- Upload the zipped folder onto an Amazon S3 bucket and save the URL of the bucket.
  Please note that the URL should be specified in the Lambda configuration.

5. Execution of Lambda Function

When the user uploads the input file onto the S3 bucket, the Lambda event is triggered automatically. On receiving the trigger, AWS lambda searches for the configured handler function. Lambda function gets executed in the following three logical phases.

5.1 Downloading input CSV from S3 input bucket

In this phase, the Lambda downloads the input file and uploads the file onto the *"tmp"* folder of Lambda. The below script which is part of the Lambda function performs this activity. In our case, we have used Python SDK's (*boto* library) *download file* method to download the files from S3. Please note that this gives us access only to the *"tmp"* folder of Lamda instance.

```
import boto3
import botocore
BUCKET_NAME = 'input-bucket' # replace with your bucket name
KEY = 'input_CSV_in_s3.csv' # replace with your object key
s3 = boto3.resource('s3')
try:
    s3.Bucket(BUCKET_NAME).download_file(KEY, 'my_input_CSV.csv')
except botocore.exceptions.ClientError as e:
    if e.response['Error']['Code'] == "404":
        print("The object does not exist.")
    else:
        raise
```

5.2 Processing the input data using machine learning model

This part of the function is responsible for the input data analysis. Based on model file used in the Lambda function, the input data is analysed and the output fields are written to a CSV file.

In our case, we have used the model based on sentimental analysis inside the Lambda function.

5.3 Pushing the generated output CSV to S3

The output CSV generated by the Lambda function is uploaded to an amazon S3 bucket. Any visualization tool such as Amazon QuickSight, to gain deep insights into the data can consume this output data. Users can take appropriate follow-up actions based on the analysis.

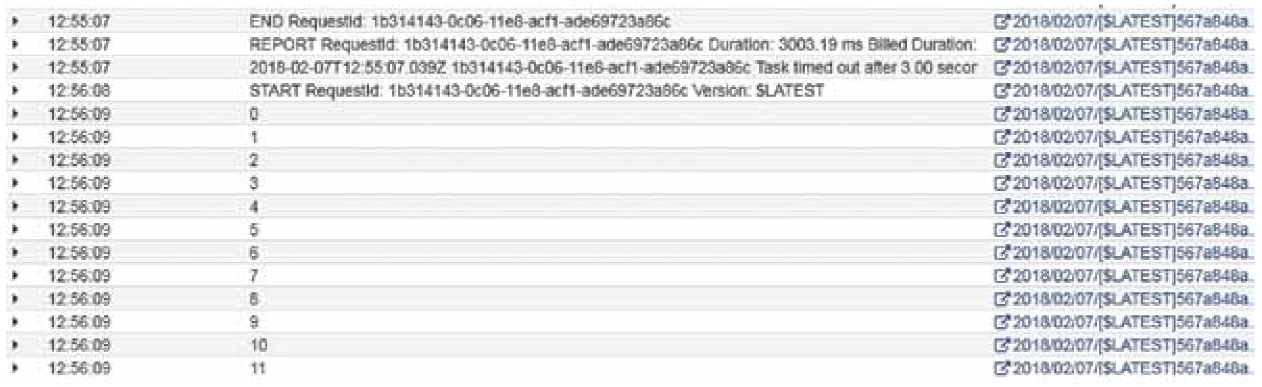
Below is the code snippet to upload output data to an S3 bucket.

```
import boto3
s3 = boto3.resource('s3')
s3.meta.client.upload_file('/tmp/output_file','mybucket','output.csv')
```

5.4 CloudWatch logs and metrics

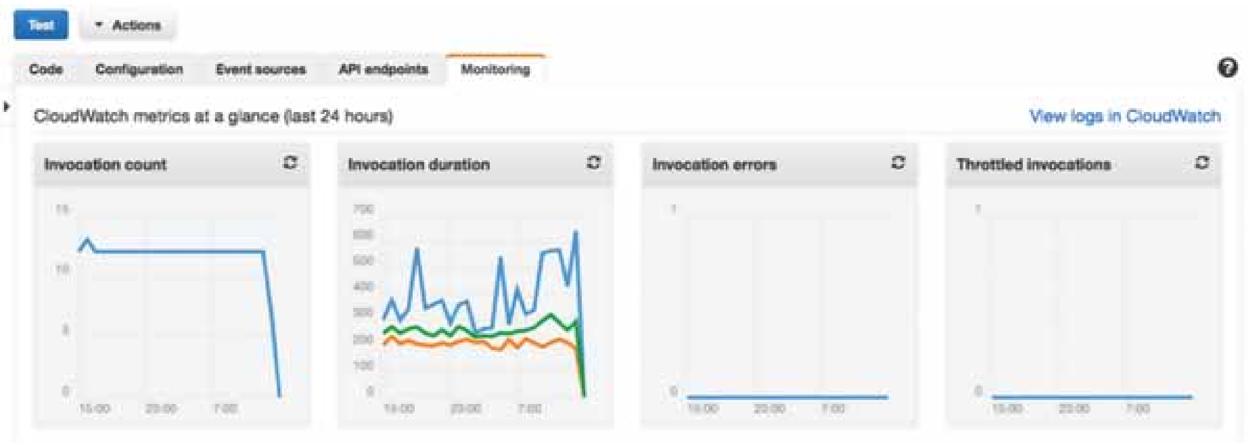
As part of the Lambda function execution, it writes the execution details to CloudWatch. Given below is sample log entry and metrics.

Logs



```
12:55:07   END RequestId: 1b314143-0c06-11e8-acf1-ade69723a86c                                                   2018/02/07/[$LATEST]567a848a
12:55:07   REPORT RequestId: 1b314143-0c06-11e8-acf1-ade69723a86c Duration: 3003.19 ms Billed Duration:          2018/02/07/[$LATEST]567a848a
12:55:07   2018-02-07T12:55:07.039Z 1b314143-0c06-11e8-acf1-ade69723a86c Task timed out after 3.00 secor          2018/02/07/[$LATEST]567a848a
12:56:08   START RequestId: 1b314143-0c06-11e8-acf1-ade69723a86c Version: $LATEST                                 2018/02/07/[$LATEST]567a848a
12:56:09   0                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   1                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   2                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   3                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   4                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   5                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   6                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   7                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   8                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   9                                                                                                      2018/02/07/[$LATEST]567a848a
12:56:09   10                                                                                                     2018/02/07/[$LATEST]567a848a
12:56:09   11                                                                                                     2018/02/07/[$LATEST]567a848a
```

Metrics



6. Challenges Faced

Below challenge was faced during the implementation of the deployment architecture for writing this document.

- Finding the lambda machine specific packages and libraries
  - ♦ AWS Lambda service internally runs on amazon Linux AMI. It took a lot of time to identify the required packages

7. References

We referred to the following links for our work on this paper.

- https://aws.amazon.com/lambda/
- https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html
- https://aws.amazon.com/sdk-for-python/

8. Disclaimer

Information in this document is provided 'AS IS' without warranty of any kind. Mention or reference to non-Mphasis products are for informational purposes only and does not constitute an endorsement of such products by Mphasis. Performance is based on measurements and projections using standard Mphasis benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

9. Trademarks

The following terms are trademarks of Mphasis and other companies in the United States, other countries, or both:

- AWS® is trademark of Amazon Web Services
- Python is a software distributed under Python Software Foundation License
- Amazon Linux is a version of Linux operating system developed by Amazon Web Services to host in Amazon EC2 instances
- Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both

Author

Eshwar R

Eshwar works as an Associate Software Engineer at Mphasis NEXTlabs. He completed his B.Tech from M S Ramaiah Institute of Technology, Bengaluru. His interests lie in the field of Cloud Computing and Artificial Intelligence. He likes to learn latest technologies and experiment them for various use cases.

Ramaraj Periasamy

Ramaraj is a Cloud Architect with Mphasis NEXTlabs. His focus area is design and development of AI-based cognitive solutions on Cloud. He leads design and development of migration artifacts that help accelerate customer Cloud journey. He also supports other Cloud IP initiatives within NEXTlabs. Prior to Mphasis, he has worked in various companies like IBM, Infosys and Wipro as Cloud Architect. He completed his Master of Engineering from Bharathiar University and B.Tech from Madras Institute of Technology, Anna University.

Sachin Kannan

Sachin, an Associate Software Engineer at Mphasis, is part of NEXTlabs HyperGraf™ team. He has implemented several modules of HyperGraf™ as Lambda function and handles all AWS services related deployment activities in NEXTlabs.

About Mphasis

Mphasis (BSE: 526299; NSE: MPHASIS) applies next-generation technology to help enterprises transform businesses globally. Customer centricity is foundational to Mphasis and is reflected in the Mphasis' Front2Back™ Transformation approach. Front2Back™ uses the exponential power of cloud and cognitive to provide hyper-personalized ($C = X2C^{2}_{TM} = 1$) digital experience to clients and their end customers. Mphasis' Service Transformation approach helps 'shrink the core' through the application of digital technologies across legacy environments within an enterprise, enabling businesses to stay ahead in a changing world. Mphasis' core reference architectures and tools, speed and innovation with domain expertise and specialization are key to building strong relationships with marquee clients. To know more, please visit www.mphasis.com

For more information, contact: marketinginfo@mphasis.com

USA	UK	INDIA
460 Park Avenue South	88 Wood Street	Bagmane World Technology Center
Suite #1101	London EC2V 7RS, UK	Marathahalli Ring Road
New York, NY 10016, USA	Tel.: +44 20 8528 1000	DoddanakundhiVillage, Mahadevapura
Tel.: +1 212 686 6655		Bangalore 560 048, India
		Tel.: +91 80 3352 5000

www.mphasis.com

VAS 06/06/18 US LETTER BASIL 4710