EMAIL SPAM DETECTION

PROJECT REPORT

18CSE344T- CLOUD ARCHITECTURE

(2018 Regulation)

III Year/ V Semester

Academic Year: 2022 -2023

By

Chinta Pradeep(RA2011028010059)

Roshini Jammula(RA2011028010104)

Shardul Bhardwaj(RA2011028010058)

Avipsha Panigrahi(RA2011028010101)

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Kancheepuram

NOVEMBER 2022

BONAFIDE

This is to certify that the project report titled "EMAIL SPAM DETECTION" is the bonafide work of Chinta Pradeep(RA2011028010059), Roshini Jammula(RA2011028010104), Shardul Bhardwaj(RA2011028010058), Avipsha Panigrahi(RA2011028010101) who undertook the task of completing the project withinthe allotted time.

Signature of the Course Faculty

Dr. M. Shobana

Assistant Professor

Department of NWC,

SRM Institute of Science and Technology

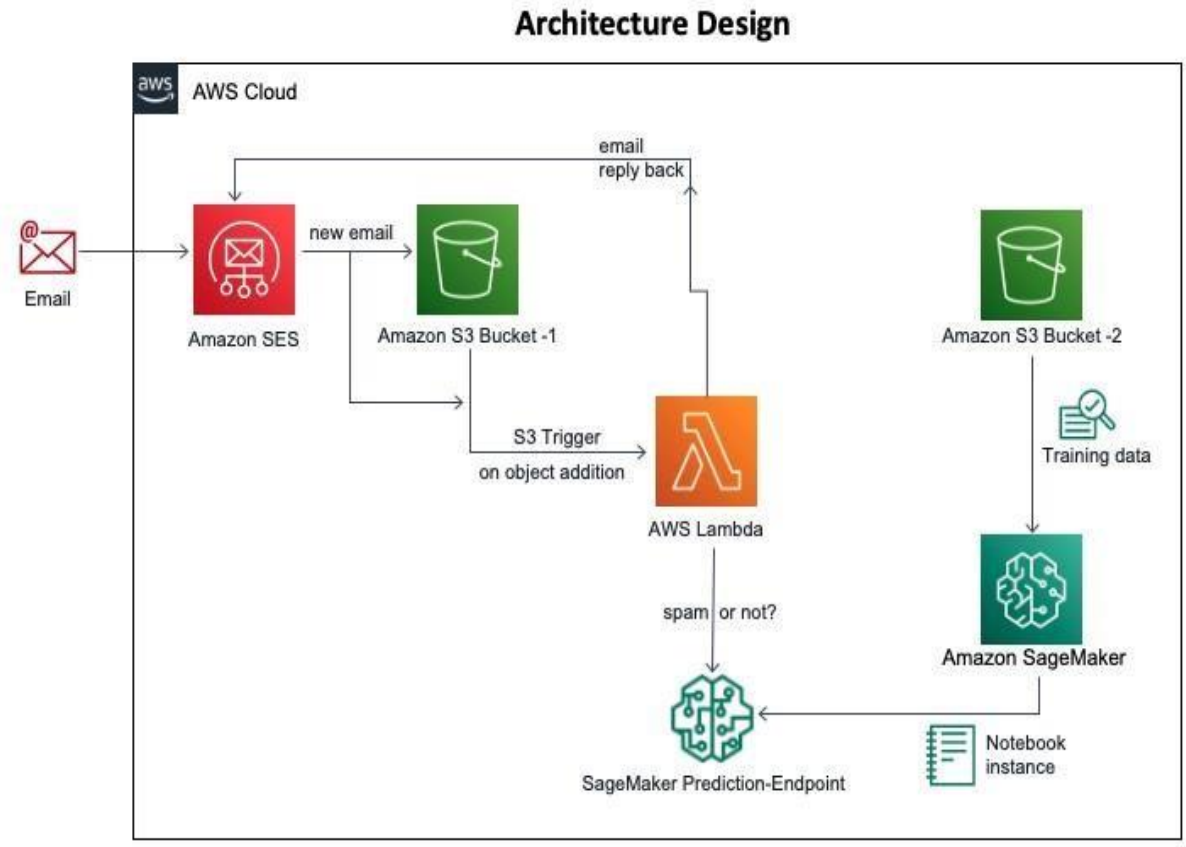
TABLE OF CONTENTS

EMAIL SPAM DETECTION

ABSTRACT: -

A spam detection system built on AWS cloud, that upon receipt of an email message, automatically flags it as spam or not. This is based on the prediction obtained from the machine learning model created using Amazon SageMaker. The definition and provision of the resources on AWS cloud is done through the AWS Cloudformation template.

ARCHITECTURE DESIGN:-



PROCEDURE: -

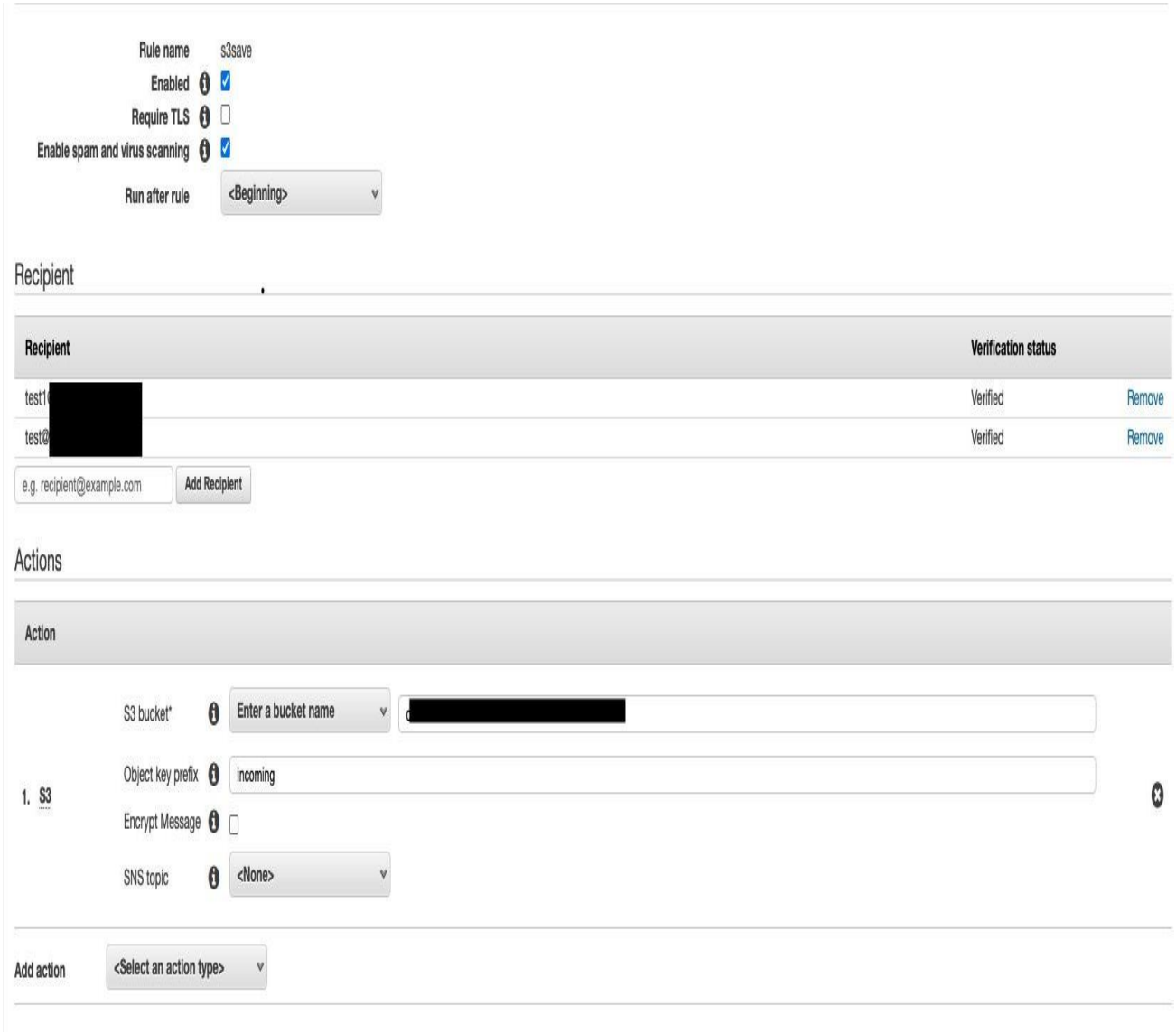
- AWS cloud subscription (AWS Free tier works)
- A custom domain
- An email address for that domain (user1@example.com). This will require adding up *MX and TXT records* on the domain provider site to receive an email. Refer the documentation of domain provider to add the records.

Standing up the AWS resources and Machine Learning spam classifier model using AWS Sagemaker:

1. Use the Cloud Formation template to create the S3 buckets (for storing emails), AWS sage maker endpoint, notebook instance, Lambda funtion and appropriate roles.
2. Follow the reference link ***Build and train a spam filter machine learning model*** to create, train and deploy the spam classifier model. AWS sagemaker uses XGBoost ML algorithm and the Bank marketing data set to build and train the model. You can use your own custom made machine learning model and data set for this purpose.

Setting up Simple Email Service (SES) on AWS:

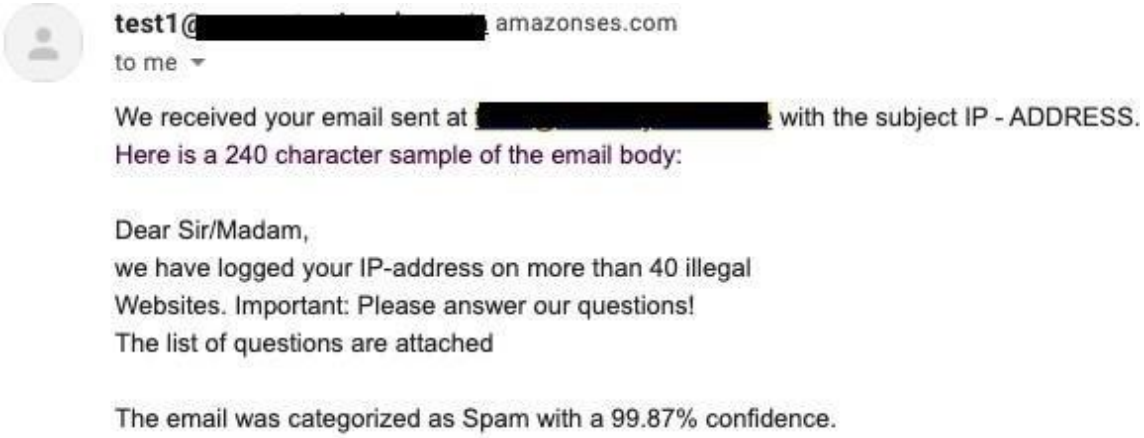
1. Open SES home on AWS console and navigate to *Domains*. Verify a new domain (prerequisites) that you already own. Copy the 'TXT record' provided on console and add it in your Domain provider's site.
2. From the SES home, go to the 'SMTP settings' and copy the 'server name'. Add this as a 'MX record' in your Domain provider's site, just like the 'TXT record' added in the last step. This and previous settings will link the Amazon SES with your custom email address.
3. Now we need to receive the email that is sent on our custom email address, and store it in the S3 bucket. For this - from the SES home, navigate to the *Email Receiving* > *Rule Sets* > *Create Rule*. Use something like the below image to configure the rule set. Use the name of the S3 bucket which was created using the cloud formation template in the previous steps.


Rule name
s3save
Enabled
Require TLS
Enable spam and virus scanning
Run after rule
<Beginning>
Recipient
Recipient
Verification status
Verified
Remove
Verified
Remove
e.g. recipient@example.com
Add Recipient
Actions
Action
S3 bucket*
Enter a bucket name
Object key prefix
incoming
1. S3
Encrypt Message
SNS topic
<None>
Add action
<Select an action type>

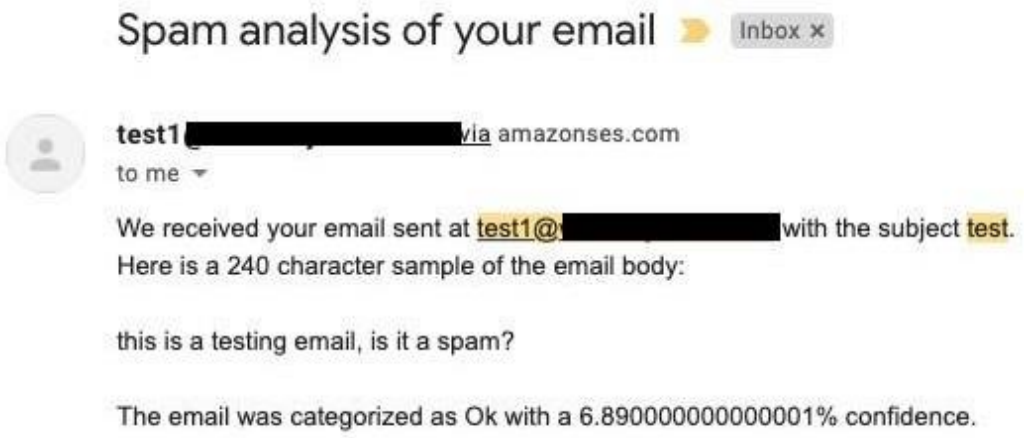
WORKING: -

1. Send an email to the email address that is used by the spam detector:-

2. Reply that we get, if the email is a spam:-



3. Reply that we get, if the email is NOT a spam:-



IMPLEMENTATION CODE:-

CloudFormationStack.json:-

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",
 "Description" : "AWS CloudFormation Template",
 "Resources" : {
  "hw003buck001" : {
    "Type" : "AWS::S3::Bucket",
    "Properties" : {
      "AccessControl" : "PublicRead"
    }
  },
  "BucketPolicy": {
     "Type": "AWS::S3::BucketPolicy",
     "Properties": {
        "PolicyDocument": {
           "Id": "MyPolicyforbuck001",
           "Version": "2012-10-17",
```

```
        "Statement": [
            {
                "Sid": "PublicReadForGetBucketObjects",
                "Effect": "Allow",
                "Principal":  "*",
                "Action": "s3:GetObject",
                "Resource": {
                    "Fn::Join": [
                        "",
                        [
                            "arn:aws:s3:::",
                            {
                                "Ref": "hw003buck001"
                            },
                            "/*"
                        ]
                    ]
                }
            }
        ]
    },
    "Bucket": {
        "Ref": "hw003buck001"
    }
}
```

```
},
"hw003buck002" : {
  "Type" : "AWS::S3::Bucket",
  "Properties" : {
    "AccessControl" : "PublicRead",
    "WebsiteConfiguration" : {
    "IndexDocument" : "index.html"
     }
  }
},
"Bucket2Policy": {
    "Type": "AWS::S3::BucketPolicy",
    "Properties": {
        "PolicyDocument": {
            "Id": "MyPolicyforbuck002",
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "PublicReadForGetBucketObjects",
                    "Effect": "Allow",
                    "Principal":  "*",
                    "Action": "s3:GetObject",
                    "Resource": {
                        "Fn::Join": [
                            "",
```

```
                    [
                        "arn:aws:s3:::",
                        {
                            "Ref": "hw003buck002"
                        },
                        "/*"
                    ]
                ]
            }
        }
      ]
    },
    "Bucket": {
      "Ref": "hw003buck002"
    }
  }
},
"ExecutionRoleLambda": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
    "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
```

```
  "Principal": {
    "Service": [
      "lambda.amazonaws.com"
        ]
      },
      "Action": [
        "sts:AssumeRole"
      ]
    }
  ]
},
"Path": "/",
"Policies": [
  {
    "PolicyName": "root2",
    "PolicyDocument": {
    "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "*",
          "Resource": "*"
        }
      ]
    }
```

```
            }
          ]
        }
      },
    "hw3lf001": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "Handler": "index.handler",
        "Role" : {
          "Fn::GetAtt" : [
            "ExecutionRoleLambda", "Arn"
          ]
        },
        "Code": {
          "ZipFile": "def my_handler(event, context):\n message = \"Hello Lambda World!\"\n  return message\n"
      },
        "Runtime": "python3.8",
        "Timeout": 25,
        "TracingConfig": {
          "Mode": "Active"
        }
      }
    },
    "NotebookInstance-hw3": {
      "Type": "AWS::SageMaker::NotebookInstance",
```

```
      "Properties": {
       "InstanceType": "ml.t3.medium",
       "RoleArn": { "Fn::GetAtt" : [ "ExecutionRole", "Arn" ] }
         }
       },
  "ExecutionRole": {
     "Type": "AWS::IAM::Role",
     "Properties": {
       "AssumeRolePolicyDocument": {
       "Version": "2012-10-17",
         "Statement": [
           {
             "Effect": "Allow",
             "Principal": {
               "Service": [
                 "sagemaker.amazonaws.com"
                   ]
                 },
                 "Action": [
                   "sts:AssumeRole"
                 ]
               }
            ]
           },
           "Path": "/",
```

```
            "Policies": [
              {
                "PolicyName": "root",
                "PolicyDocument": {
                "Version": "2012-10-17",
                  "Statement": [
                    {
                      "Effect": "Allow",
                      "Action": "*",
                      "Resource": "*"
                    }
                  ]
                }
              }
            ]
          }
        }
    },
  "Outputs" : {
    "S3Bucket1SecureURL" : {
      "Value" : { "Fn::Join" : [ "", [ "https://", { "Fn::GetAtt" : [ "hw003buck001", "DomainName"
] } ] ] },
      "Description" : "Name of S3 bucket to hold email content"
    },
    "S3Bucket2SecureURL" : {
```

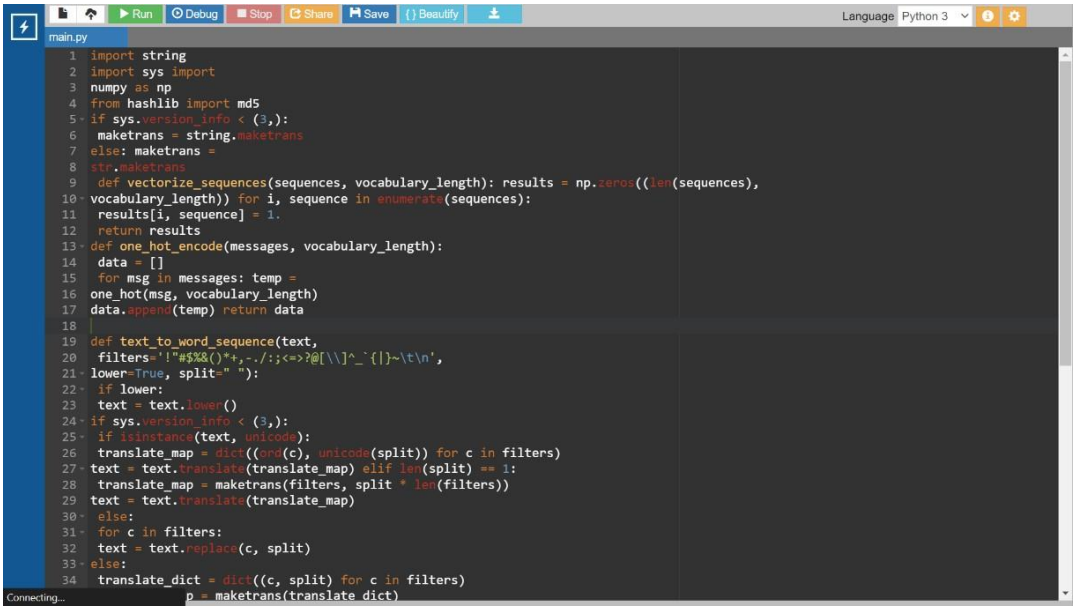
```
      "Value" : { "Fn::Join" : [ "", [ "https://", { "Fn::GetAtt" : [ "hw003buck002",
"DomainName" ] } ] ] },

      "Description" : "Name of S3 bucket to hold email content"

    },

    "NotebookInstanceId": {

      "Value": { "Ref" : "NotebookInstance" }

   }

  }

}
```

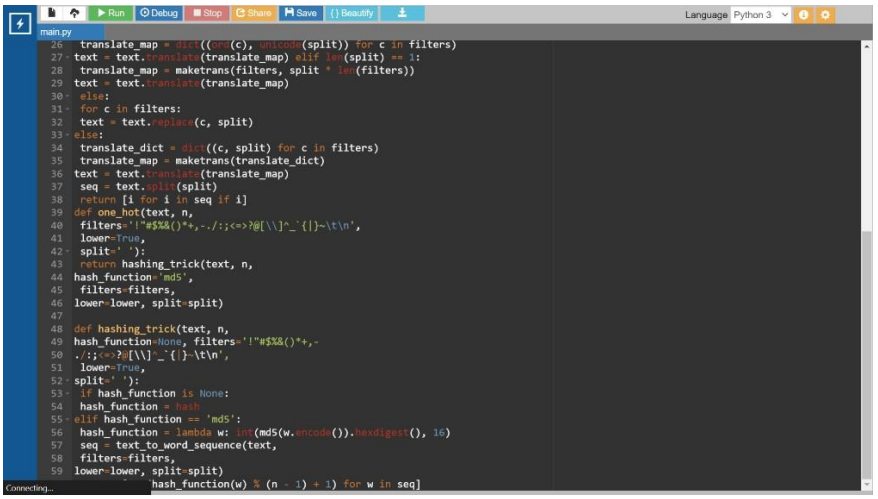
sms_spam_classifier_utilities.py :-

```
import string

import sys import

numpy as np

 from hashlib import md5

 if sys.version_info < (3,):

   maketrans = string.maketrans

else:    maketrans =

str.maketrans

    def vectorize_sequences(sequences, vocabulary_length):    results = np.zeros((len(sequences),
vocabulary_length))    for i, sequence in enumerate(sequences):

     results[i, sequence] = 1.

   return results
 def one_hot_encode(messages, vocabulary_length):

   data = []

   for msg in messages:       temp =

one_hot(msg, vocabulary_length)

data.append(temp)    return data
```

```
def text_to_word_sequence(text,
                          filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
lower=True, split=" "):
    if lower:
        text = text.lower()
  if sys.version_info < (3,):
        if isinstance(text, unicode):
            translate_map = dict((ord(c), unicode(split)) for c in filters)
text = text.translate(translate_map)        elif len(split) == 1:
            translate_map = maketrans(filters, split * len(filters))
text = text.translate(translate_map)
        else:
            for c in filters:
                text = text.replace(c, split)
else:
        translate_dict = dict((c, split) for c in filters)
        translate_map = maketrans(translate_dict)
text = text.translate(translate_map)
    seq = text.split(split)
   return [i for i in seq if i]
 def one_hot(text, n,
  filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
    lower=True,
            split=' '):
    return hashing_trick(text, n,
hash_function='md5',
                         filters=filters,
lower=lower,                        split=split)
def hashing_trick(text, n,
hash_function=None,                 filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~\t\n',
```

```
            lower=True,
split=' '):
    if hash_function is None:
  hash_function = hash
 elif hash_function == 'md5':
    hash_function = lambda w: int(md5(w.encode()).hexdigest(), 16)
  seq = text_to_word_sequence(text,
                    filters=filters,
lower=lower,                    split=split)
  return [int(hash_function(w) % (n - 1) + 1) for w in seq]
```





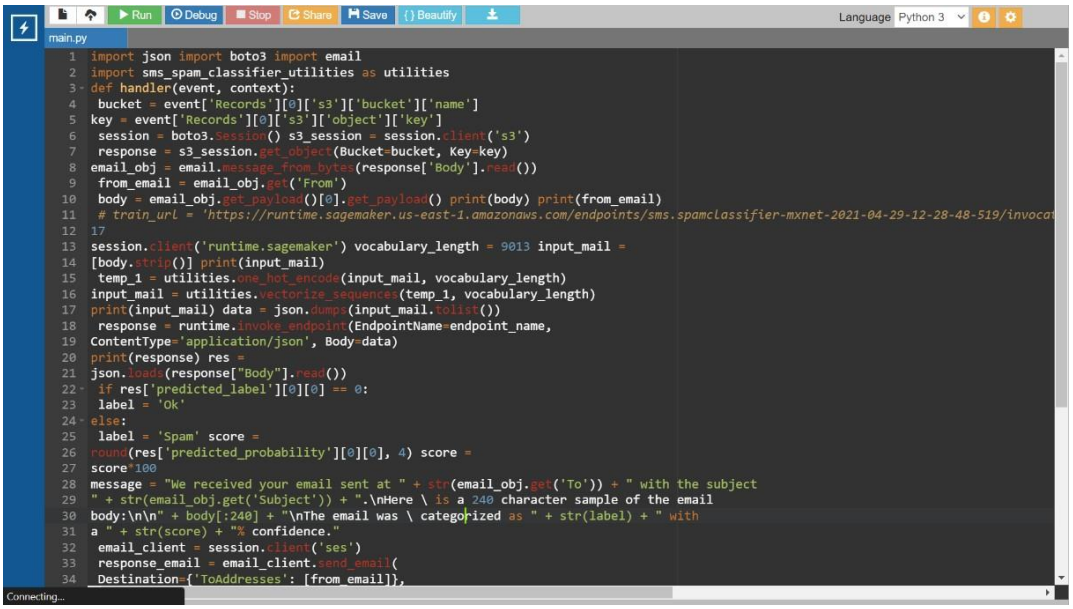
spam_classify.py:-

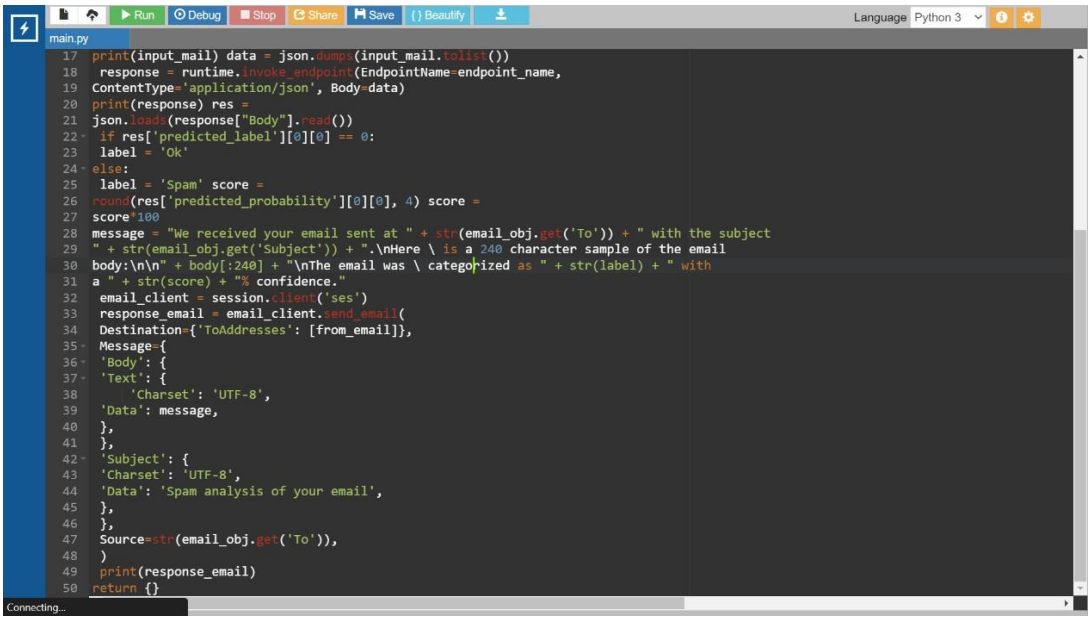
```
import json import boto3 import email
import sms_spam_classifier_utilities as utilities
def handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
key = event['Records'][0]['s3']['object']['key']
   session = boto3.Session()     s3_session = session.client('s3')
 response = s3_session.get_object(Bucket=bucket, Key=key)
 email_obj = email.message_from_bytes(response['Body'].read())
    from_email = email_obj.get('From')
   body = email_obj.get_payload()[0].get_payload()     print(body)     print(from_email)
    # train_url = 'https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/sms-
spamclassifier-mxnet-2021-04-29-12-28-48-519/invocations'     endpoint_name = 'sms-
spam-classifier-mxnet-2021-04-29-12-28-48-519'     runtime =
session.client('runtime.sagemaker')     vocabulary_length = 9013     input_mail =
[body.strip()]     print(input_mail)

    temp_1 = utilities.one_hot_encode(input_mail, vocabulary_length)
input_mail = utilities.vectorize_sequences(temp_1, vocabulary_length)
print(input_mail)     data = json.dumps(input_mail.tolist())
    response = runtime.invoke_endpoint(EndpointName=endpoint_name,
ContentType='application/json', Body=data)
print(response)     res =
json.loads(response["Body"].read())
   if res['predicted_label'][0][0] == 0:
       label = 'Ok'
else:
```

```
        label = 'Spam'     score =
round(res['predicted_probability'][0][0], 4)     score =
score*100
  message = "We received your email sent at " + str(email_obj.get('To')) + " with the subject
" + str(email_obj.get('Subject')) + ".\nHere \ is a 240 character sample of the email
body:\n\n" + body[:240] + "\nThe email was \ categorized as " + str(label) + " with
a " + str(score) + "% confidence."
   email_client = session.client('ses')
   response_email = email_client.send_email(
       Destination={'ToAddresses': [from_email]},
       Message={
          'Body': {
             'Text': {
                'Charset': 'UTF-8',
                'Data': message,
             },
          },
          'Subject':  {
             'Charset': 'UTF-8',
             'Data': 'Spam analysis of your email',
          },
       },
       Source=str(email_obj.get('To')),
    )
```

print(response_email)

return {}





CONCLUSION: -

Spam detection system built on AWS cloud, that upon receipt of an email message, automatically flags it as spam or not. This is based on the prediction obtained from the machine learning model created using Amazon SageMaker. The definition and provision of the resources on AWS cloud is done through the AWS Cloudformation template

References

- Amazon SageMaker
- Machine Learning model using Amazon SageMaker
- Build and Train a spam filter Machine Learning Model