SkipQ

Sameer Jehan

Github Username: sameerjehan2021skipq

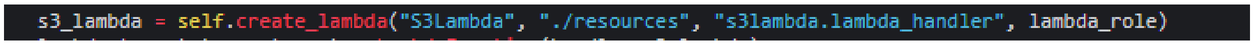
Table of Contents

1. Objective

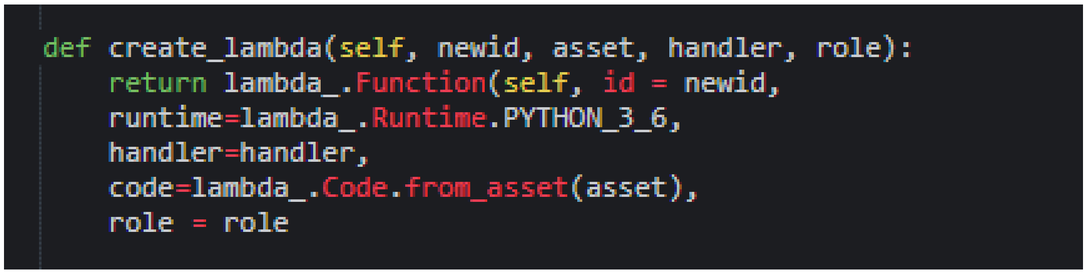
To monitor availability and latency of different websites by using CloudWatch Metrics and sending notification to user when threshold is breached.

2. Implementation

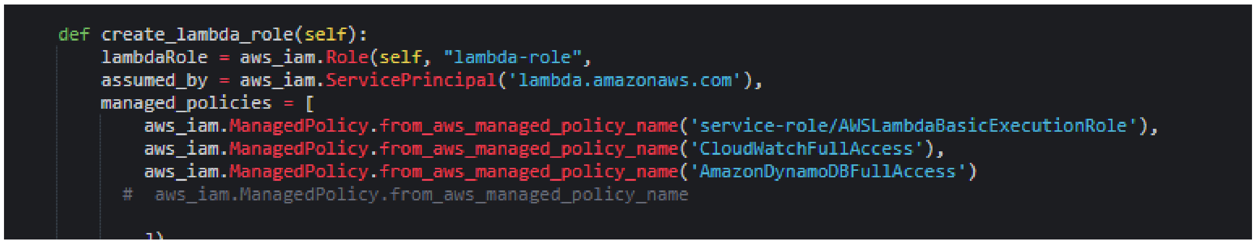
2.1. Defining roles and how to create lambda. Example of S3 Lambda



```
s3_lambda = self.create_lambda("S3Lambda", "./resources", "s3lambda.lambda_handler", lambda_role)
```



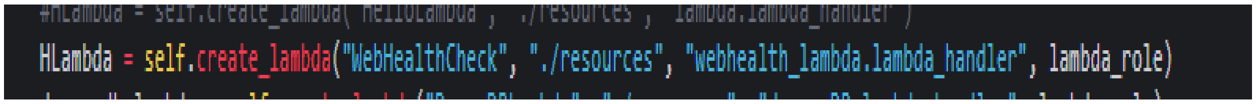
```
def create_lambda(self, newid, asset, handler, role):
    return lambda_.Function(self, id = newid,
    runtime=lambda_.Runtime.PYTHON_3_6,
    handler=handler,
    code=lambda_.Code.from_asset(asset),
    role = role
```



```
def create_lambda_role(self):
    lambdaRole = aws_iam.Role(self, "lambda-role",
    assumed_by = aws_iam.ServicePrincipal('lambda.amazonaws.com'),
    managed_policies = [
        aws_iam.ManagedPolicy.from_aws_managed_policy_name('service-role/AWSLambdaBasicExecutionRole'),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name('CloudWatchFullAccess'),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name('AmazonDynamoDBFullAccess')
      #  aws_iam.ManagedPolicy.from_aws_managed_policy_name

        ])
```

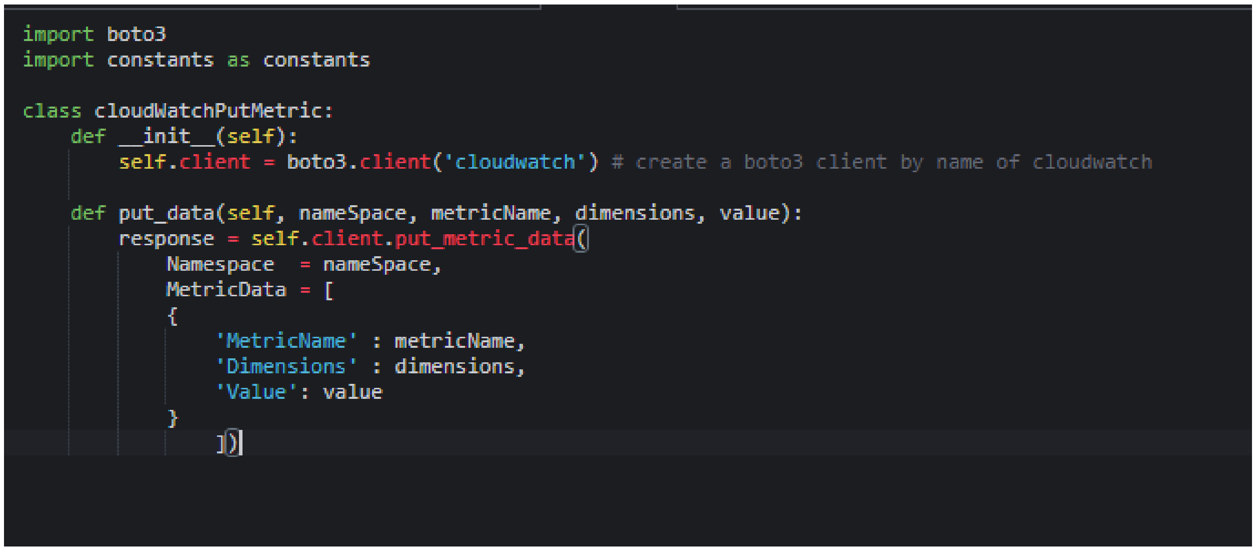
2.2 Write a lambda function that measures availability and latency

Calling the lambda handler



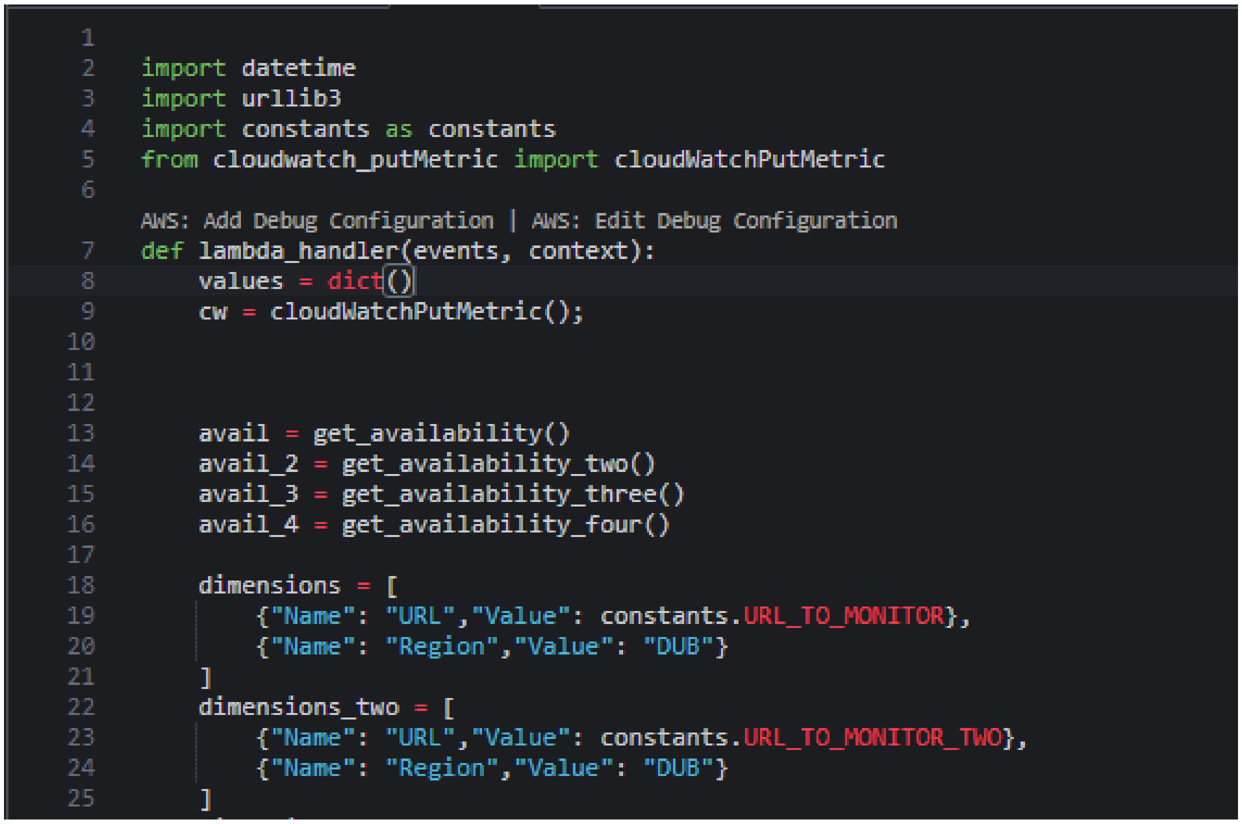
```
#HLambda = self.create_lambda("HelloLambda", "./resources", "lambda.lambda_handler")
HLambda = self.create_lambda("WebHealthCheck", "./resources", "webhealth_lambda.lambda_handler", lambda_role)
```

First we defined a class which will be used to create a namespace for publishing the metrics.



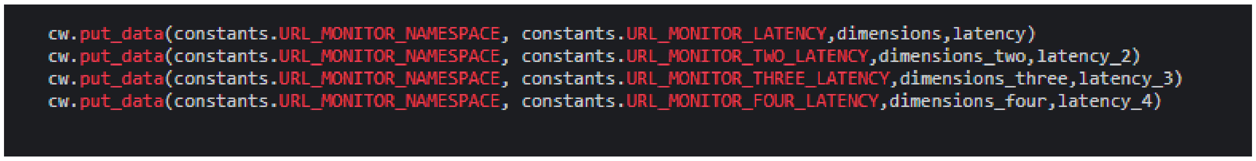
```python
import boto3
import constants as constants

class cloudWatchPutMetric:
    def __init__(self):
        self.client = boto3.client('cloudwatch') # create a boto3 client by name of cloudwatch

    def put_data(self, nameSpace, metricName, dimensions, value):
        response = self.client.put_metric_data(
            Namespace  = nameSpace,
            MetricData = [
            {
                'MetricName' : metricName,
                'Dimensions' : dimensions,
                'Value': value
            }
                ])
```

Than, what I have done is basically I wasn’t able to found a way around for using to manage 4 urls so I used separate dimension and had values for each of URL to monitors.



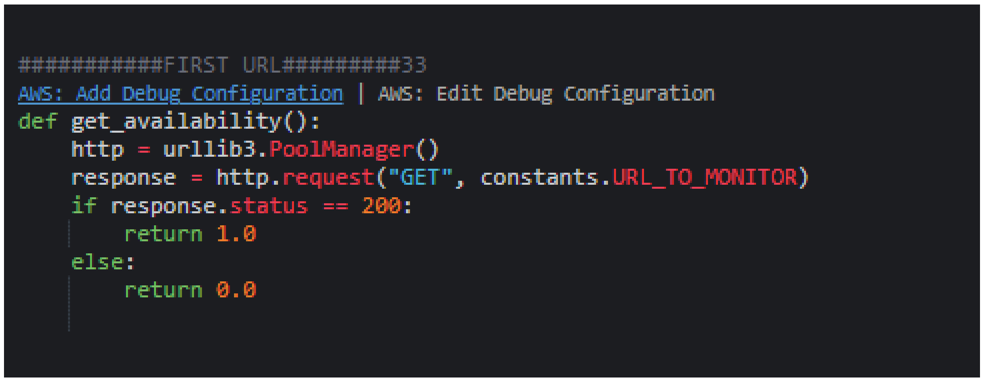
```python

import datetime
import urllib3
import constants as constants
from cloudwatch_putMetric import cloudWatchPutMetric

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(events, context):
    values = dict()
    cw = cloudWatchPutMetric();



    avail = get_availability()
    avail_2 = get_availability_two()
    avail_3 = get_availability_three()
    avail_4 = get_availability_four()

    dimensions = [
        {"Name": "URL","Value": constants.URL_TO_MONITOR},
        {"Name": "Region","Value": "DUB"}
    ]
    dimensions_two = [
        {"Name": "URL","Value": constants.URL_TO_MONITOR_TWO},
        {"Name": "Region","Value": "DUB"}
    ]
```

Putting Data into cloudwatch metrics

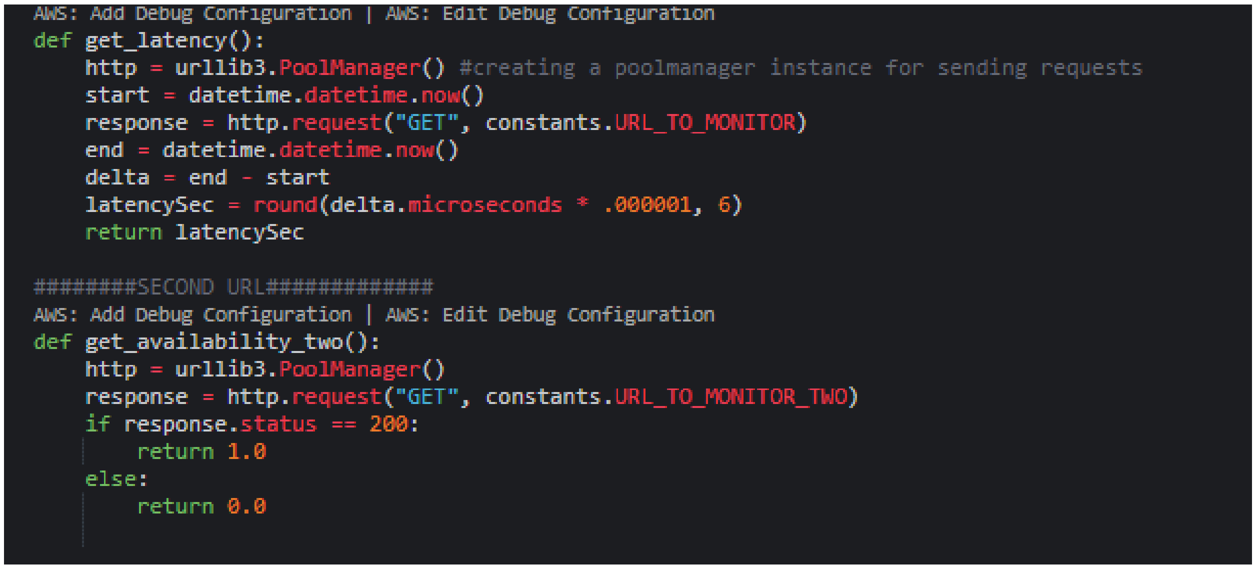


```python
cw.put_data(constants.URL_MONITOR_NAMESPACE, constants.URL_MONITOR_LATENCY,dimensions,latency)
cw.put_data(constants.URL_MONITOR_NAMESPACE, constants.URL_MONITOR_TWO_LATENCY,dimensions_two,latency_2)
cw.put_data(constants.URL_MONITOR_NAMESPACE, constants.URL_MONITOR_THREE_LATENCY,dimensions_three,latency_3)
cw.put_data(constants.URL_MONITOR_NAMESPACE, constants.URL_MONITOR_FOUR_LATENCY,dimensions_four,latency_4)
```

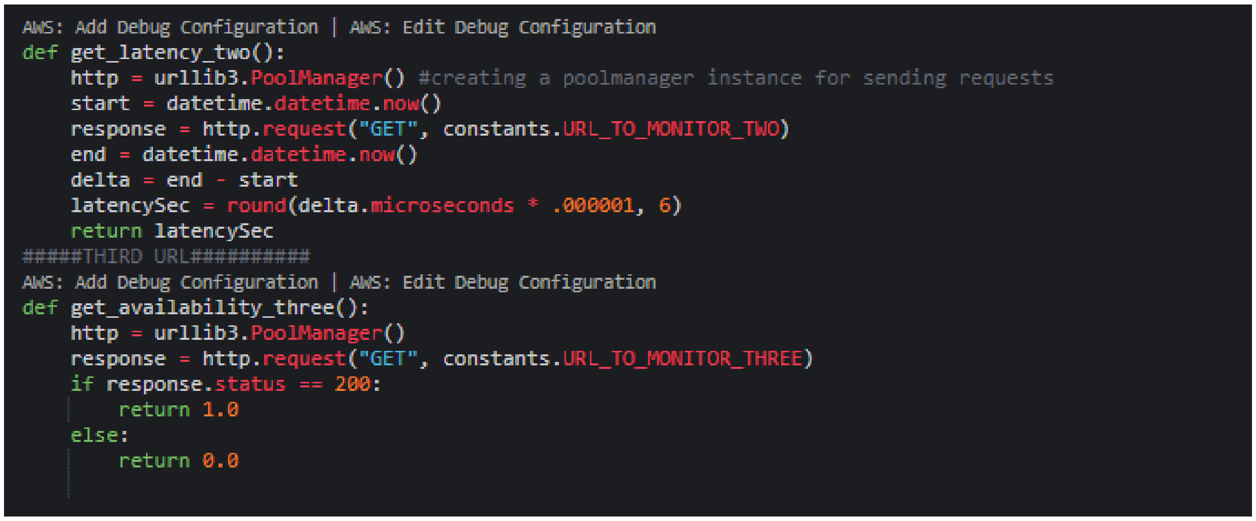
Setting latency and availability for four urls



```
###########FIRST URL#########33
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_availability():
    http = urllib3.PoolManager()
    response = http.request("GET", constants.URL_TO_MONITOR)
    if response.status == 200:
        return 1.0
    else:
        return 0.0
```

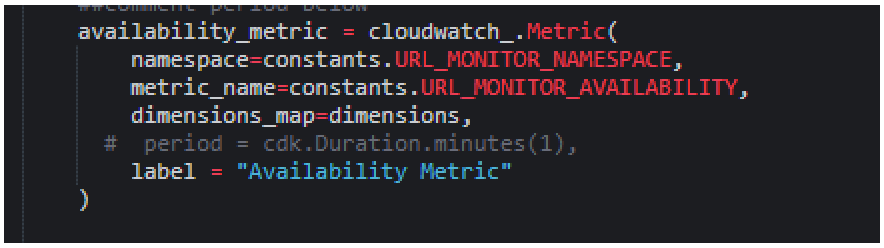


```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_latency():
    http = urllib3.PoolManager() #creating a poolmanager instance for sending requests
    start = datetime.datetime.now()
    response = http.request("GET", constants.URL_TO_MONITOR)
    end = datetime.datetime.now()
    delta = end - start
    latencySec = round(delta.microseconds * .000001, 6)
    return latencySec

########SECOND URL#############
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_availability_two():
    http = urllib3.PoolManager()
    response = http.request("GET", constants.URL_TO_MONITOR_TWO)
    if response.status == 200:
        return 1.0
    else:
        return 0.0
```

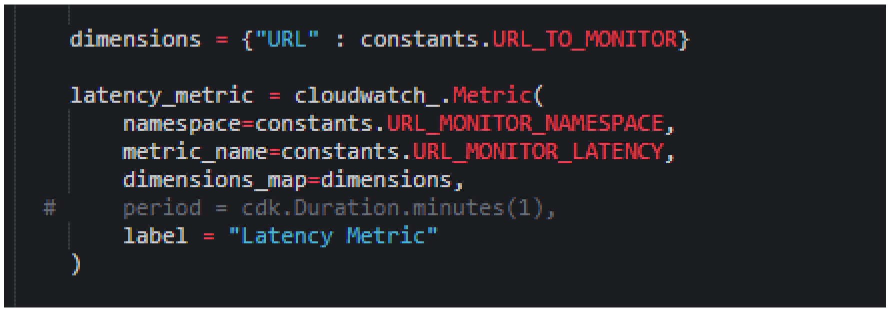


```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_latency_two():
    http = urllib3.PoolManager() #creating a poolmanager instance for sending requests
    start = datetime.datetime.now()
    response = http.request("GET", constants.URL_TO_MONITOR_TWO)
    end = datetime.datetime.now()
    delta = end - start
    latencySec = round(delta.microseconds * .000001, 6)
    return latencySec
#####THIRD URL##########
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_availability_three():
    http = urllib3.PoolManager()
    response = http.request("GET", constants.URL_TO_MONITOR_THREE)
    if response.status == 200:
        return 1.0
    else:
        return 0.0
```

2.3 Schedule the lambda to run for every minutes

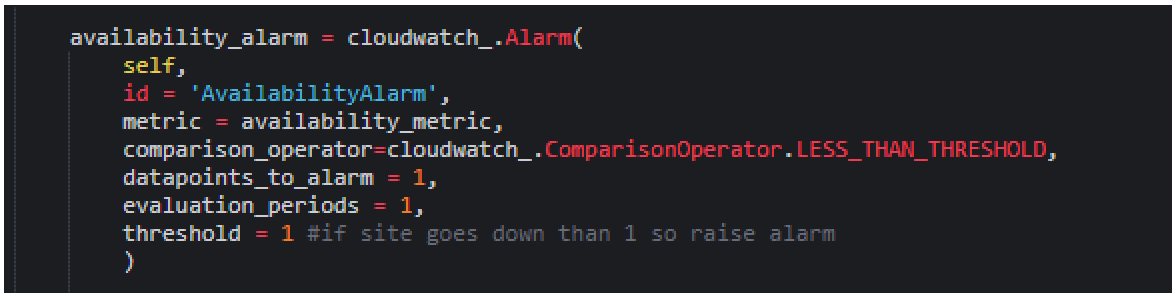


```
    availability_metric = cloudwatch_.Metric(
        namespace=constants.URL_MONITOR_NAMESPACE,
        metric_name=constants.URL_MONITOR_AVAILABILITY,
        dimensions_map=dimensions,
      #  period = cdk.Duration.minutes(1),
        label = "Availability Metric"
    )
```

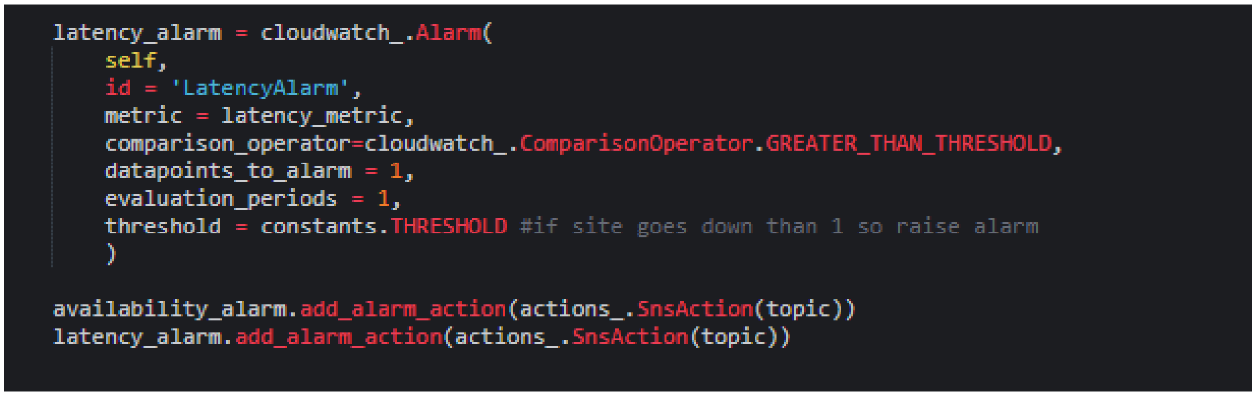


```
    dimensions = {"URL" : constants.URL_TO_MONITOR}

    latency_metric = cloudwatch_.Metric(
        namespace=constants.URL_MONITOR_NAMESPACE,
        metric_name=constants.URL_MONITOR_LATENCY,
        dimensions_map=dimensions,
#       period = cdk.Duration.minutes(1),
        label = "Latency Metric"
    )
```

2.4 Setting alarm on a certain threshold

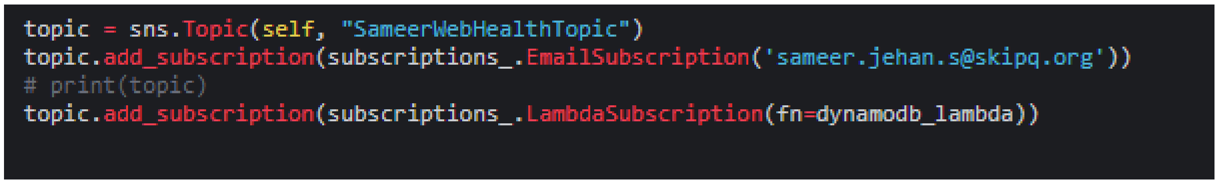


```
    availability_alarm = cloudwatch_.Alarm(
        self,
        id = 'AvailabilityAlarm',
        metric = availability_metric,
        comparison_operator=cloudwatch_.ComparisonOperator.LESS_THAN_THRESHOLD,
        datapoints_to_alarm = 1,
        evaluation_periods = 1,
        threshold = 1 #if site goes down than 1 so raise alarm
        )
```



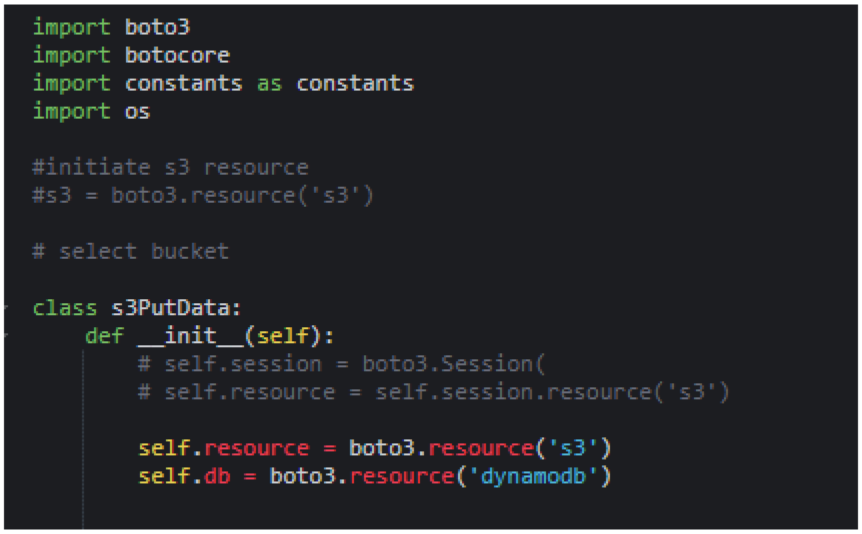
```
    latency_alarm = cloudwatch_.Alarm(
        self,
        id = 'LatencyAlarm',
        metric = latency_metric,
        comparison_operator=cloudwatch_.ComparisonOperator.GREATER_THAN_THRESHOLD,
        datapoints_to_alarm = 1,
        evaluation_periods = 1,
        threshold = constants.THRESHOLD #if site goes down than 1 so raise alarm
        )

    availability_alarm.add_alarm_action(actions_.SnsAction(topic))
    latency_alarm.add_alarm_action(actions_.SnsAction(topic))
```

2.5 Sending notification if alarm has breached threshold



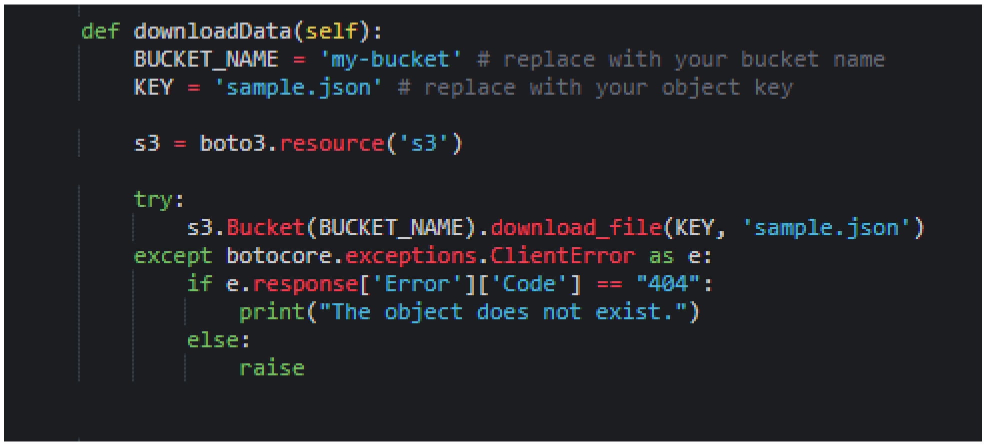
```
topic = sns.Topic(self, "SameerWebHealthTopic")
topic.add_subscription(subscriptions_.EmailSubscription('sameer.jehan.s@skipq.org'))
# print(topic)
topic.add_subscription(subscriptions_.LambdaSubscription(fn=dynamodb_lambda))
```

2.6 Create S3 Bucket for custom list of websites



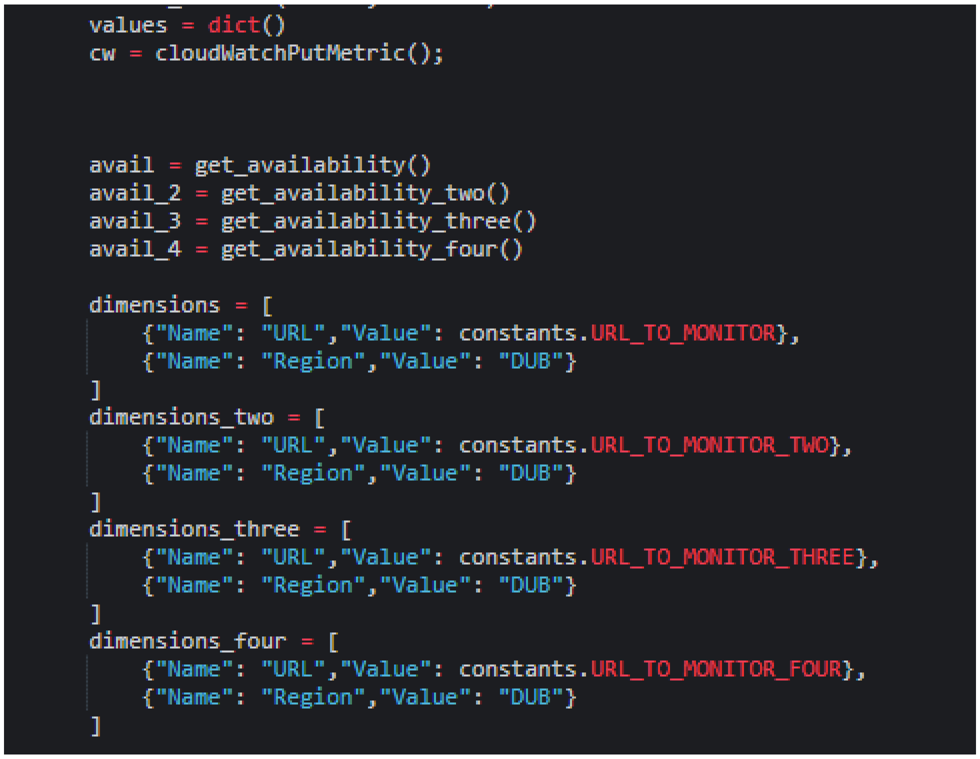
```python
import boto3
import botocore
import constants as constants
import os

#initiate s3 resource
#s3 = boto3.resource('s3')

# select bucket

class s3PutData:
    def __init__(self):
        # self.session = boto3.Session(
        # self.resource = self.session.resource('s3')

        self.resource = boto3.resource('s3')
        self.db = boto3.resource('dynamodb')
```

2.7 Read and download from S3 Bucket

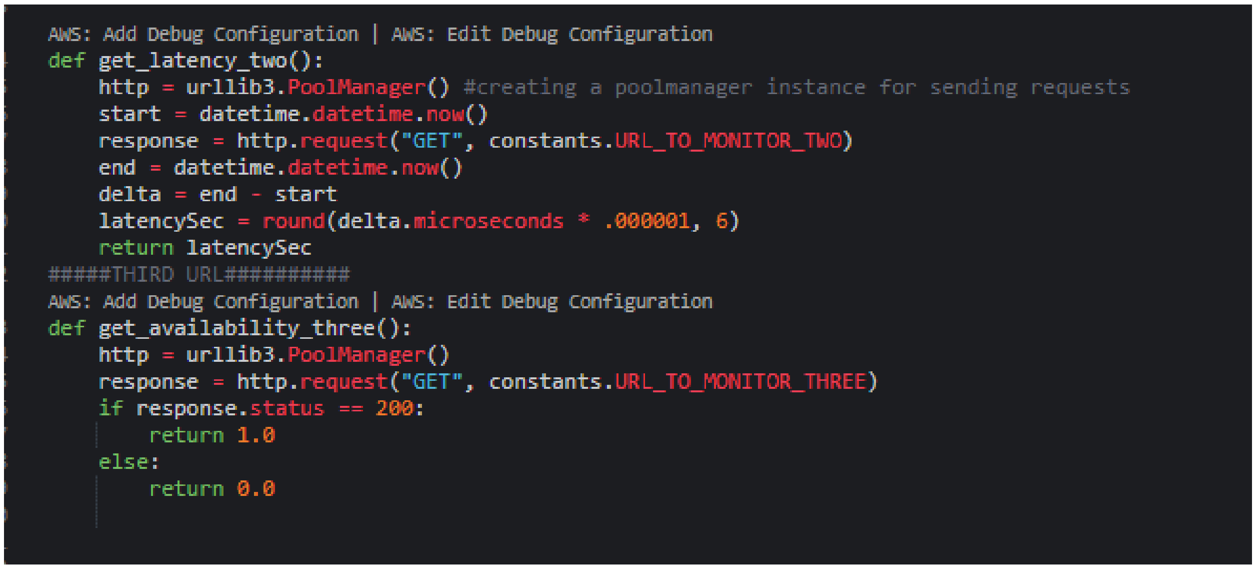


```python
    def downloadData(self):
        BUCKET_NAME = 'my-bucket' # replace with your bucket name
        KEY = 'sample.json' # replace with your object key

        s3 = boto3.resource('s3')

        try:
            s3.Bucket(BUCKET_NAME).download_file(KEY, 'sample.json')
        except botocore.exceptions.ClientError as e:
            if e.response['Error']['Code'] == "404":
                print("The object does not exist.")
            else:
                raise
```

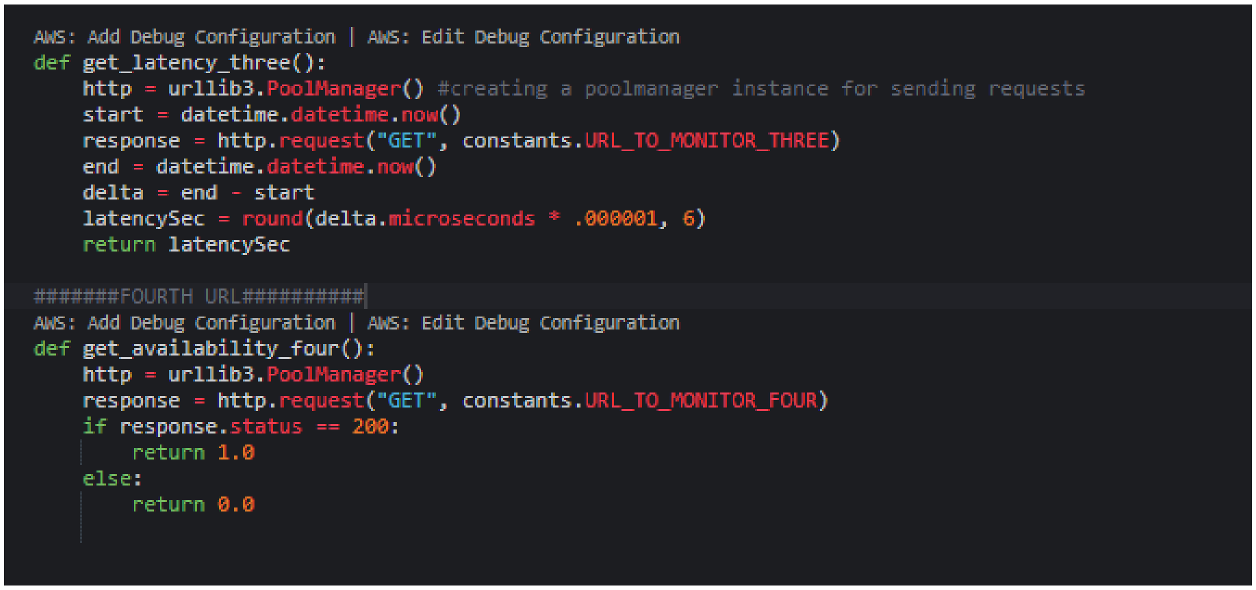
2.8 Monitor health of four urls



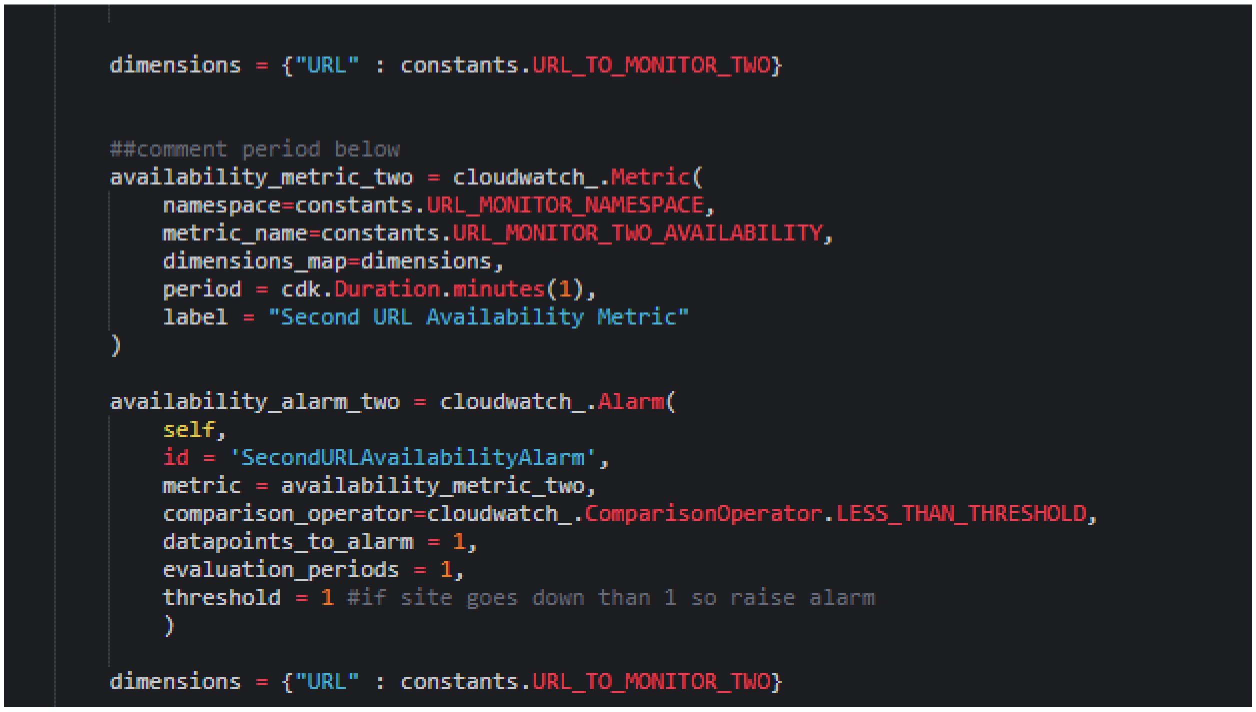
```
    values = dict()
    cw = cloudWatchPutMetric();



    avail = get_availability()
    avail_2 = get_availability_two()
    avail_3 = get_availability_three()
    avail_4 = get_availability_four()

    dimensions = [
        {"Name": "URL","Value": constants.URL_TO_MONITOR},
        {"Name": "Region","Value": "DUB"}
    ]
    dimensions_two = [
        {"Name": "URL","Value": constants.URL_TO_MONITOR_TWO},
        {"Name": "Region","Value": "DUB"}
    ]
    dimensions_three = [
        {"Name": "URL","Value": constants.URL_TO_MONITOR_THREE},
        {"Name": "Region","Value": "DUB"}
    ]
    dimensions_four = [
        {"Name": "URL","Value": constants.URL_TO_MONITOR_FOUR},
        {"Name": "Region","Value": "DUB"}
    ]
```



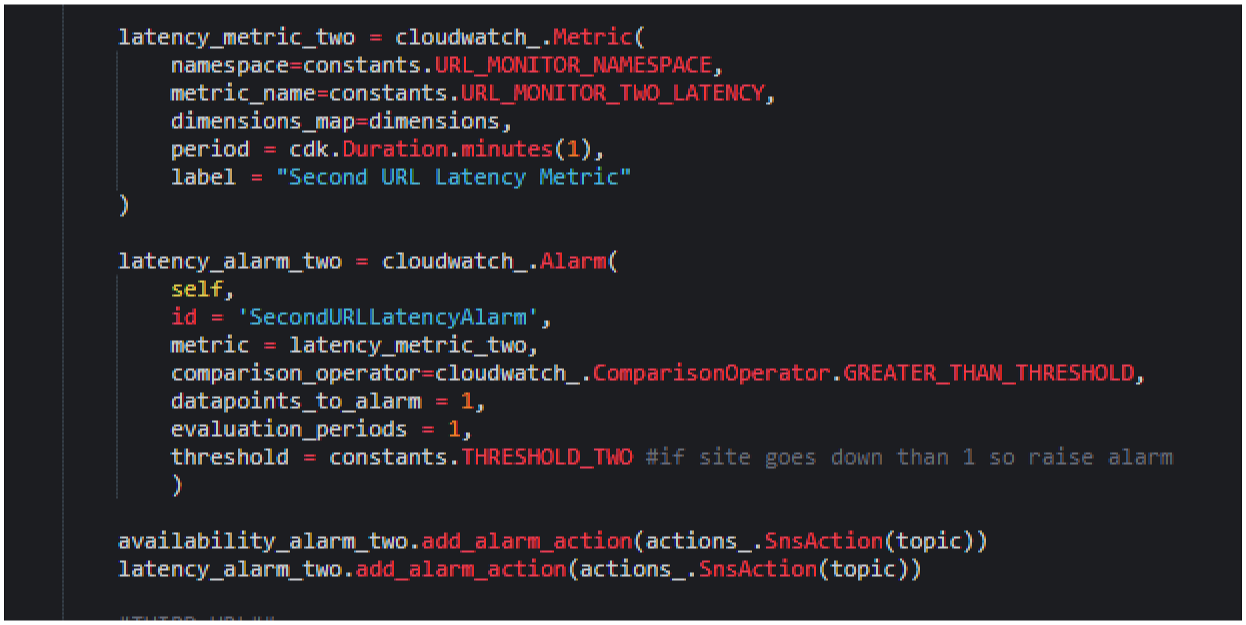
```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_latency_two():
    http = urllib3.PoolManager() #creating a poolmanager instance for sending requests
    start = datetime.datetime.now()
    response = http.request("GET", constants.URL_TO_MONITOR_TWO)
    end = datetime.datetime.now()
    delta = end - start
    latencySec = round(delta.microseconds * .000001, 6)
    return latencySec
#####THIRD URL##########
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_availability_three():
    http = urllib3.PoolManager()
    response = http.request("GET", constants.URL_TO_MONITOR_THREE)
    if response.status == 200:
        return 1.0
    else:
        return 0.0
```



```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_latency_three():
    http = urllib3.PoolManager() #creating a poolmanager instance for sending requests
    start = datetime.datetime.now()
    response = http.request("GET", constants.URL_TO_MONITOR_THREE)
    end = datetime.datetime.now()
    delta = end - start
    latencySec = round(delta.microseconds * .000001, 6)
    return latencySec

#######FOURTH URL##########
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_availability_four():
    http = urllib3.PoolManager()
    response = http.request("GET", constants.URL_TO_MONITOR_FOUR)
    if response.status == 200:
        return 1.0
    else:
        return 0.0
```

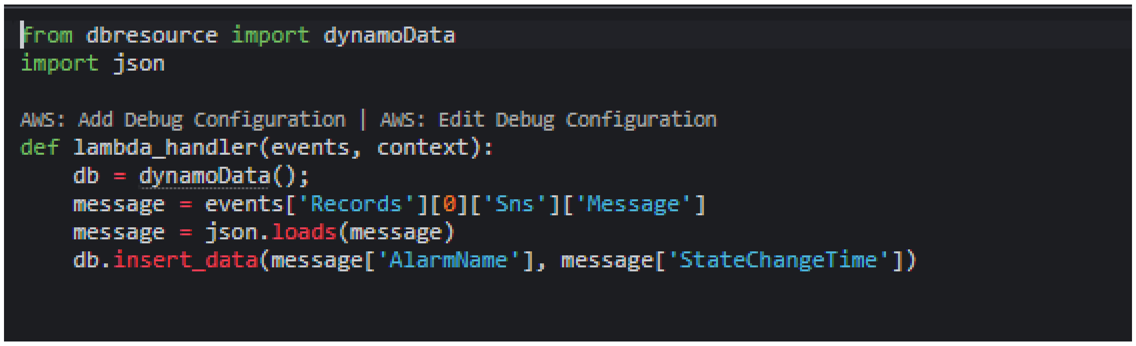


```
        dimensions = {"URL" : constants.URL_TO_MONITOR_TWO}


        ##comment period below
        availability_metric_two = cloudwatch_.Metric(
            namespace=constants.URL_MONITOR_NAMESPACE,
            metric_name=constants.URL_MONITOR_TWO_AVAILABILITY,
            dimensions_map=dimensions,
            period = cdk.Duration.minutes(1),
            label = "Second URL Availability Metric"
        )

        availability_alarm_two = cloudwatch_.Alarm(
            self,
            id = 'SecondURLAvailabilityAlarm',
            metric = availability_metric_two,
            comparison_operator=cloudwatch_.ComparisonOperator.LESS_THAN_THRESHOLD,
            datapoints_to_alarm = 1,
            evaluation_periods = 1,
            threshold = 1 #if site goes down than 1 so raise alarm
            )

        dimensions = {"URL" : constants.URL_TO_MONITOR_TWO}
```

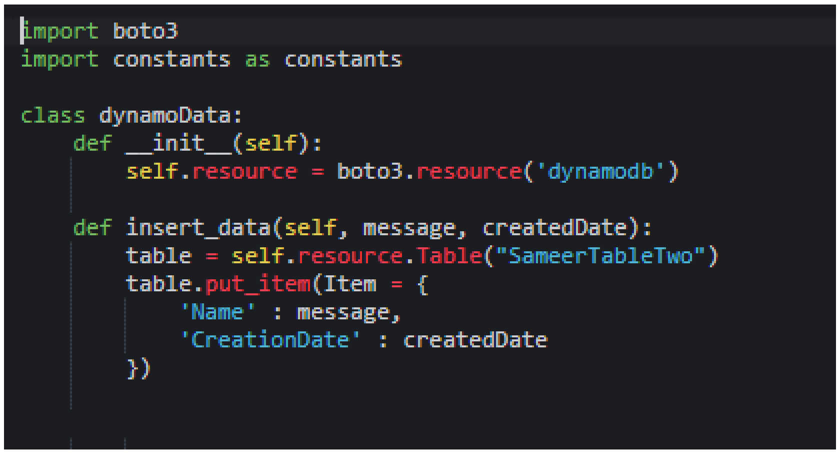


```
        latency_metric_two = cloudwatch_.Metric(
            namespace=constants.URL_MONITOR_NAMESPACE,
            metric_name=constants.URL_MONITOR_TWO_LATENCY,
            dimensions_map=dimensions,
            period = cdk.Duration.minutes(1),
            label = "Second URL Latency Metric"
        )

        latency_alarm_two = cloudwatch_.Alarm(
            self,
            id = 'SecondURLLatencyAlarm',
            metric = latency_metric_two,
            comparison_operator=cloudwatch_.ComparisonOperator.GREATER_THAN_THRESHOLD,
            datapoints_to_alarm = 1,
            evaluation_periods = 1,
            threshold = constants.THRESHOLD_TWO #if site goes down than 1 so raise alarm
            )

        availability_alarm_two.add_alarm_action(actions_.SnsAction(topic))
        latency_alarm_two.add_alarm_action(actions_.SnsAction(topic))
```

2.9 DynamoDB Handler

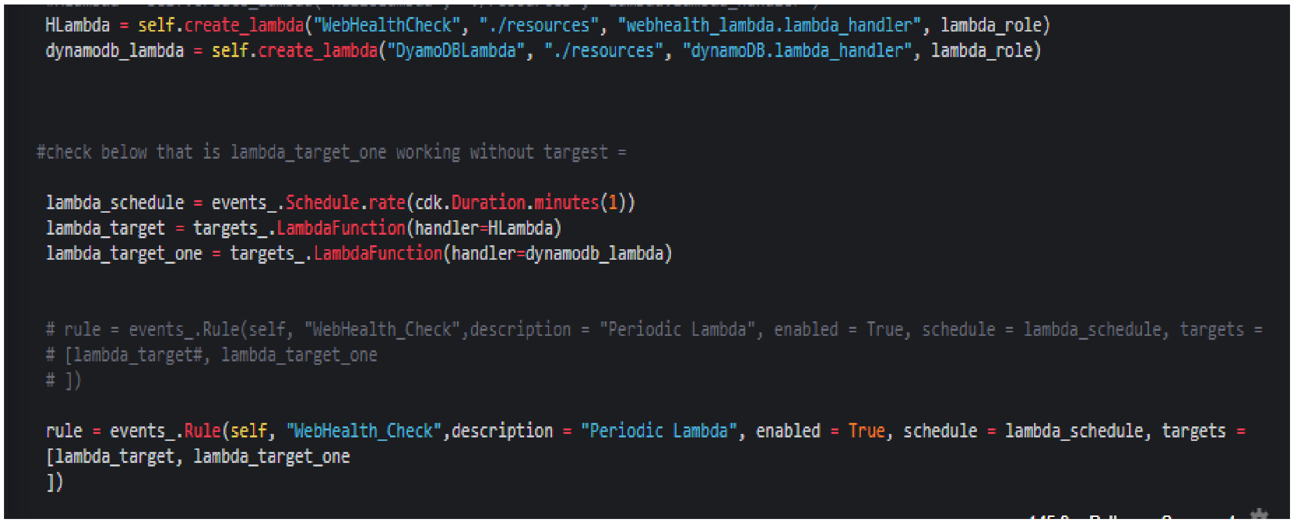


```
from dbresource import dynamoData
import json

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(events, context):
    db = dynamoData();
    message = events['Records'][0]['Sns']['Message']
    message = json.loads(message)
    db.insert_data(message['AlarmName'], message['StateChangeTime'])
```



```
import boto3
import constants as constants

class dynamoData:
    def __init__(self):
        self.resource = boto3.resource('dynamodb')

    def insert_data(self, message, createdDate):
        table = self.resource.Table("SameerTableTwo")
        table.put_item(Item = {
            'Name' : message,
            'CreationDate' : createdDate
        })
```

I have also linked periodic to dynamodb to invoke it after duration of 1 minute



```
 HLambda = self.create_lambda("WebHealthCheck", "./resources", "webhealth_lambda.lambda_handler", lambda_role)
 dynamodb_lambda = self.create_lambda("DyamoDBLambda", "./resources", "dynamoDB.lambda_handler", lambda_role)



#check below that is lambda_target_one working without targest =

 lambda_schedule = events_.Schedule.rate(cdk.Duration.minutes(1))
 lambda_target = targets_.LambdaFunction(handler=HLambda)
 lambda_target_one = targets_.LambdaFunction(handler=dynamodb_lambda)


 # rule = events_.Rule(self, "WebHealth_Check",description = "Periodic Lambda", enabled = True, schedule = lambda_schedule, targets =
 # [lambda_target#, lambda_target_one
 # ])

 rule = events_.Rule(self, "WebHealth_Check",description = "Periodic Lambda", enabled = True, schedule = lambda_schedule, targets =
 [lambda_target, lambda_target_one
 ])
```