SkipQ

Proxima Centauri

Documentation

Written by:

Muhammad Irfan Hassan

Trainee @SkipQ2021

Table of Content

Sprint1: Web Health Monitor Application

Project Description

This project aims to measure the availability and latency of a custom list (a JSON file placed in an s3 bucket) using AWS CDK. It will update latency and availability after each 1 minute and will write metrics for latency and availability on cloud watch using cloud watch's API. Also, set an alarm to notify the subscriber when the threshold for latency and availability is preached. Push SNS notification to subscribers using the email address and also trigger lambda and store alarm data into dynamo dB when alarm generated.

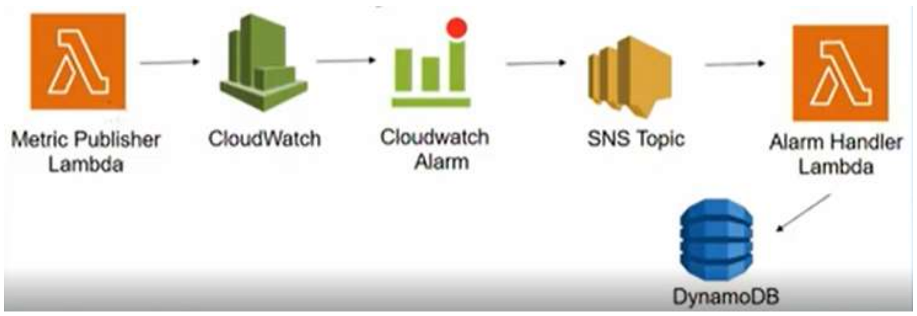


Figure 1: Web Health Monitor CDK Application

Technologies/Services

To build this application we will use the following AWS services

- Cloud9
- Lambda
- Cloud Watch
- SNS
- Dynamo DB

Cloud9

AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. Cloud9 comes prepackaged with an essential tool for popular programming languages, including JavaScript, Python, PHP, and more, don't need to install files or configure a development machine to start the project.

Lambda

Lambda is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring, and logging. With Lambda, you can run code for virtually any type of application or backend service.

Cloud Watch

Amazon Cloud Watch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real-time. You can use Cloud Watch to collect and track metrics, which are variables you can measure for your resources and applications. Cloud Watch's home page automatically displays metrics about every AWS service you use. You can also create custom dashboards to display metrics about your custom applications and display custom collections of metrics you choose.

You can create alarms that watch metrics and send notifications or automatically make changes to the resources you are monitoring when a threshold is breached.

SNS

Amazon Simple Notification Service (Amazon SNS) is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication. The A2A pub/sub functionality provides topics for high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. Using Amazon SNS topics, your publisher systems can fan-out messages to a large number of subscriber systems, including Amazon SQS queues, AWS Lambda functions, HTTPS endpoints, and Amazon Kinesis Data Firehose, for parallel processing. The A2P functionality enables you to send messages to users at scale via SMS, mobile push, and email.

Dynamo DB

Amazon Dynamo DB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. Dynamo DB offers built-in security, continuous backups, automated multi-region replication, in-memory caching, and data export tools.

Setup

Before starting the project, we have to set up the environment and install requirements for the project. The steps for setup are following.

- First of all, log in to AWS amazon and create a virtual machine.
- check the version of python and if it is an old version check new version is available then make a new version as the default version.

```
python --version
python3 --version
source ~/.bashrc
alis python='/usr/bin/python3' (press ESC on keyboard)
:w! (press Enter on keyboard)
:q! (press Enter on keyboard)
```

- check the version of AWS and if it is an old version then update it to the new version.

```
aws --version
curl       "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"       -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

- create a directory of your choice and change the directory to new created

```
mkdir IrfanskipQ_Project1
cd IrfanskipQ_Project1
```

- Create CDK project in python language

```
cdk init app --language python
```

- Install all requirements for the project.

```
python -m pip install aws-cdk.core==1.135.0
python -m pip install -r requirements.txt
nvm install v16.3.0 && nvm use 16.3.0 && nvm alias default v16.3.0
npm install -g aws-cdk
export PATH=$PATH:$(npm get prefix)/bin
python -m pip install aws-cdk.aws-s3 aws-cdk.aws-lambda
```

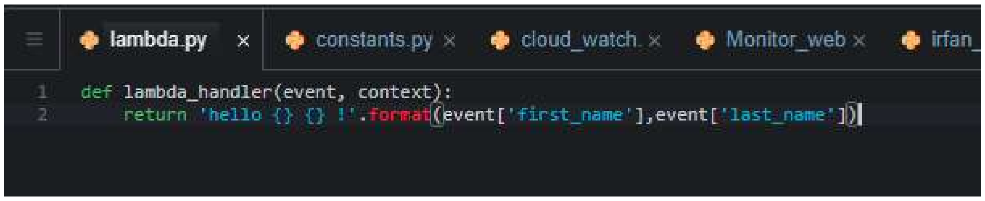
- Create a new folder resource and add a new lambda.py file.

Project milestones

I have divided my project into subtasks and then completed subtasks on daily basis. Let’s discuss each subtask in detail.

Task 01: Implementing Hello Lambda Function:

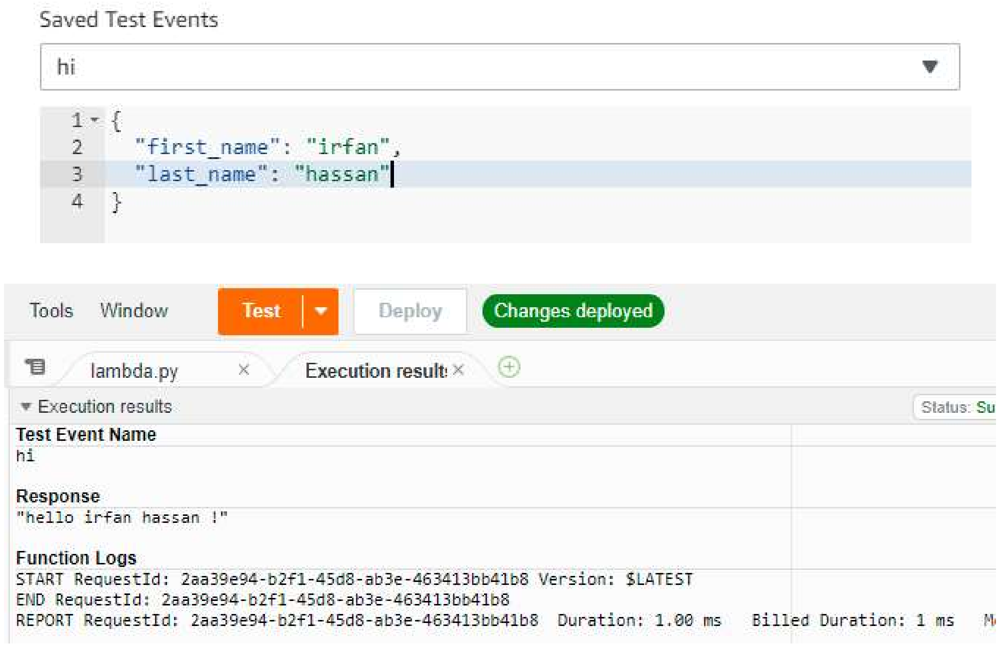
First I started with the Hello Lambda function using the lambda handler. This function takes two strings as input “first name” and “last-named”. code for Hello Lambda function is given below.



```
def lambda_handler(event, context):
    return 'hello {} {} !'.format(event['first_name'],event['last_name'])
```

Figure 2: Hello Lambda function

To test this function, I put my first name and last name, and here is the output of this function.



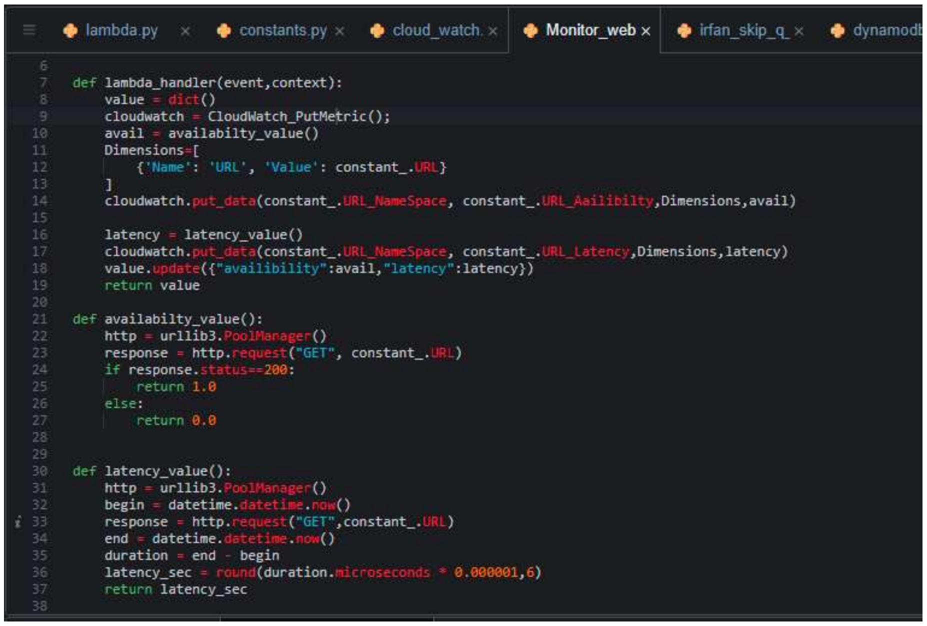
```
{
  "first_name": "irfan",
  "last_name": "hassan"
}
```

```
Test Event Name
hi

Response
"hello irfan hassan !"

Function Logs
START RequestId: 2aa39e94-b2f1-45d8-ab3e-463413bb41b8 Version: $LATEST
END RequestId: 2aa39e94-b2f1-45d8-ab3e-463413bb41b8
REPORT RequestId: 2aa39e94-b2f1-45d8-ab3e-463413bb41b8  Duration: 1.00 ms   Billed Duration: 1 ms   M
```

Figure 3:Testing hellolambda

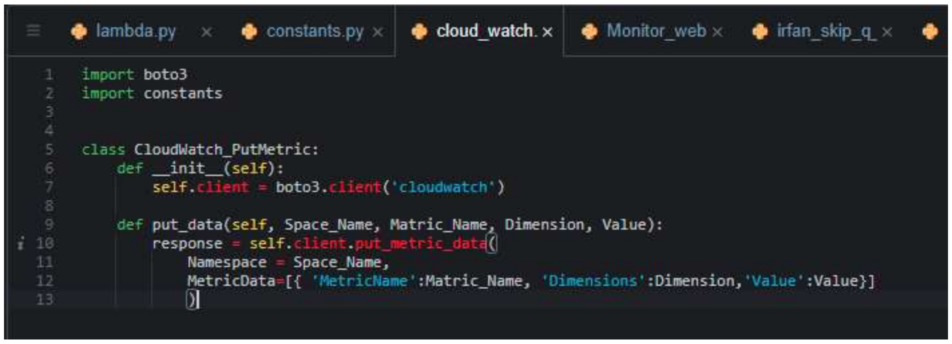
Task 02: Availability and latency of webpage by using periodic lambda function

Then we create another lambda function, which triggers after 1 minute and checks the availability and latency of the webpage (URL will be given), and stores the metric for latency and availability on cloud watch.



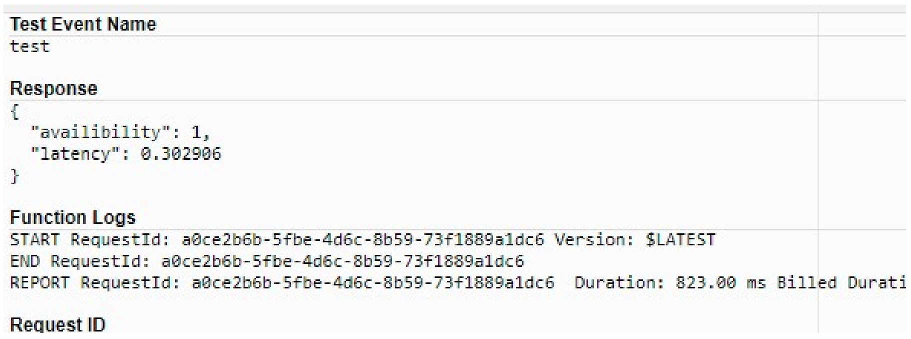
```
def lambda_handler(event,context):
    value = dict()
    cloudwatch = CloudWatch_PutMetric();
    avail = availabilty_value()
    Dimensions=[
        {'Name': 'URL', 'Value': constant_.URL}
    ]
    cloudwatch.put_data(constant_.URL_NameSpace, constant_.URL_Aailibilty,Dimensions,avail)

    latency = latency_value()
    cloudwatch.put_data(constant_.URL_NameSpace, constant_.URL_Latency,Dimensions,latency)
    value.update({"availibility":avail,"latency":latency})
    return value

def availabilty_value():
    http = urllib3.PoolManager()
    response = http.request("GET", constant_.URL)
    if response.status==200:
        return 1.0
    else:
        return 0.0


def latency_value():
    http = urllib3.PoolManager()
    begin = datetime.datetime.now()
    response = http.request("GET",constant_.URL)
    end = datetime.datetime.now()
    duration = end - begin
    latency_sec = round(duration.microseconds * 0.000001,6)
    return latency_sec
```

Figure 4: Periodic lambda for availability and latency of webpage



```
import boto3
import constants


class CloudWatch_PutMetric:
    def __init__(self):
        self.client = boto3.client('cloudwatch')

    def put_data(self, Space_Name, Matric_Name, Dimension, Value):
        response = self.client.put_metric_data(
            Namespace = Space_Name,
            MetricData=[{ 'MetricName':Matric_Name, 'Dimensions':Dimension,'Value':Value}]
            )]
```

Figure 5:putting metric on Cloud watch



Test Event Name

```
test
```

Response

```
{
  "availibility": 1,
  "latency": 0.302906
}
```

Function Logs

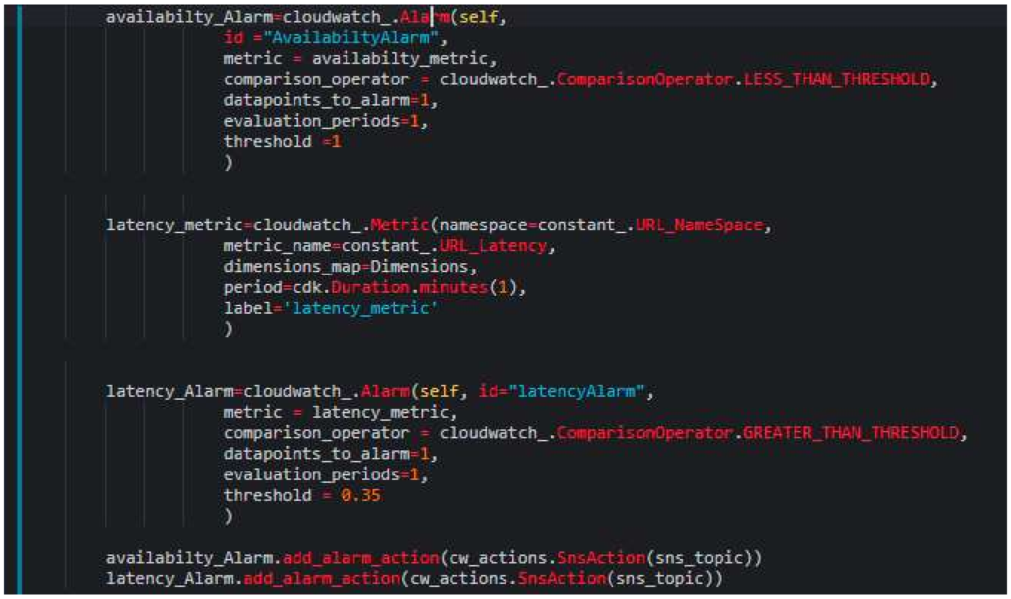
```
START RequestId: a0ce2b6b-5fbe-4d6c-8b59-73f1889a1dc6 Version: $LATEST
END RequestId: a0ce2b6b-5fbe-4d6c-8b59-73f1889a1dc6
REPORT RequestId: a0ce2b6b-5fbe-4d6c-8b59-73f1889a1dc6  Duration: 823.00 ms Billed Durati
```

Request ID

Figure 6: Latency and availability test result

Task 03: Alarm generate when threshold breached and send sns to subscribers

Then we set a threshold on metrics and generate an alarm when the threshold is breached. The alarm will notify the subscriber about the threshold breached through email notification.



```
availabilty_Alarm=cloudwatch_.Alarm(self,
            id ="AvailabiltyAlarm",
            metric = availabilty_metric,
            comparison_operator = cloudwatch_.ComparisonOperator.LESS_THAN_THRESHOLD,
            datapoints_to_alarm=1,
            evaluation_periods=1,
            threshold =1
            )

latency_metric=cloudwatch_.Metric(namespace=constant_.URL_NameSpace,
            metric_name=constant_.URL_Latency,
            dimensions_map=Dimensions,
            period=cdk.Duration.minutes(1),
            label='latency_metric'
            )

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 = 0.35
            )

availabilty_Alarm.add_alarm_action(cw_actions.SnsAction(sns_topic))
latency_Alarm.add_alarm_action(cw_actions.SnsAction(sns_topic))
```

Figure 7: Alarm for latency and availability on cloud watch

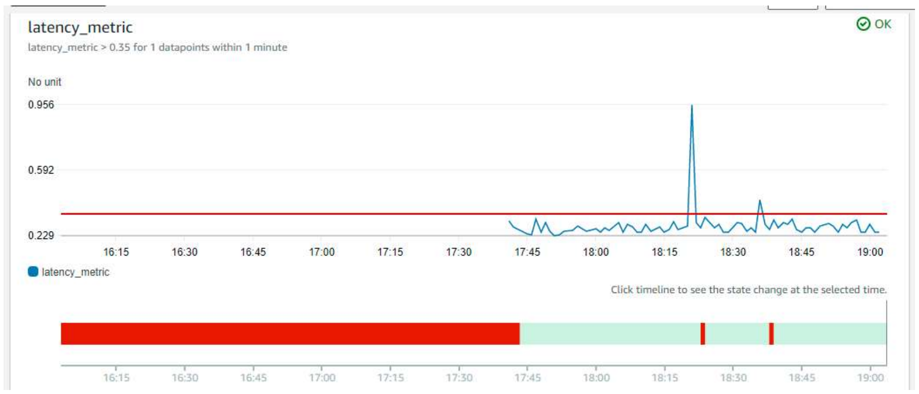
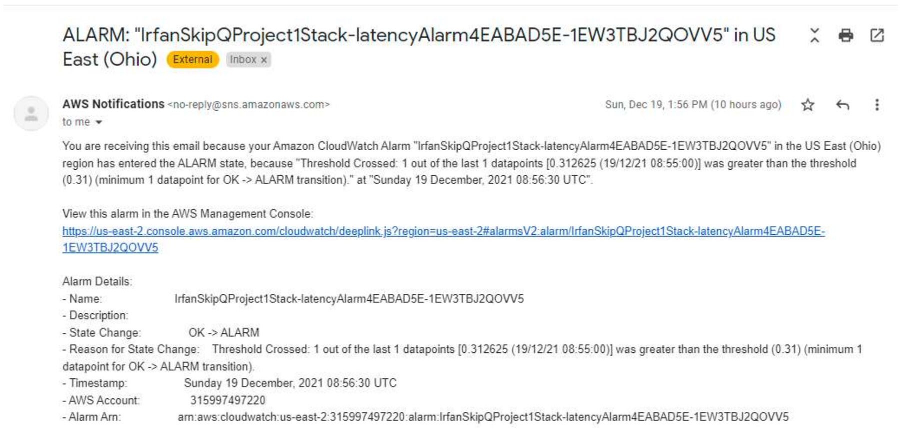


Figure 8: Alarm triggered when threshold breached



ALARM: "IrfanSkipQProject1Stack-latencyAlarm4EABAD5E-1EW3TBJ2QOVV5" in US East (Ohio) External Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> Sun, Dec 19, 1:56 PM (10 hours ago)
to me

You are receiving this email because your Amazon CloudWatch Alarm "IrfanSkipQProject1Stack-latencyAlarm4EABAD5E-1EW3TBJ2QOVV5" in the US East (Ohio) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [0.312625 (19/12/21 08:55:00)] was greater than the threshold (0.31) (minimum 1 datapoint for OK -> ALARM transition)." at "Sunday 19 December, 2021 08:56:30 UTC".

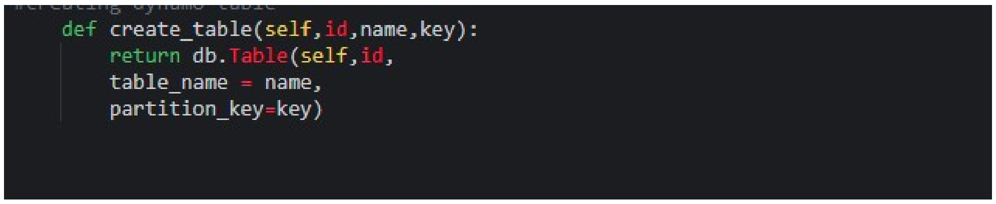
View this alarm in the AWS Management Console:
https://us-east-2.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-2#alarmsV2:alarm/IrfanSkipQProject1Stack-latencyAlarm4EABAD5E-1EW3TBJ2QOVV5

Alarm Details:
- Name: IrfanSkipQProject1Stack-latencyAlarm4EABAD5E-1EW3TBJ2QOVV5
- Description:
- State Change: OK -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [0.312625 (19/12/21 08:55:00)] was greater than the threshold (0.31) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Sunday 19 December, 2021 08:56:30 UTC
- AWS Account: 315997497220
- Alarm Arn: arn:aws:cloudwatch:us-east-2:315997497220:alarm:IrfanSkipQProject1Stack-latencyAlarm4EABAD5E-1EW3TBJ2QOVV5

Figure 9: Email notification to the subscriber

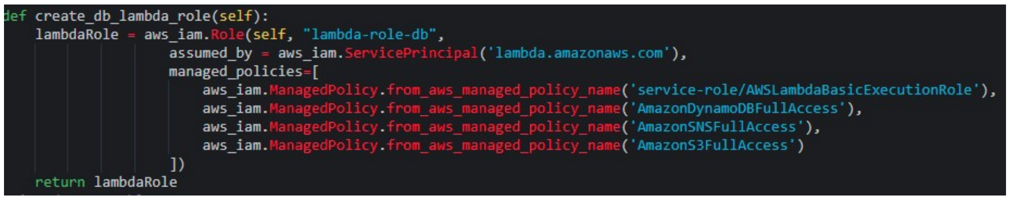
Task 04: Creating dynamo DB Table and Storing Alarm data to Table

In this task, we created a dynamo DB table and created a roll for dynamo Lambda to allow it to write in Table. Here is the code for creating a table. In the dynamo DB lambda function, we are putting alarm details in the table. Dynamo DB lambda will trigger when the alarm generates.



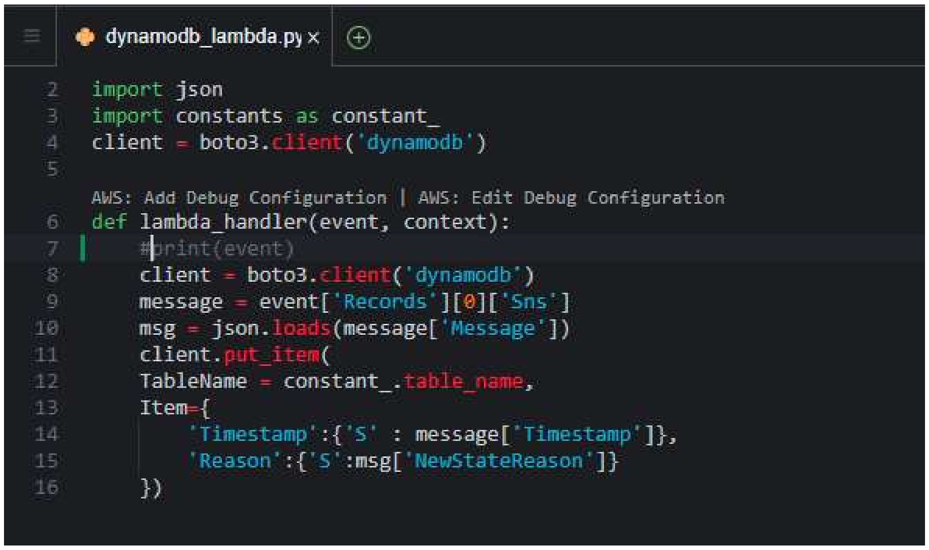
```
    def create_table(self,id,name,key):
        return db.Table(self,id,
        table_name = name,
        partition_key=key)
```

Figure 10: Dynamo DB Table creation function



```
def create_db_lambda_role(self):
    lambdaRole = aws_iam.Role(self, "lambda-role-db",
                    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('AmazonDynamoDBFullAccess'),
                        aws_iam.ManagedPolicy.from_aws_managed_policy_name('AmazonSNSFullAccess'),
                        aws_iam.ManagedPolicy.from_aws_managed_policy_name('AmazonS3FullAccess')
                    ])
    return lambdaRole
```

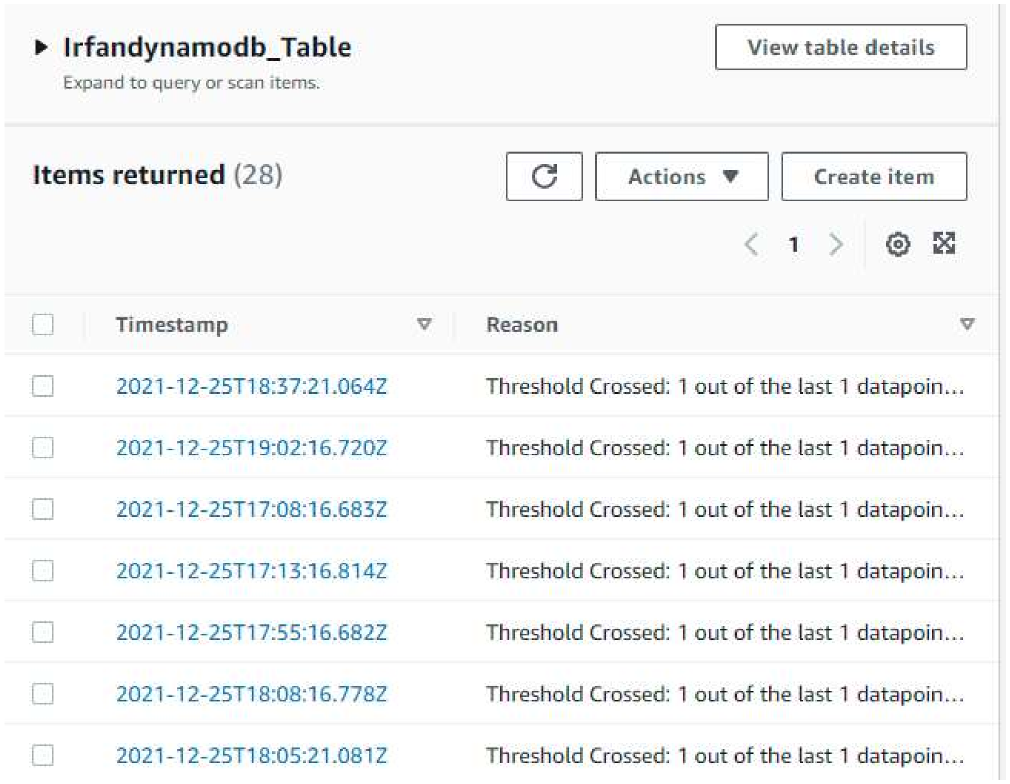
Figure 11: function for Dynamo DB lambda role



```
import json
import constants as constant_
client = boto3.client('dynamodb')

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(event, context):
    #print(event)
    client = boto3.client('dynamodb')
    message = event['Records'][0]['Sns']
    msg = json.loads(message['Message'])
    client.put_item(
    TableName = constant_.table_name,
    Item={
        'Timestamp':{'S' : message['Timestamp']},
        'Reason':{'S':msg['NewStateReason']}
    })
```

Figure 12: Dynamo DB Lambda function

Here is the table we created.

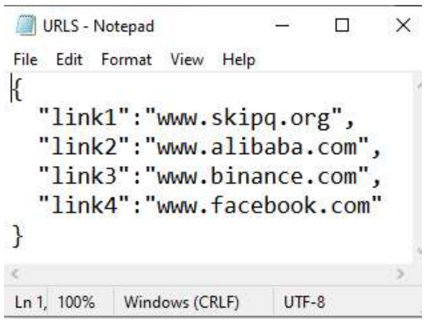


Timestamp	Reason
2021-12-25T18:37:21.064Z	Threshold Crossed: 1 out of the last 1 datapoin...
2021-12-25T19:02:16.720Z	Threshold Crossed: 1 out of the last 1 datapoin...
2021-12-25T17:08:16.683Z	Threshold Crossed: 1 out of the last 1 datapoin...
2021-12-25T17:13:16.814Z	Threshold Crossed: 1 out of the last 1 datapoin...
2021-12-25T17:55:16.682Z	Threshold Crossed: 1 out of the last 1 datapoin...
2021-12-25T18:08:16.778Z	Threshold Crossed: 1 out of the last 1 datapoin...
2021-12-25T18:05:21.081Z	Threshold Crossed: 1 out of the last 1 datapoin...

Figure 13: Alarm details in Dynamo DB Table

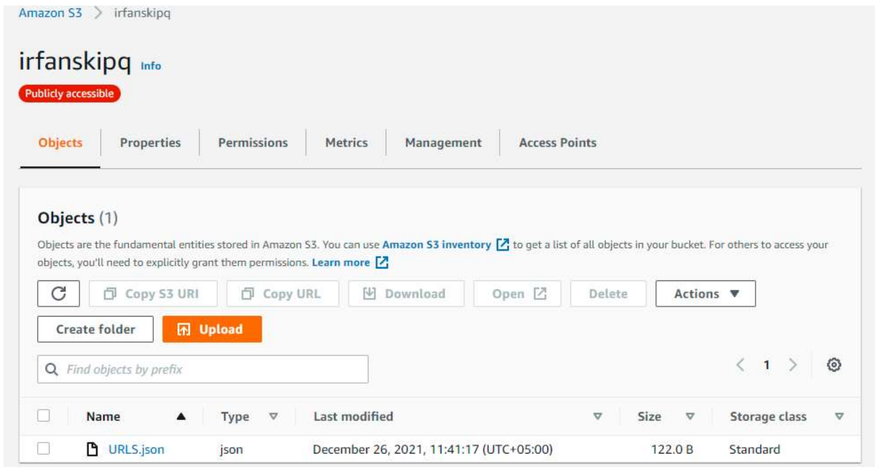
Task 05: Read JSON file from S3 bucket

Our project is to monitor the web health of custom provided webpages. We have data. Jason file has the URL of each webpage and this file is stored in AWS S3 bucket. We write another lambda function to read URLs from JSON files and store these URLs in the array.



```
{
  "link1":"www.skipq.org",
  "link2":"www.alibaba.com",
  "link3":"www.binance.com",
  "link4":"www.facebook.com"
}
```

Figure 14: URLs in the JSON file

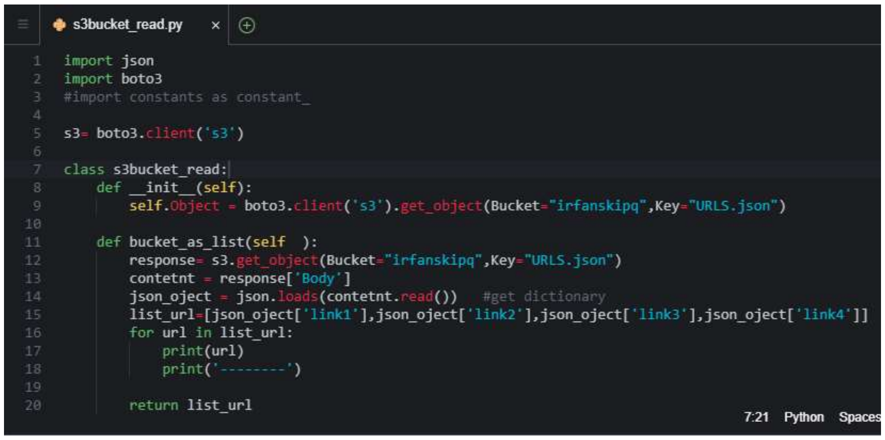

Amazon S3 > irfanskipq
irfanskipq Info
Publicly accessible
Objects Properties Permissions Metrics Management Access Points
Copy S3 URI Copy URL Download Open Delete Actions
Create folder Upload
Find objects by prefix

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
URLS.json	json	December 26, 2021, 11:41:17 (UTC+05:00)	122.0 B	Standard

Figure 15:Json file in S3 Bucket

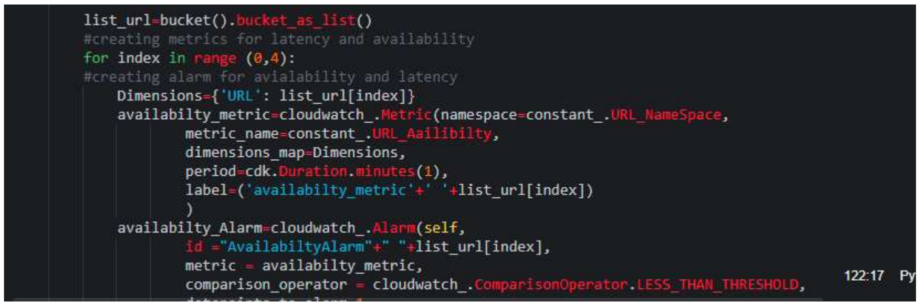


```python
import json
import boto3
#import constants as constant_

s3= boto3.client('s3')

class s3bucket_read:
    def __init__(self):
        self.Object = boto3.client('s3').get_object(Bucket="irfanskipq",Key="URLS.json")

    def bucket_as_list(self  ):
        response= s3.get_object(Bucket="irfanskipq",Key="URLS.json")
        contetnt = response['Body']
        json_oject = json.loads(contetnt.read())   #get dictionary
        list_url=[json_oject['link1'],json_oject['link2'],json_oject['link3'],json_oject['link4']]
        for url in list_url:
            print(url)
            print('--------')

        return list_url
```

Figure 16: S3 bucket reading

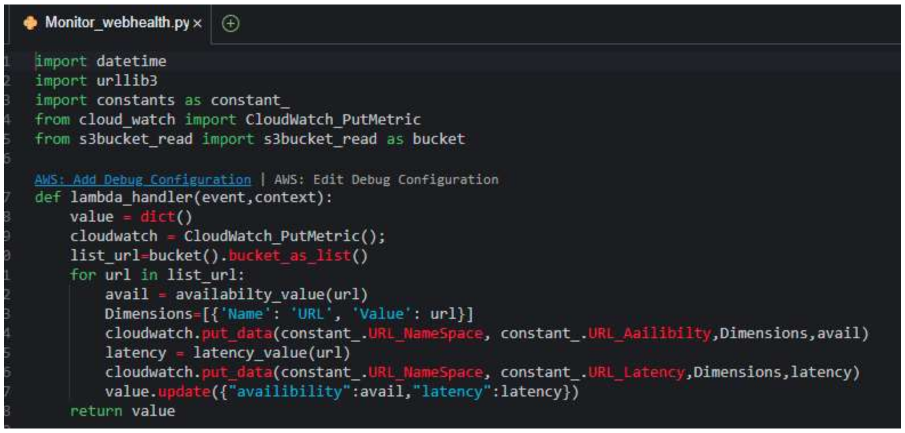
Task 05: Implement Project for customizing URLs List

Our aim for this project was to implement Health Monitoring for a list t Customize webpage JSON files in S3 bucket). We have read URLs from the S3 bucket in the previous task. Now we have to run a for loop to monitor web health owe web pages are some changes we made.



```
list_url=bucket().bucket_as_list()
#creating metrics for latency and availability
for index in range (0,4):
#creating alarm for avialability and latency
    Dimensions={'URL': list_url[index]}
    availabilty_metric=cloudwatch_.Metric(namespace=constant_.URL_NameSpace,
            metric_name=constant_.URL_Aailibilty,
            dimensions_map=Dimensions,
            period=cdk.Duration.minutes(1),
            label=('availabilty_metric'+' '+list_url[index])
            )
    availabilty_Alarm=cloudwatch_.Alarm(self,
            id ="AvailabiltyAlarm"+" "+list_url[index],
            metric = availabilty_metric,
            comparison_operator = cloudwatch_.ComparisonOperator.LESS_THAN_THRESHOLD,
```

Figure 17: Adding for loop in the Main stack

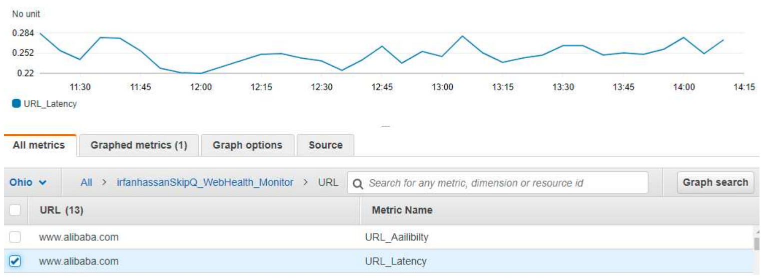


Monitor_webhealth.py ×

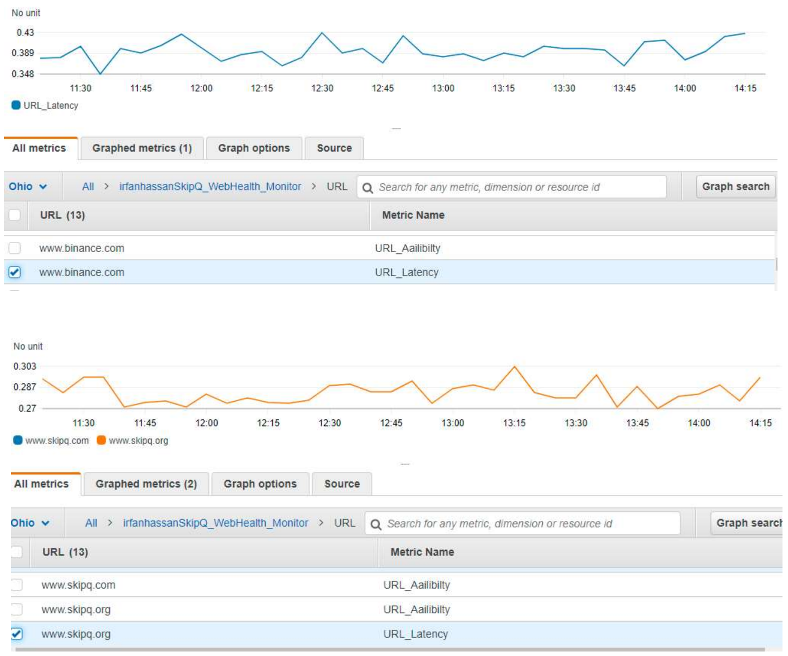
```
import datetime
import urllib3
import constants as constant_
from cloud_watch import CloudWatch_PutMetric
from s3bucket_read import s3bucket_read as bucket

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(event,context):
    value = dict()
    cloudwatch = CloudWatch_PutMetric();
    list_url=bucket().bucket_as_list()
    for url in list_url:
        avail = availabilty_value(url)
        Dimensions=[{'Name': 'URL', 'Value': url}]
        cloudwatch.put_data(constant_.URL_NameSpace, constant_.URL_Aailibilty,Dimensions,avail)
        latency = latency_value(url)
        cloudwatch.put_data(constant_.URL_NameSpace, constant_.URL_Latency,Dimensions,latency)
        value.update({"availibility":avail,"latency":latency})
    return value
```

Figure 18:Adding for loop in Web health Lambda function

Now we can see in availability and latency of each URL in Cloud Watch.



URL (13)	Metric Name
www.alibaba.com	URL_Aailibilty
✓ www.alibaba.com	URL_Latency



URL (13)	Metric Name
www.binance.com	URL_Aailibilty
✓ www.binance.com	URL_Latency

URL (13)	Metric Name
www.skipq.com	URL_Aailibilty
www.skipq.org	URL_Aailibilty
✓ www.skipq.org	URL_Latency

Figure 19: Latency of 3 URLs

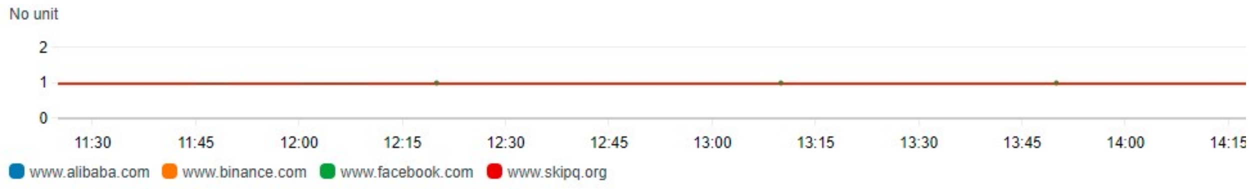


Figure 20: Latency for 4 URL

Error and solution:

Here are some common errors I faced and their solution as well.

Unknown variable

To solve this issue check spelling and if it is an issue when importing some function then install using the command “pip install –m zyz==1.135.0”

Syntax Error

Check the syntax from the API reference for the function.

Insufficient data

Check dimension parameter and duration time. Also check threshold is right or not.

References

- API Reference — AWS Cloud Development Kit 1.134.0 documentation (amazon.com)
- AWS S3 Tutorial For Beginners | AWS S3 Bucket Tutorial | AWS Training | Edureka - YouTube
- AWS DynamoDB Tutorial | AWS Services | AWS Tutorial For Beginners | AWS Training Video | Simplilearn - YouTube
- Insufficient data: CloudWatch alarm based on custom metric filter | by Marta Tatiana | Medium
- comm command in Linux with examples - GeeksforGeeks

Sprint2: CI/CD Pipeline

Project Description

Creating multi-stage pipeline having Beta/Gamma and Prod stage using CDK. the source from my GitHub repository for CI/CD Pipeline line will get the web health monitor application's source code from the GitHub repository and will automate the process of building, testing, and deploying the application. First, it installs requirements for source code then builds the code after installing all requirements. In the Beta stage, it runs the unit test and integration test. After passing through the unit test, it deploys the source code In the Production stage, it asks for manual approval then deploys the source code resource over aws.

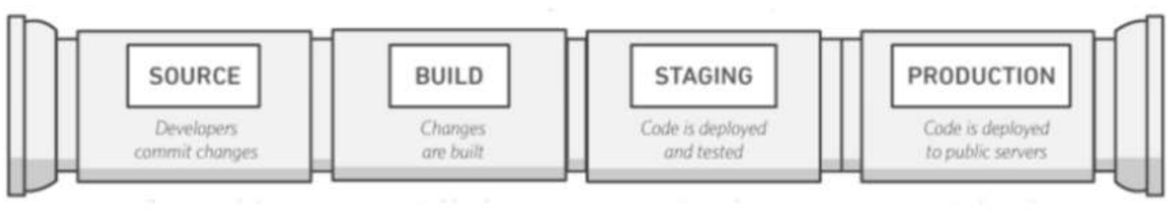

SOURCE
Developers commit changes
BUILD
Changes are built
STAGING
Code is deployed and tested
PRODUCTION
Code is deployed to public servers

Figure 21: CI/CD Pipeline

Technologies/Services

To build this application we will use the following AWS services

- Code Pipeline
- Secret Manager
- Cloud Formation

Code Pipeline

AWS Code Pipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. Code Pipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. You can easily integrate AWS Code Pipeline with third-party services such as GitHub.

Secret Manager

AWS Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources. It allows to user to save the key and then access the key when they want to use it in any application.

Cloud Formation

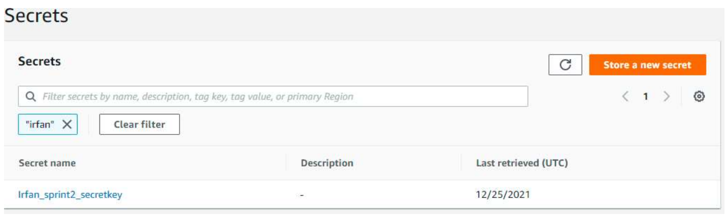
AWS Cloud Formation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and Cloud Formation takes care of provisioning and configuring those resources for you.

Setup

Before starting working on project, we have set up a few things. Follow these steps.

- Make copy of Sprint1 and remain as Sprint2.

- Create Personal Access Token from your GitHub account and store it into Secret Manager.



Secrets

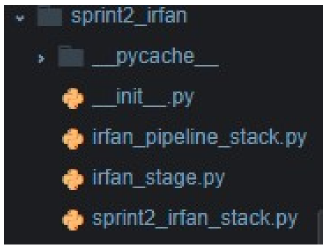
Secrets

Store a new secret
Filter secrets by name, description, tag key, tag value, or primary Region
"irfan"
Clear filter

Secret name	Description	Last retrieved (UTC)
Irfan_sprint2_secretkey	-	12/25/2021

Figure 22: Secret manager

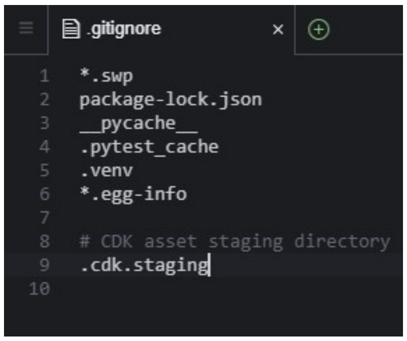
- Create pipeline stack and pipeline_satges files in the n project_stack folder.



```
sprint2_irfan
  __pycache__
  __init__.py
  irfan_pipeline_stack.py
  irfan_stage.py
  sprint2_irfan_stack.py
```

Figure 23: Files in stack folder

- Remove CDK.out from. gitignore file.

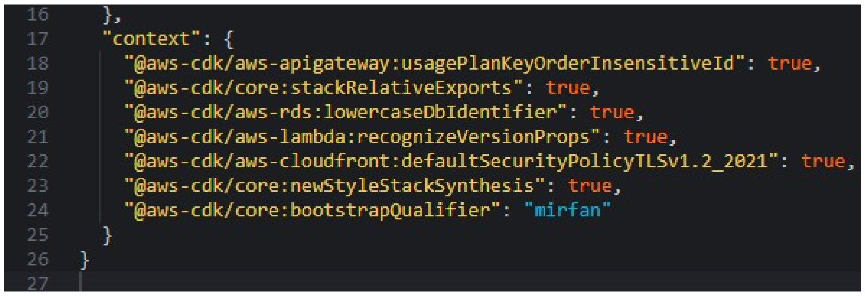


.gitignore

```
*.swp
package-lock.json
__pycache__
.pytest_cache
.venv
*.egg-info

# CDK asset staging directory
.cdk.staging
```

Figure 24: .gitignore file

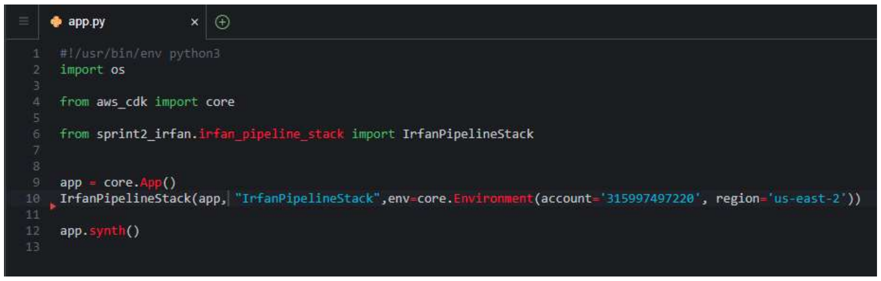
- Add "@aws-cdk/core:bootstrapQualifier": "mirfan" to cdk. json file.
-



```
    },
    "context": {
      "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": true,
      "@aws-cdk/core:stackRelativeExports": true,
      "@aws-cdk/aws-rds:lowercaseDbIdentifier": true,
      "@aws-cdk/aws-lambda:recognizeVersionProps": true,
      "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": true,
      "@aws-cdk/core:newStyleStackSynthesis": true,
      "@aws-cdk/core:bootstrapQualifier": "mirfan"
    }
  }
```

Figure 25: CDK JSON file

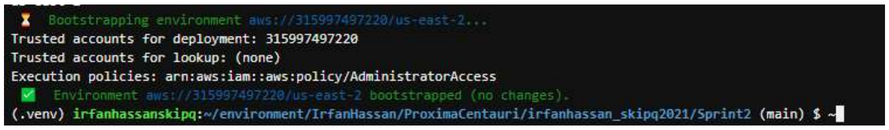
- Go into the APP.py file and change the code as shown in the figure.



```python
#!/usr/bin/env python3
import os

from aws_cdk import core

from sprint2_irfan.irfan_pipeline_stack import IrfanPipelineStack


app = core.App()
IrfanPipelineStack(app, "IrfanPipelineStack",env=core.Environment(account='315997497220', region='us-east-2'))

app.synth()
```

Figure 26: APP.py

- Bootstrap the environment using this command.
- `cdk bootstrap aws://<Acount ID>/<Region> --qualifier <name> --toolkit-stack-name <name>`



```
 ⏳  Bootstrapping environment aws://315997497220/us-east-2...
Trusted accounts for deployment: 315997497220
Trusted accounts for lookup: (none)
Execution policies: arn:aws:iam::aws:policy/AdministratorAccess
 ✅  Environment aws://315997497220/us-east-2 bootstrapped (no changes).
(.venv) irfanhassanskipq:~/environment/IrfanHassan/ProximaCentauri/irfanhassan_skipq2021/Sprint2 (main) $ ~
```

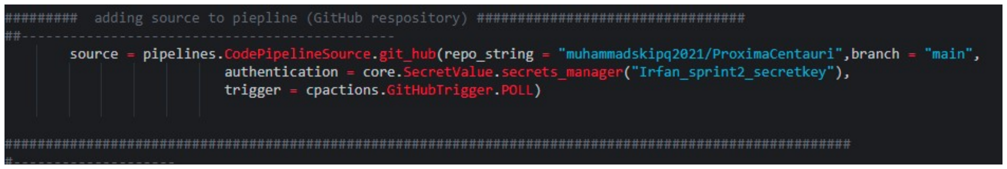
Figure 27: Bootstrap don

Project milestones

This project is divided into subtasks. Subtasks are discussed below.

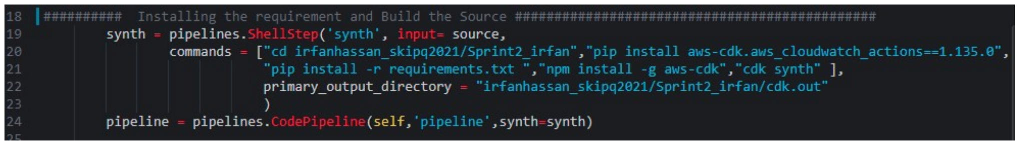
Task 01: GitHub repo as Source in Pipeline and Build the source

First, we have to create source for the pipeline. GitHub will be 3rd party we will be integrating to our pipeline as the source. After adding source build the source. Code for Source is shown below. Add repository in repo string, set branch and then add the filename of secret manager where you have stored personal access token. With GitHub triggered "POLL", Code Pipeline periodically checks the source for changes.



```
#########  adding source to piepline (GitHub respository) ##################################
##-----------------------------------------
        source = pipelines.CodePipelineSource.git_hub(repo_string = "muhammadskipq2021/ProximaCentauri",branch = "main",
                                                      authentication = core.SecretValue.secrets_manager("Irfan_sprint2_secretkey"),
                                                      trigger = cpactions.GitHubTrigger.POLL)

##############################################################################################################
#-------------------
```

Figure 28: Adding GitHub Repo as Source

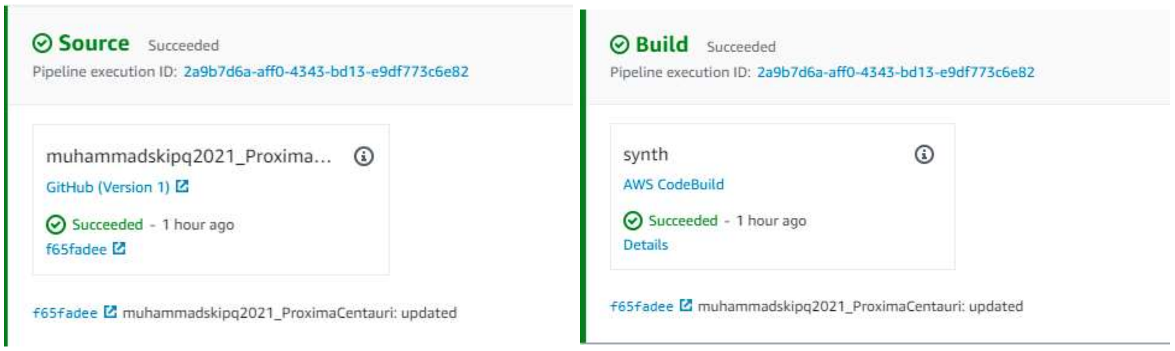


```
##########  Installing the requirement and Build the Source ##############################################
            synth = pipelines.ShellStep('synth', input= source,
                    commands = ["cd irfanhassan_skipq2021/Sprint2_irfan","pip install aws-cdk.aws_cloudwatch_actions==1.135.0",
                                "pip install -r requirements.txt ","npm install -g aws-cdk","cdk synth" ],
                                primary_output_directory = "irfanhassan_skipq2021/Sprint2_irfan/cdk.out"
                                )
            pipeline = pipelines.CodePipeline(self,'pipeline',synth=synth)
```

Figure 29: Building the Source

Commit code and push code to GitHub. Now run command "cdk deploy pipeline stack"

If it run successfully then check Code Pipeline. You will get these results if the pipeline is run successfully.



Source Succeeded
Pipeline execution ID: 2a9b7d6a-aff0-4343-bd13-e9df773c6e82

muhammadskipq2021_Proxima...
GitHub (Version 1)
Succeeded - 1 hour ago
f65fadee

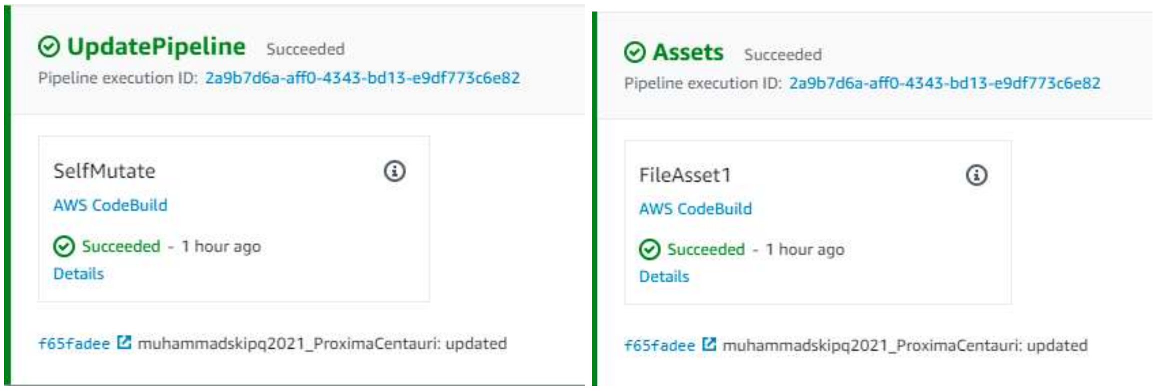
f65fadee muhammadskipq2021_ProximaCentauri: updated

Build Succeeded
Pipeline execution ID: 2a9b7d6a-aff0-4343-bd13-e9df773c6e82

synth
AWS CodeBuild
Succeeded - 1 hour ago
Details

f65fadee muhammadskipq2021_ProximaCentauri: updated

Figure 30: Output (Source and Build)



UpdatePipeline Succeeded
Pipeline execution ID: 2a9b7d6a-aff0-4343-bd13-e9df773c6e82

SelfMutate
AWS CodeBuild
Succeeded - 1 hour ago
Details

f65fadee muhammadskipq2021_ProximaCentauri: updated

Assets Succeeded
Pipeline execution ID: 2a9b7d6a-aff0-4343-bd13-e9df773c6e82

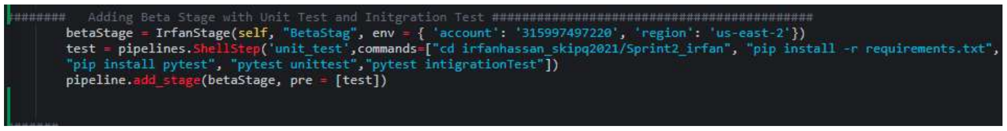
FileAsset1
AWS CodeBuild
Succeeded - 1 hour ago
Details

f65fadee muhammadskipq2021_ProximaCentauri: updated

Figure 31: Pipeline updated and Assert are successful

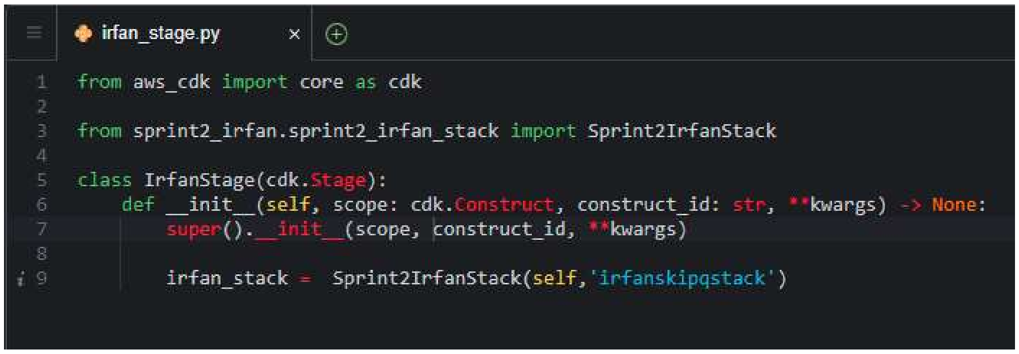
Task 02: Adding Beta stage with unit test in Pipeline

Now we have to create Beta Stage. Unit test is also created. While adding Beta stage, test is set as pre. It means if the it passes test then it moves toward deploying the code.



```
#######   Adding Beta Stage with Unit Test and Initgration Test ##########################################
        betaStage = IrfanStage(self, "BetaStag", env = { 'account': '315997497220', 'region': 'us-east-2'})
        test = pipelines.ShellStep('unit_test',commands=["cd irfanhassan_skipq2021/Sprint2_irfan", "pip install -r requirements.txt",
        "pip install pytest", "pytest unittest","pytest intigrationTest"])
        pipeline.add_stage(betaStage, pre = [test])
```

Figure 32: Creating Beta st,age, unit test and athe dding to pipeline

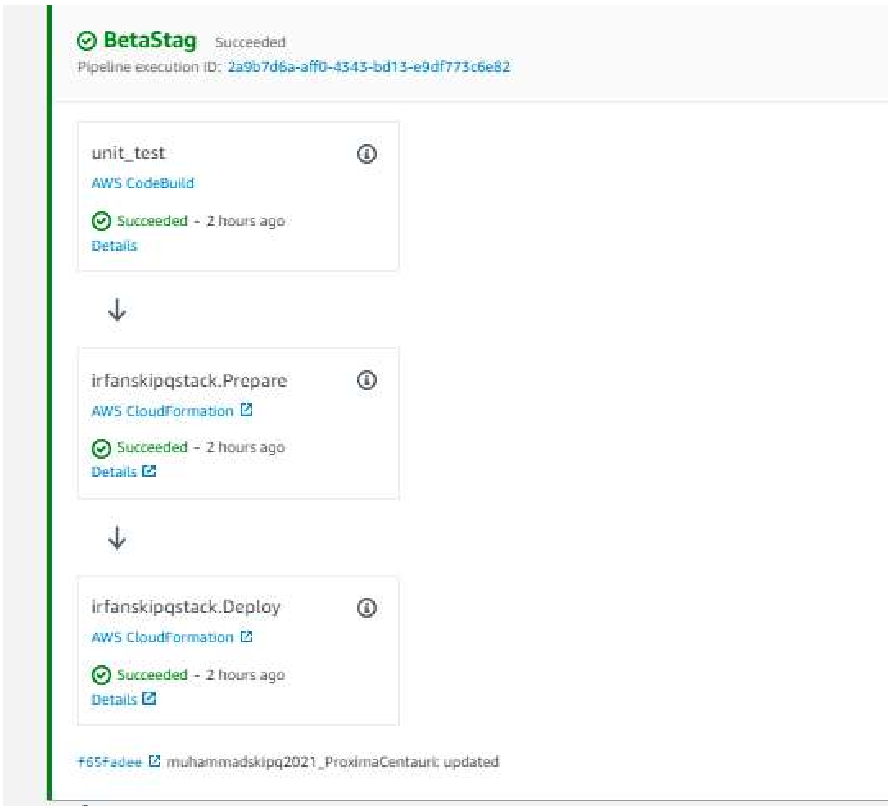


irfan_stage.py

```
from aws_cdk import core as cdk

from sprint2_irfan.sprint2_irfan_stack import Sprint2IrfanStack

class IrfanStage(cdk.Stage):
    def __init__(self, scope: cdk.Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        irfan_stack =  Sprint2IrfanStack(self,'irfanskipqstack')
```

Figure 33: Pipeline Stage class

Now again commit the code and push to GitHub.



BetaStag Succeeded
Pipeline execution ID: 2a9b7d6a-aff0-4343-bd13-e9df773c6e82

unit_test
AWS CodeBuild
Succeeded - 2 hours ago
Details

irfanskipqstack.Prepare
AWS CloudFormation
Succeeded - 2 hours ago
Details

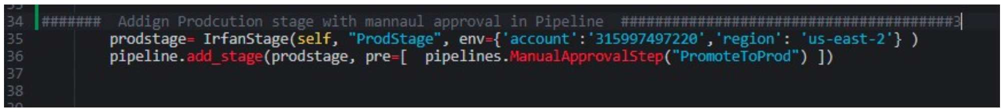
irfanskipqstack.Deploy
AWS CloudFormation
Succeeded - 2 hours ago
Details

f65fadee muhammadskipq2021_ProximaCentauri: updated

Figure 34: Beta stage is added successfully

Task 03: Adding Production stage with Manual Approval

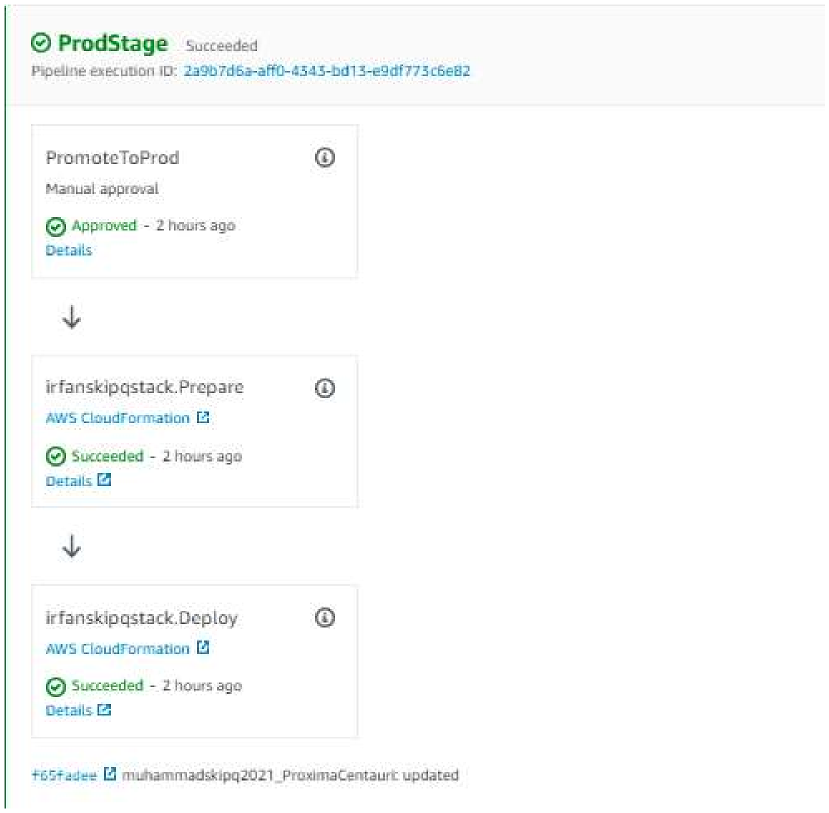
Adding Production is last part of Pipeline. We created production stage and then add it to pipeline with manual permission as pre. It means it will ask user to approve then it will deploy the code to server. Here is code for adding production stage.



```
#######  Addign Prodcution stage with mannaul approval in Pipeline  ########################################3
        prodstage= IrfanStage(self, "ProdStage", env={'account':'315997497220','region': 'us-east-2'} )
        pipeline.add_stage(prodstage, pre=[  pipelines.ManualApprovalStep("PromoteToProd") ])
```

Figure 35: Adding Production stage to pipeline

Again commit and push code to GitHub repo. Check Code Pipeline you will get these results.


ProdStage Succeeded
Pipeline execution ID: 2a9b7d6a-aff0-4343-bd13-e9df773c6e82

PromoteToProd
Manual approval
Approved - 2 hours ago
Details

irfanskipqstack.Prepare
AWS CloudFormation
Succeeded - 2 hours ago
Details

irfanskipqstack.Deploy
AWS CloudFormation
Succeeded - 2 hours ago
Details

f65fadee muhammadskipq2021_ProximaCentauri: updated

Figure 36: Adding Production stage

Task 04: Add Auto Rollback automation

In last step we have added an auto roll back to automate the process when there is mistake in current version of code. Store the current version in alias. We use AWS duration metric and will measure the time duration for our web health lambda function. Then put alarm when it will cross threshold (we assume 5sec). when alarm generated it will redeploy the save version in alias. Here is code screenshot.

Results and Discussion

As we have deployed the code in Beta stage and then production stage. We get Availability and Latency Metrics for both Beta and production stage. For one URLs availability graph for Beta and Prod are shown below.

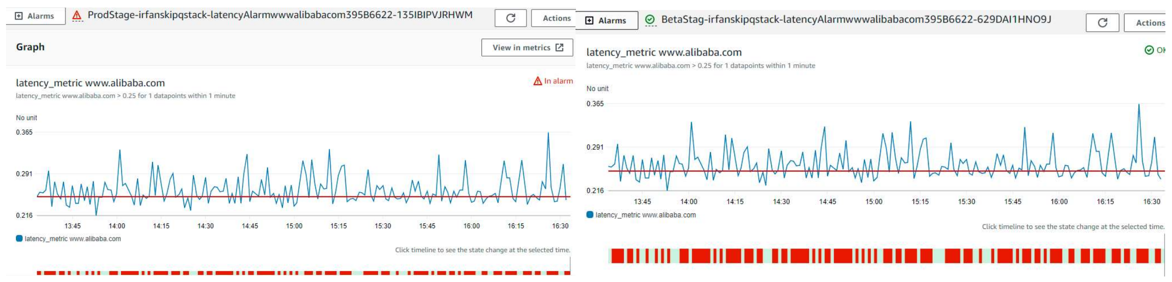
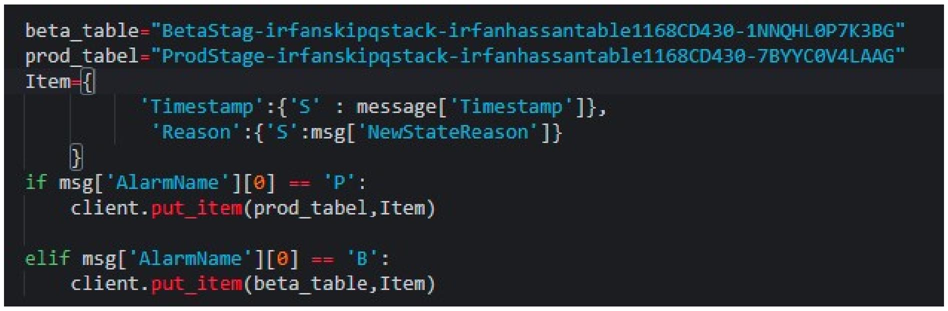


Figure 37: Plot of Latency for alibab.com for Beta and Prod stages

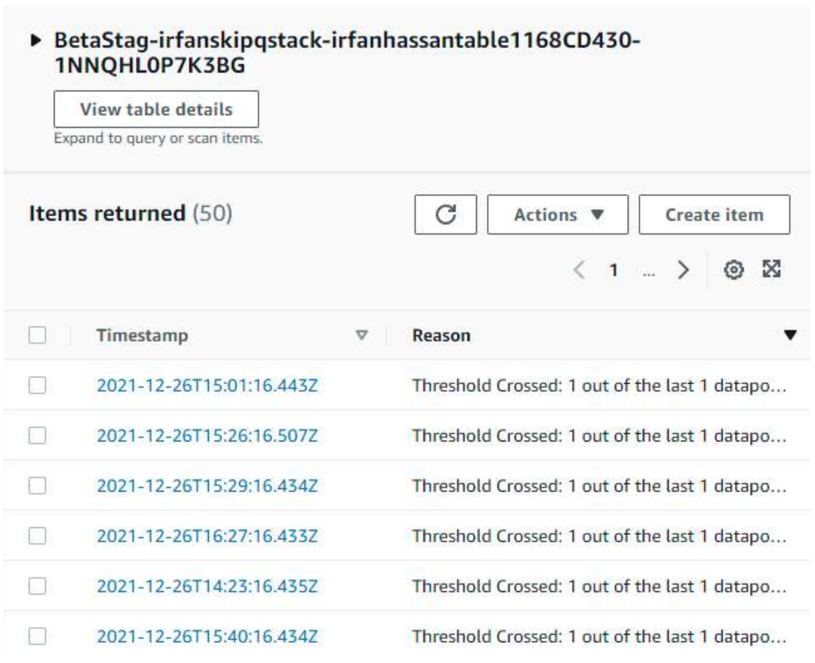
Same as metrics and alarm are created for both Beta and Prod stages. Dynamo DB table are also created separately. So we change the code in dynamo Lambda accordingly. According to Alarm generate on Pro or Beta stage. It writes alarm details in corresponding table. Here is code.



```
beta_table="BetaStag-irfanskipqstack-irfanhassantable1168CD430-1NNQHL0P7K3BG"
prod_tabel="ProdStage-irfanskipqstack-irfanhassantable1168CD430-7BYYC0V4LAAG"
Item={
          'Timestamp':{'S' : message['Timestamp']},
           'Reason':{'S':msg['NewStateReason']}
    }
if msg['AlarmName'][0] == 'P':
    client.put_item(prod_tabel,Item)

elif msg['AlarmName'][0] == 'B':
    client.put_item(beta_table,Item)
```

Figure 38: dynamo DB lambda

Here is screenshot for Table.



▸ BetaStag-irfanskipqstack-irfanhassantable1168CD430-1NNQHL0P7K3BG

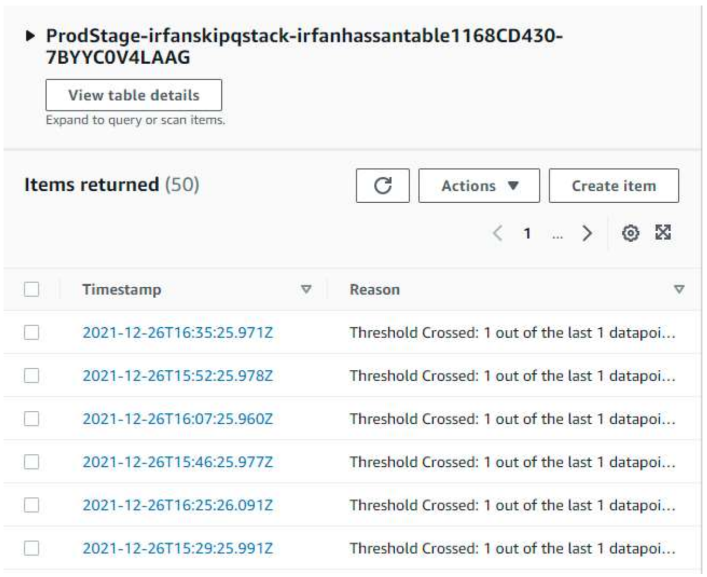
View table details

Expand to query or scan items.

Items returned (50)

	Timestamp	Reason
☐	2021-12-26T15:01:16.443Z	Threshold Crossed: 1 out of the last 1 datapo...
☐	2021-12-26T15:26:16.507Z	Threshold Crossed: 1 out of the last 1 datapo...
☐	2021-12-26T15:29:16.434Z	Threshold Crossed: 1 out of the last 1 datapo...
☐	2021-12-26T16:27:16.433Z	Threshold Crossed: 1 out of the last 1 datapo...
☐	2021-12-26T14:23:16.435Z	Threshold Crossed: 1 out of the last 1 datapo...
☐	2021-12-26T15:40:16.434Z	Threshold Crossed: 1 out of the last 1 datapo...

Figure 39: Beta stage Table

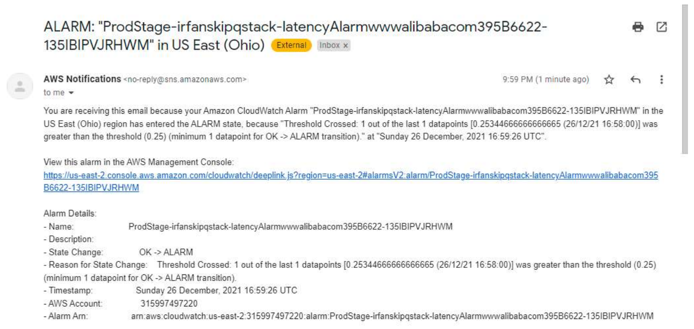


▸ ProdStage-irfanskipqstack-irfanhassantable1168CD430-7BYYC0V4LAAG

Timestamp	Reason
2021-12-26T16:35:25.971Z	Threshold Crossed: 1 out of the last 1 datapoi…
2021-12-26T15:52:25.978Z	Threshold Crossed: 1 out of the last 1 datapoi…
2021-12-26T16:07:25.960Z	Threshold Crossed: 1 out of the last 1 datapoi…
2021-12-26T15:46:25.977Z	Threshold Crossed: 1 out of the last 1 datapoi…
2021-12-26T16:25:26.091Z	Threshold Crossed: 1 out of the last 1 datapoi…
2021-12-26T15:29:25.991Z	Threshold Crossed: 1 out of the last 1 datapoi…

Figure 40: Prod Stage Table

Email Notification are also received when alarm trigger.



ALARM: "ProdStage-irfanskipqstack-latencyAlarmwwwalibabacom395B6622-135IBIPVJRHWM" in US East (Ohio)

AWS Notifications <no-reply@sns.amazonaws.com> 9:59 PM (1 minute ago)
to me

You are receiving this email because your Amazon CloudWatch Alarm "ProdStage-irfanskipqstack-latencyAlarmwwwalibabacom395B6622-135IBIPVJRHWM" in the US East (Ohio) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [0.25344666666666665 (26/12/21 16:58:00)] was greater than the threshold (0.25) (minimum 1 datapoint for OK -> ALARM transition)." at "Sunday 26 December, 2021 16:59:26 UTC".

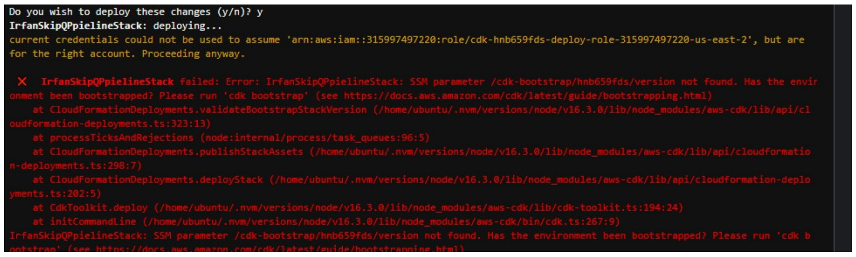
View this alarm in the AWS Management Console:
https://us-east-2.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-2#alarmsV2:alarm/ProdStage-irfanskipqstack-latencyAlarmwwwalibabacom395B6622-135IBIPVJRHWM

Alarm Details:
- Name: ProdStage-irfanskipqstack-latencyAlarmwwwalibabacom395B6622-135IBIPVJRHWM
- Description:
- State Change: OK -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [0.25344666666666665 (26/12/21 16:58:00)] was greater than the threshold (0.25) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Sunday 26 December, 2021 16:59:26 UTC
- AWS Account: 315997497220
- Alarm Arn: arn:aws:cloudwatch:us-east-2:315997497220:alarm:ProdStage-irfanskipqstack-latencyAlarmwwwalibabacom395B6622-135IBIPVJRHWM

Figure 41: Alarm Notification at Email

Errors and Solution

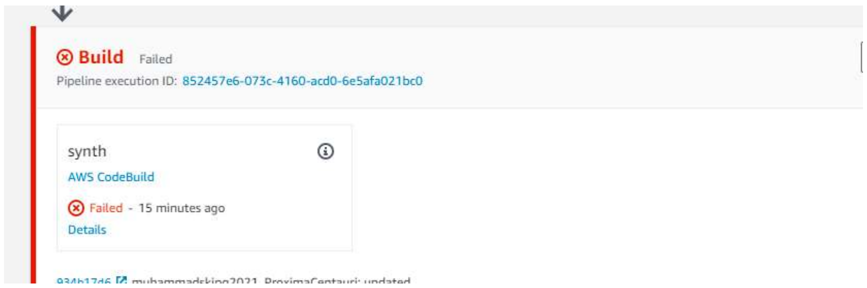
- While deploying the pipeline facing this error. To resolve this issue remove the "@aws-cdk/core:newStyleStackSynthesis": true" from cdk.json file.



```
Do you wish to deploy these changes (y/n)? y
IrfanSkipQPpielineStack: deploying...
current credentials could not be used to assume 'arn:aws:iam::315997497220:role/cdk-hnb659fds-deploy-role-315997497220-us-east-2', but are
for the right account. Proceeding anyway.

 ✗  IrfanSkipQPpielineStack failed: Error: IrfanSkipQPpielineStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the envir
onment been bootstrapped? Please run 'cdk bootstrap' (see https://docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html)
    at CloudFormationDeployments.validateBootstrapStackVersion (/home/ubuntu/.nvm/versions/node/v16.3.0/lib/node_modules/aws-cdk/lib/api/cl
oudformation-deployments.ts:323:13)
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
    at CloudFormationDeployments.publishStackAssets (/home/ubuntu/.nvm/versions/node/v16.3.0/lib/node_modules/aws-cdk/lib/api/cloudformatio
n-deployments.ts:298:7)
    at CloudFormationDeployments.deployStack (/home/ubuntu/.nvm/versions/node/v16.3.0/lib/node_modules/aws-cdk/lib/api/cloudformation-deplo
yments.ts:202:5)
    at CdkToolkit.deploy (/home/ubuntu/.nvm/versions/node/v16.3.0/lib/node_modules/aws-cdk/lib/cdk-toolkit.ts:194:24)
    at initCommandLine (/home/ubuntu/.nvm/versions/node/v16.3.0/lib/node_modules/aws-cdk/bin/cdk.ts:267:9)
IrfanSkipQPpielineStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment been bootstrapped? Please run 'cdk b
ootstrap' (see https://docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html)
```

Figure 42: while deploying: Error

- Build Failed: Go into details and check logs. Resolve issue accordingly. Also check on GitHub you have push cdk.out folder. If it is not solved, then check. ignore and remove cdk.out.



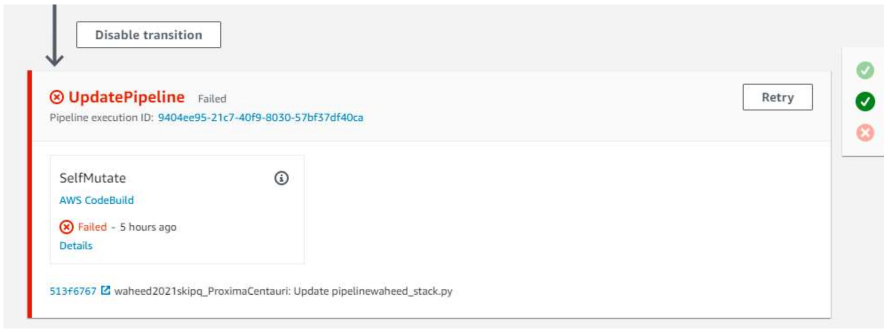
Build Failed
Pipeline execution ID: 852457e6-073c-4160-acd0-6e5afa021bc0

synth
AWS CodeBuild
Failed - 15 minutes ago
Details

934b17d6 muhammadskipq2021_ProximaCentauri: updated

Figure 43: Build failed: Error

- Updated Failed: Add policy to stack resources. Go to stack and open resources for stacks. Find resource having IAM Role. Click on resource and add policy.



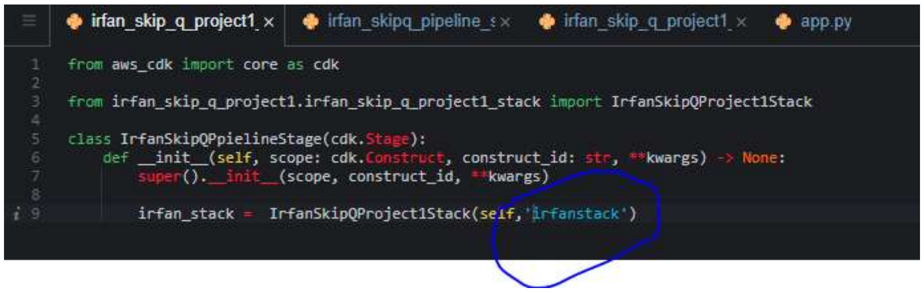
UpdatePipeline Failed
Pipeline execution ID: 9404ee95-21c7-40f9-8030-57bf37df40ca

SelfMutate
AWS CodeBuild
Failed - 5 hours ago
Details

513f6767 waheed2021skipq_ProximaCentauri: Update pipelinewaheed_stack.py

Figure 44:Update Failed: Error

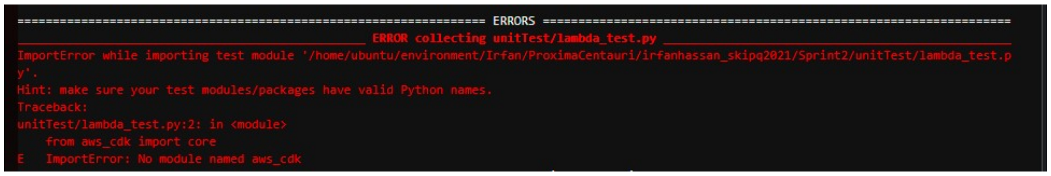
- While Adding stage getting error. The reason is that you can use alphabets or number only for creating stack in stage file.



```python
from aws_cdk import core as cdk

from irfan_skip_q_project1.irfan_skip_q_project1_stack import IrfanSkipQProject1Stack

class IrfanSkipQPpielineStage(cdk.Stage):
    def __init__(self, scope: cdk.Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        irfan_stack =  IrfanSkipQProject1Stack(self,'irfanstack')
```

Figure 45: Adding stage: Error

- While running pytest unit test not able to import aws_core. install requirements using "npm install -g aws-cdk " and "pip install -r requiremets.txt ". second thing file name for unit test should be like "irfan_test". and class in file should be like "test_irfan". then run pytest.



```
================================================================ ERRORS ================================================================
______________________________________________ ERROR collecting unitTest/lambda_test.py ______________________________________________
ImportError while importing test module '/home/ubuntu/environment/Irfan/ProximaCentauri/irfanhassan_skipq2021/Sprint2/unitTest/lambda_test.p
y'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
unitTest/lambda_test.py:2: in <module>
    from aws_cdk import core
E   ImportError: No module named aws_cdk
```

Figure 46: Pytest Unit test : Error

- Table already exist: Remove table name and it will generate table by itself. Then add table name into dynamo DB Lambda function.

References

- pytest: ModuleNotFoundError: No module named ‘requests’ | by Dirk Avery | Medium
- https://www.youtube.com/watch?v=sTTvZ5ltZG0
- Deploying Infrastructure on AWS with Terraform and AWS CodePipeline (#CloudGuruChallenge Series) (Part 1/3) - DEV Community
- https://www.bing.com/search?q=can+not+import+aws_cdk&cvid=37fbce8a809d4e45a575ba04e1ce7264&aqs=edge..69i57.16974j0j4&FORM=ANAB01&DAF0=1&PC=U531

<u>Sprint3: API Gateway endpoint for web crawler</u>

Project Description

The aim of this project is to create a public CRUD API Gateway endpoint for web crawler to create/read/update/delete the target list containing the list of website/webpages to crawl. A JSON file will be uploaded to bucket and lambda function will trigger and will store URL from JSON to dynamo DB table. Then user can update the URL in table using API Gateway by using PUT/DELETE/UPDATE/GET method.

Technologies/Services

- API Gateway

- Dynamo DB
- S3
- Lambda

API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to publish, maintain, monitor, secure, and operate APIs at any scale. It's a pay-as-you-go service that takes care of all of the undifferentiated heavy lifting involved in securely and reliably running APIs at scale.

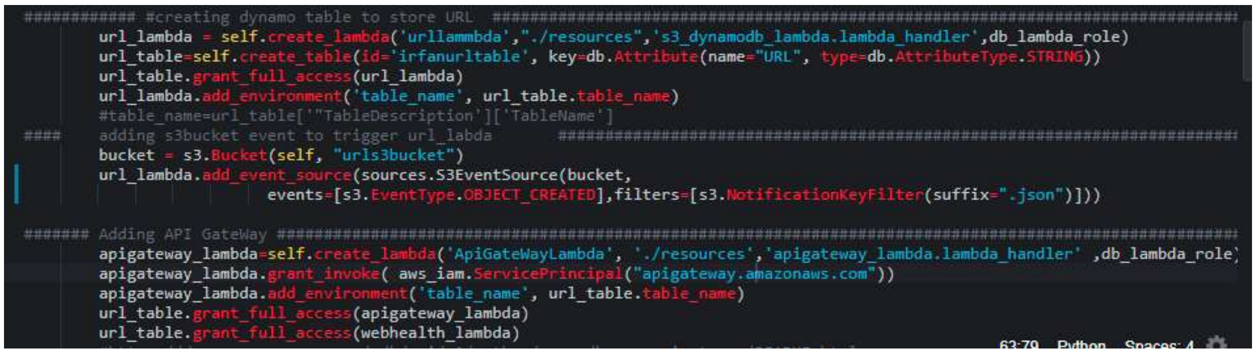
Project milestones

Here are subtasks for this project.

- Create Lambda functions and Dynamo DB table.
- Upload JSON file from S3 bucket to Dynamo DB Table
- Create API Gateway and add PUT/DELETE/UPDATE/GET method.

Task 01: Create Lambda Function and Dynamo DB table

The first step was to create a dynamo DB table to store the URL of web pages. We created a dynamo DB table. Then created two lambda functions. Lambda function will be triggered when JSON file will be uploaded to the S3 bucket. We give all permission to this lambda function for the dynamo DB table. The second lambda function will be triggered when the user uses any method in API Gateway. Here is code for creating lambda functions and dynamo DB table.



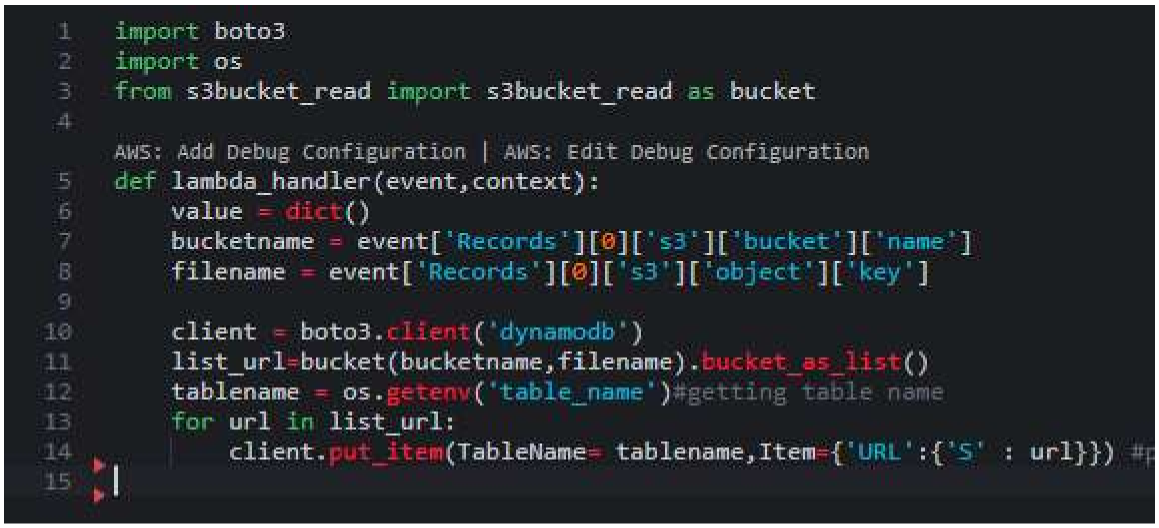
```
############ #creating dynamo table to store URL  ##################################################################
        url_lambda = self.create_lambda('urllammbda',"./resources",'s3_dynamodb_lambda.lambda_handler',db_lambda_role)
        url_table=self.create_table(id='irfanurltable', key=db.Attribute(name="URL", type=db.AttributeType.STRING))
        url_table.grant_full_access(url_lambda)
        url_lambda.add_environment('table_name', url_table.table_name)
        #table_name=url_table['"TableDescription']['TableName']
####    adding s3bucket event to trigger url_labda      ##############################################################
        bucket = s3.Bucket(self, "urls3bucket")
        url_lambda.add_event_source(sources.S3EventSource(bucket,
                             events=[s3.EventType.OBJECT_CREATED],filters=[s3.NotificationKeyFilter(suffix=".json")]))

####### Adding API GateWay ##########################################################################################
        apigateway_lambda=self.create_lambda('ApiGateWayLambda', './resources','apigateway_lambda.lambda_handler' ,db_lambda_role)
        apigateway_lambda.grant_invoke( aws_iam.ServicePrincipal("apigateway.amazonaws.com"))
        apigateway_lambda.add_environment('table_name', url_table.table_name)
        url_table.grant_full_access(apigateway_lambda)
        url_table.grant_full_access(webhealth_lambda)
```

Figure 47: Code for Lambda Function and Dynamo DB Table

Task 02: Upload JSON file from S3 bucket to Dynamo DB Table

In Lambda Function for S3to dynamo DB, we get bucket name and key. Then read content from JSON file and store the content into dynamo DB table.

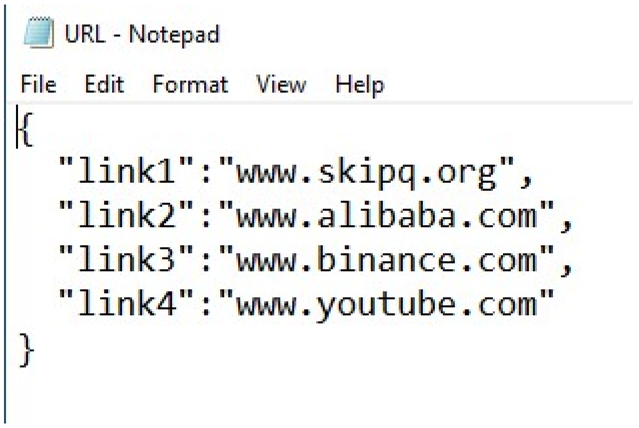
Here is code.



```
import boto3
import os
from s3bucket_read import s3bucket_read as bucket

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(event,context):
    value = dict()
    bucketname = event['Records'][0]['s3']['bucket']['name']
    filename = event['Records'][0]['s3']['object']['key']

    client = boto3.client('dynamodb')
    list_url=bucket(bucketname,filename).bucket_as_list()
    tablename = os.getenv('table_name')#getting table name
    for url in list_url:
        client.put_item(TableName= tablename,Item={'URL':{'S' : url}}) #p
```

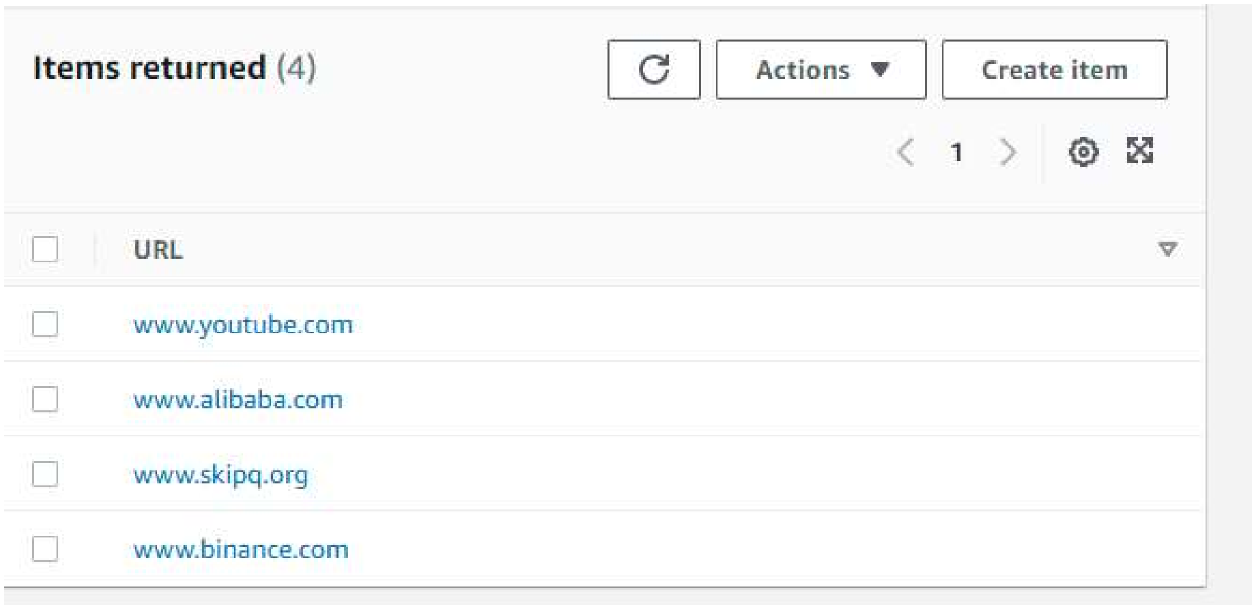
Figure 48: Load JSON to Dynamo DB table

Here is output when JSON file is uploaded.



```
{
  "link1":"www.skipq.org",
  "link2":"www.alibaba.com",
  "link3":"www.binance.com",
  "link4":"www.youtube.com"
}
```

Figure 49: input JSON file uploaded to S3 bucket

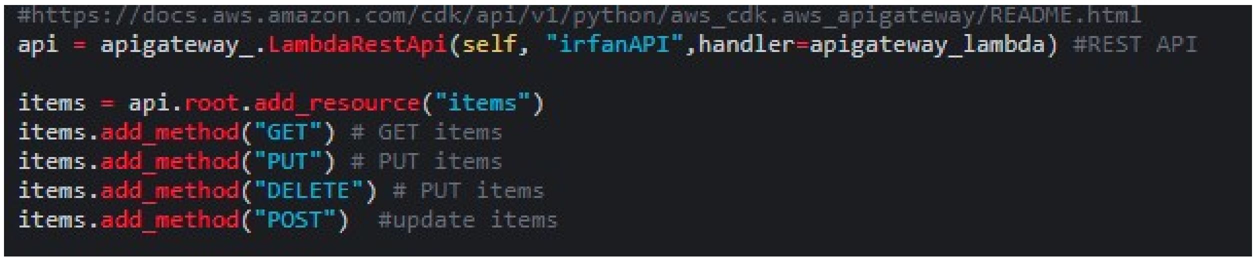


URL
www.youtube.com
www.alibaba.com
www.skipq.org
www.binance.com

Figure 50: Dynamo DB table after JSON file is uploaded in an S3 bucket

Task 03: Create API Gateway and add PUT/DELETE/UPDATE/GET method

In lambda function for API gateway, we read the method name from the event and according to possibilities of the method we ad conditions. In each condition, we write code to perform that method.

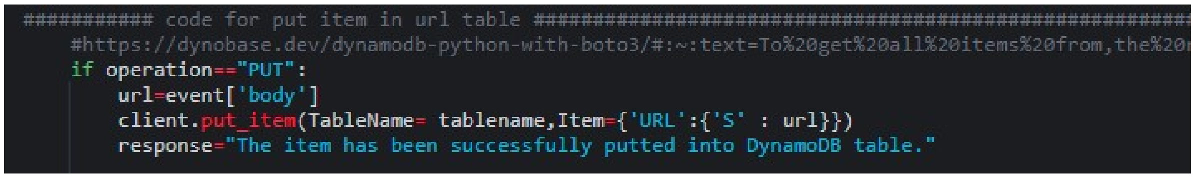


```
#https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_apigateway/README.html
api = apigateway_.LambdaRestApi(self, "irfanAPI",handler=apigateway_lambda) #REST API

items = api.root.add_resource("items")
items.add_method("GET") # GET items
items.add_method("PUT") # PUT items
items.add_method("DELETE") # PUT items
items.add_method("POST")  #update items
```

Figure 51: Created REST API gateway

Put Item

In put Item method, user enter single URL to delete from the table. So, we write that URL of webpage into Dynamo DB table and add message to display back to user in return for using PUT item method.

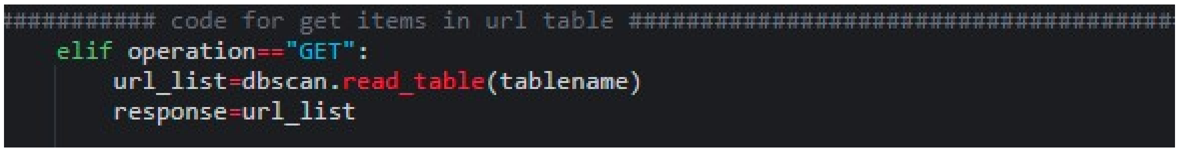


```
########### code for put item in url table ##########################################################
    #https://dynobase.dev/dynamodb-python-with-boto3/#:~:text=To%20get%20all%20items%20from,the%20r
    if operation=="PUT":
        url=event['body']
        client.put_item(TableName= tablename,Item={'URL':{'S' : url}})
        response="The item has been successfully putted into DynamoDB table."
```

Figure 52: Code for PUT item into dynamo DB table

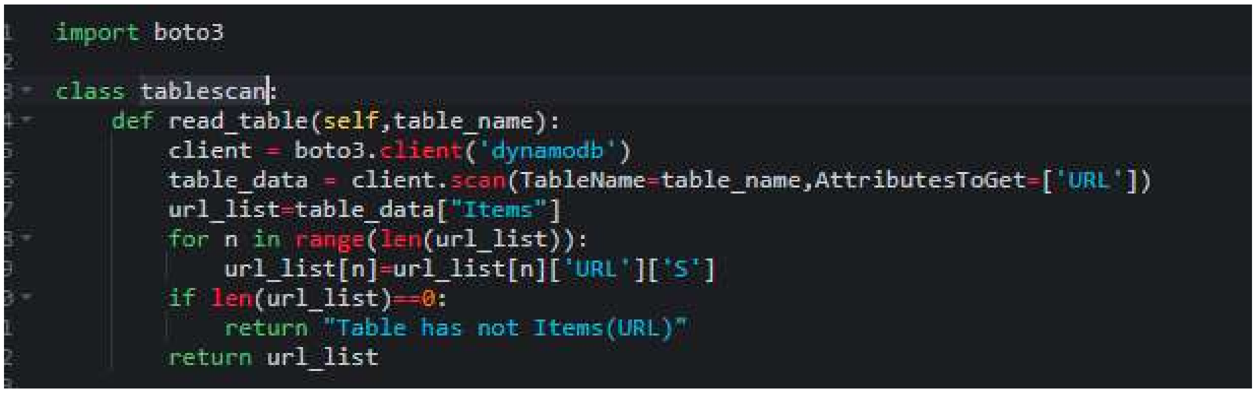
Get Item

In get item method, we scan table and get list of URL in table and the return list of URL of table is not empty and send message if table is empty.



```
########### code for get items in url table ########################################
    elif operation=="GET":
        url_list=dbscan.read_table(tablename)
        response=url_list
```

Figure 53:Code for GET item into dynamo DB table

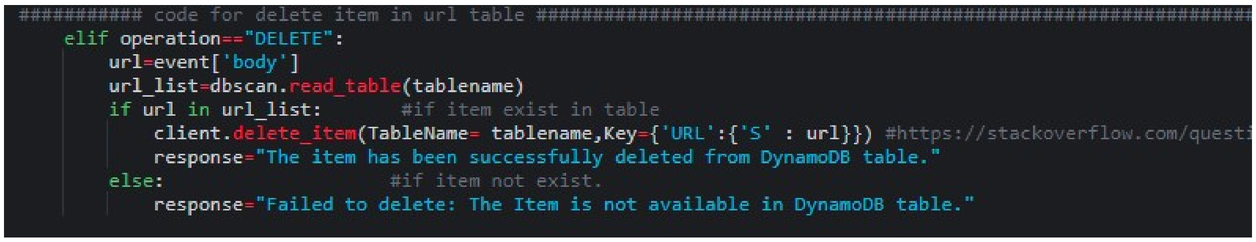


```
import boto3

class tablescan:
    def read_table(self,table_name):
        client = boto3.client('dynamodb')
        table_data = client.scan(TableName=table_name,AttributesToGet=['URL'])
        url_list=table_data["Items"]
        for n in range(len(url_list)):
            url_list[n]=url_list[n]['URL']['S']
        if len(url_list)==0:
            return "Table has not Items(URL)"
        return url_list
```

Figure 54: Scan table function

Delete Item

In delete item method, user enter single URL to delete from table. So, we get all data from dynamo dB table and check if given URL is available in table then delete it and if not available then send notification to user accordingly.

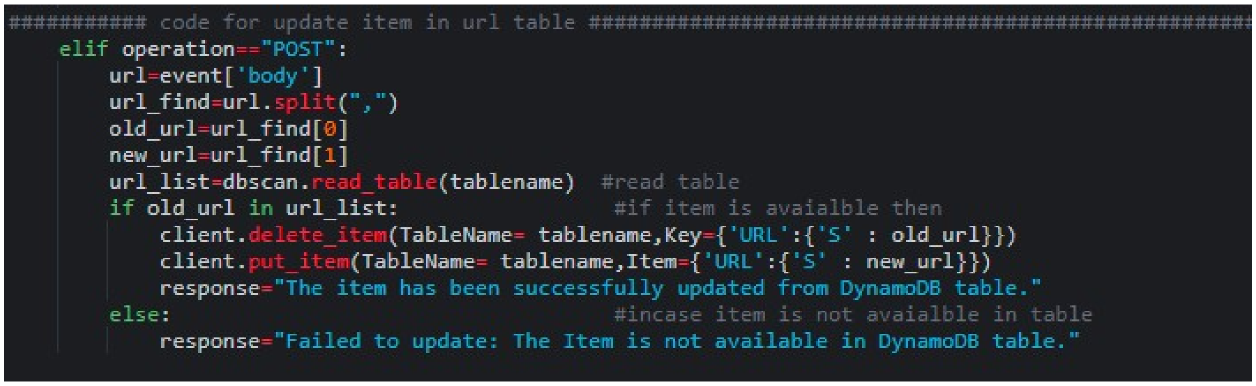


```
########### code for delete item in url table ##############################################################
    elif operation=="DELETE":
        url=event['body']
        url_list=dbscan.read_table(tablename)
        if url in url_list:       #if item exist in table
            client.delete_item(TableName= tablename,Key={'URL':{'S' : url}}) #https://stackoverflow.com/questi
            response="The item has been successfully deleted from DynamoDB table."
        else:                    #if item not exist.
            response="Failed to delete: The Item is not available in DynamoDB table."
```

Figure 55:Code for DELETE item into dynamo DB table

Update Item

In update Item method, we ask user to enter two URL separated by comma and in lambda function, first we separate the URL using split function and check if URL user want to update is available in table or not. If it is available, then update the URL with new URL and if URL is not available then send notification message to user.

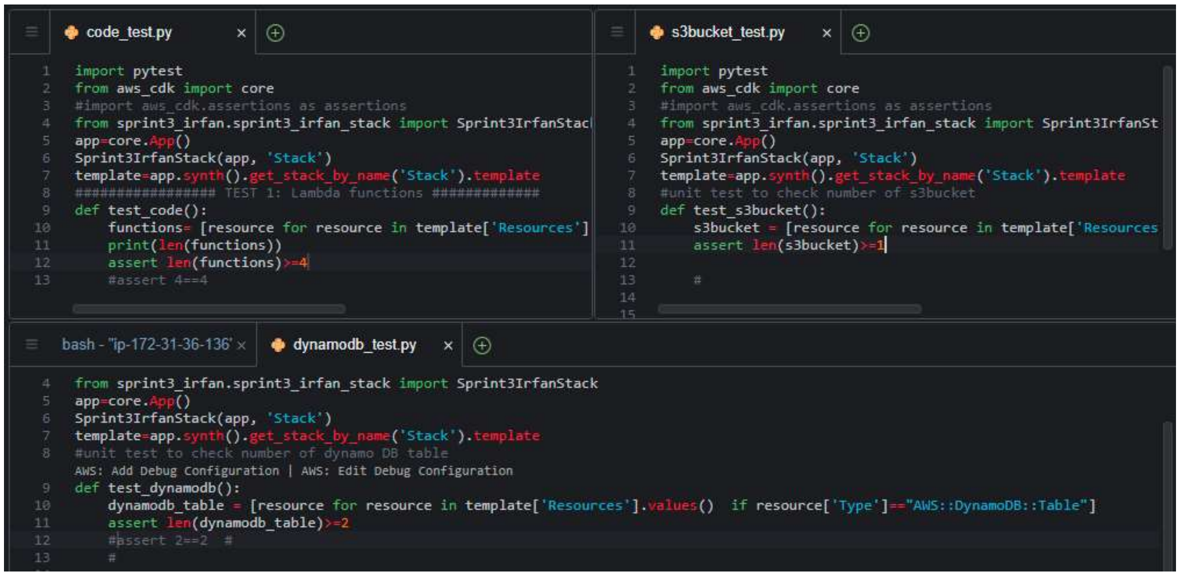


```
########### code for update item in url table ############################################################
    elif operation=="POST":
        url=event['body']
        url_find=url.split(",")
        old_url=url_find[0]
        new_url=url_find[1]
        url_list=dbscan.read_table(tablename)  #read table
        if old_url in url_list:                 #if item is avaialble then
            client.delete_item(TableName= tablename,Key={'URL':{'S' : old_url}})
            client.put_item(TableName= tablename,Item={'URL':{'S' : new_url}})
            response="The item has been successfully updated from DynamoDB table."
        else:                                   #incase item is not avaialble in table
            response="Failed to update: The Item is not available in DynamoDB table."
```

Figure 56:Code for UPDATE item into dynamo DB table

Task 04: Unit test and integration test

I have added 3-unit tests. Unit test are checking the s3 bucket, dynamo DB table and lambda function for API Gateway are created successfully.



code_test.py

```
import pytest
from aws_cdk import core
#import aws_cdk.assertions as assertions
from sprint3_irfan.sprint3_irfan_stack import Sprint3IrfanStac
app=core.App()
Sprint3IrfanStack(app, 'Stack')
template=app.synth().get_stack_by_name('Stack').template
################# TEST 1: Lambda functions #############
def test_code():
    functions= [resource for resource in template['Resources']]
    print(len(functions))
    assert len(functions)>=4
    #assert 4==4
```

s3bucket_test.py

```
import pytest
from aws_cdk import core
#import aws_cdk.assertions as assertions
from sprint3_irfan.sprint3_irfan_stack import Sprint3IrfanSt
app=core.App()
Sprint3IrfanStack(app, 'Stack')
template=app.synth().get_stack_by_name('Stack').template
#unit test to check number of s3bucket
def test_s3bucket():
    s3bucket = [resource for resource in template['Resources
    assert len(s3bucket)>=1

    #
```

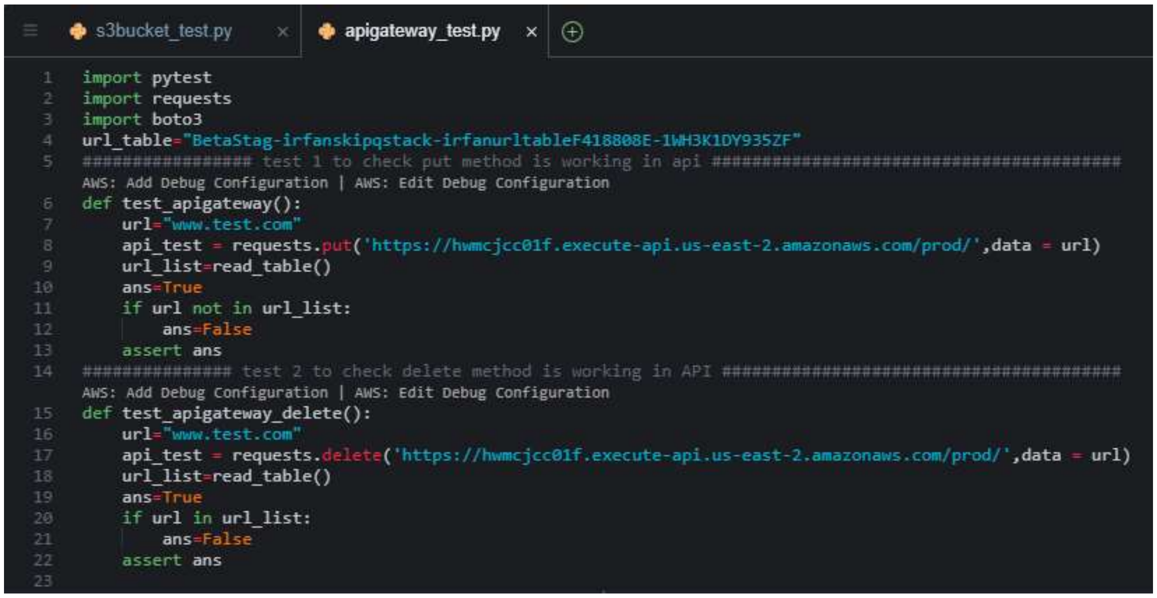
bash - "ip-172-31-36-136'

dynamodb_test.py

```
from sprint3_irfan.sprint3_irfan_stack import Sprint3IrfanStack
app=core.App()
Sprint3IrfanStack(app, 'Stack')
template=app.synth().get_stack_by_name('Stack').template
#unit test to check number of dynamo DB table
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def test_dynamodb():
    dynamodb_table = [resource for resource in template['Resources'].values()  if resource['Type']=="AWS::DynamoDB::Table"]
    assert len(dynamodb_table)>=2
    #assert 2==2  #
    #
```

Figure 57: Unit test

In the integration test check that our API gateway is working fine when the user tests any method. For that, I created 2 integration tests. First to check that put item is working fine by checking the item is inserted in the table after using the put method. In the second test, I am testing the delete item method

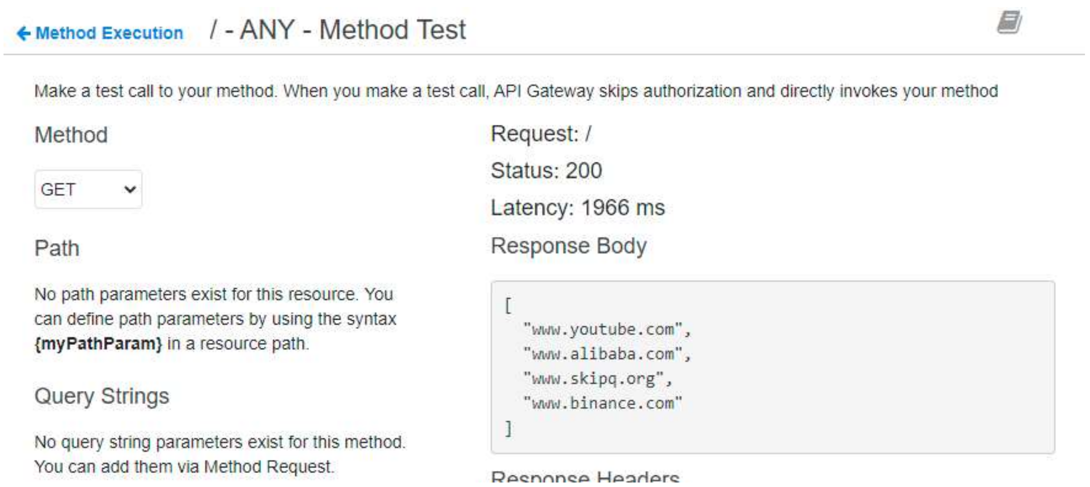


```python
import pytest
import requests
import boto3
url_table="BetaStag-irfanskipqstack-irfanurltableF418808E-1WH3K1DY935ZF"
################# test 1 to check put method is working in api ###########################################
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def test_apigateway():
    url="www.test.com"
    api_test = requests.put('https://hwmcjcc01f.execute-api.us-east-2.amazonaws.com/prod/',data = url)
    url_list=read_table()
    ans=True
    if url not in url_list:
        ans=False
    assert ans
############### test 2 to check delete method is working in API ##########################################
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def test_apigateway_delete():
    url="www.test.com"
    api_test = requests.delete('https://hwmcjcc01f.execute-api.us-east-2.amazonaws.com/prod/',data = url)
    url_list=read_table()
    ans=True
    if url in url_list:
        ans=False
    assert ans
```

Figure 58: Integration test

Test API Gateway

To test API Gateway, open the AWS API gateway service and search for your created API. Then test it by using the method you want to test. Here are some figures for testing API Gateway.



Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Method

GET

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Query Strings

No query string parameters exist for this method. You can add them via Method Request.

Request: /

Status: 200

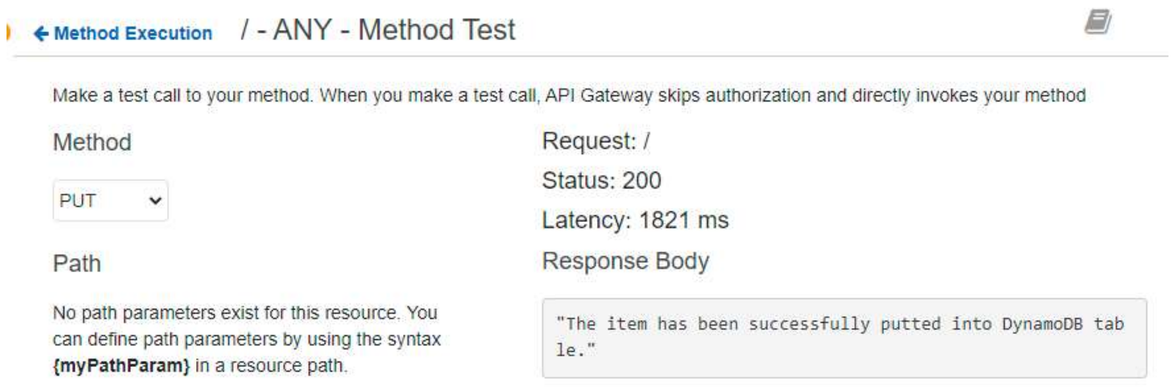
Latency: 1966 ms

Response Body

```
[
  "www.youtube.com",
  "www.alibaba.com",
  "www.skipq.org",
  "www.binance.com"
]
```

Response Headers

Figure 59: API Gateway GET Item method



Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Method: PUT

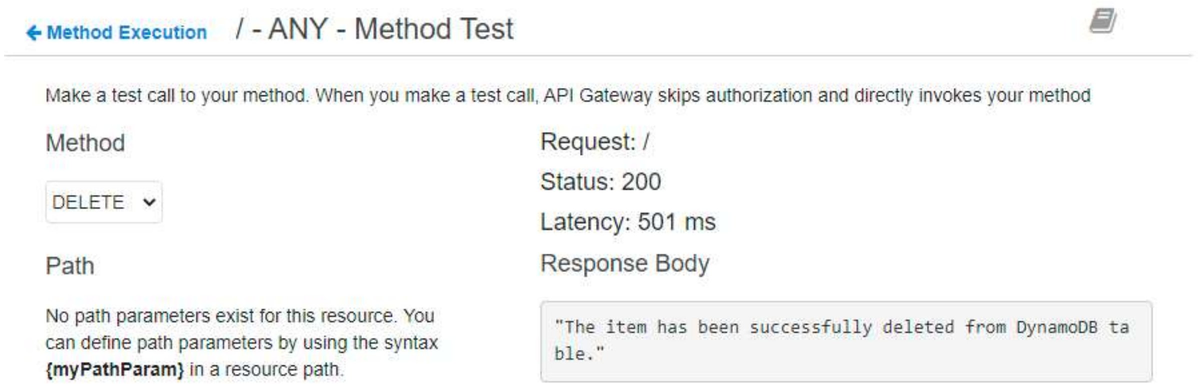
Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Request: /
Status: 200
Latency: 1821 ms
Response Body

```
"The item has been successfully putted into DynamoDB tab
le."
```

Figure 60:API Gateway PUT Item method



Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Method: DELETE

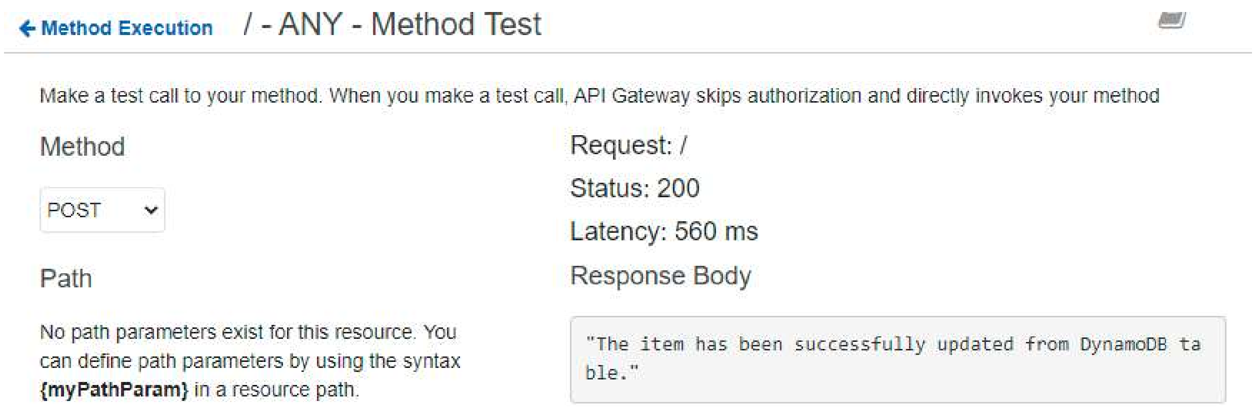
Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Request: /
Status: 200
Latency: 501 ms
Response Body

```
"The item has been successfully deleted from DynamoDB ta
ble."
```

Figure 61:API Gateway DELETE Item method



Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Method: POST

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Request: /
Status: 200
Latency: 560 ms
Response Body

```
"The item has been successfully updated from DynamoDB ta
ble."
```

Figure 62:API Gateway UPDATE Item method

Error and solution

During this project, I faced a few issues, and then I solved them with some solutions.

- Low memory space on your Local Machine: to solve this issue I deleted my other project.
- Lambda function for S3 bucket triggered trigger while uploading the JSON file. Add an event for the or lambda function.
- Lambda function for API was not triggered when I test it. I was not giving invoke permission to this lambda for API Gateway.
- Internal Issue when delete method is used in API Gateway. Check function name and the input arguments for the function.
- Issue in a unit test: when I run code it shows an error that cannot import module. I remove the __init__.py file and then it works.

References

- https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html
- https://www.youtube.com/watch?v=Ut5CkSz6NR0&t=1064s
- https://www.youtube.com/playlist?list=PLL2hlSFBmWwx7AFCvrurMhUOJc7kc0ynP
- https://dynobase.dev/dynamodb-python-with-boto3/#:~:text=To%20get%20all%20items%20from,the%20results%20in%20a%20loop.