



ProximaCentauri

Documentation



Written by:

Muhammad Irfan Hassan

Trainee @SkipQ2021

Table of Content

| | |
|--|-----------|
| Sprint 01: Web Health Monitor Application | 4 |
| Project Description | 4 |
| Technologies/Services | 4 |
| Cloud9 | 5 |
| Lambda | 5 |
| Cloud Watch | 5 |
| SNS | 5 |
| Dynamo DB | 6 |
| Setup | 6 |
| Project milestones | 7 |
| Task 01: Implementing Hello Lambda Function: | 7 |
| Task 02: Availability and latency of webpage by using periodic lambda function | 8 |
| Task 03: Alarm generate when threshold breached and send sns to subscribers | 9 |
| Task 04: Creating dynamo DB Table and Storing Alarm data to Table | 10 |
| Task 05: Read JSON file from S3 bucket | 12 |
| Task 06: Implement Project for customizing URLs List | 13 |
| Error and solution: | 14 |
| Unknown variable | 14 |
| Syntax Error | 14 |
| Insufficient data | 15 |
| References | 15 |
| Sprint 02: CI/CD Pipeline | 16 |
| Project Description | 16 |
| Technologies/Services | 16 |
| Code Pipeline | 16 |
| Secret Manager | 17 |
| Cloud Formation | 17 |
| Setup | 17 |
| Project milestones | 19 |

| | |
|--|-----------|
| Task 01: GitHub repo as Source in Pipeline and Build the source ----- | 19 |
| Task 02: Adding Beta stage with unit test in Pipeline ----- | 20 |
| Task 03: Adding Production stage with Manual Approval ----- | 21 |
| Task 04: Add Auto Rollback automation ----- | 22 |
| Results and Discussion ----- | 22 |
| Errors and Solution ----- | 24 |
| References ----- | 26 |
| Sprint 03: API Gateway endpoint for web crawler ----- | 27 |
| Project Description ----- | 27 |
| Technologies/Services ----- | 27 |
| API Gateway ----- | 27 |
| Project milestones ----- | 27 |
| Task 01: Create Lambda Function and Dynamo DB table ----- | 27 |
| Task 02: Upload JSON file from S3 bucket to Dynamo DB Table ----- | 28 |
| Task 03: Create API Gateway and add PUT/DELETE/UPDATE/GET method ----- | 29 |
| Task 04: Unit test and integration test ----- | 31 |
| Test API Gateway ----- | 32 |
| Error and solution ----- | 33 |
| References ----- | 34 |
| Sprint 04: Frond-End User Interface for CRUD API Gateway ----- | 35 |
| Description ----- | 35 |
| Technologies / Software ----- | 35 |
| Node JS ----- | 36 |
| Visual Studio Code ----- | 36 |
| React JS ----- | 36 |
| Chakra UI ----- | 36 |
| AWS Amplify ----- | 36 |
| Set-Up ----- | 37 |
| Project milestones ----- | 37 |
| Task 01: Create Front-end of App ----- | 37 |
| Task 02: Integrate the App to API Gateway ----- | 39 |

| | |
|--|-----------|
| Task 03: React Pagination and Search Function ----- | 39 |
| Task 04: Deploy React App using AWS Amplify----- | 40 |
| Task 05: Authorization on App using OAuth method ----- | 40 |
| Error and solution ----- | 41 |
| References ----- | 42 |
| Sprint 05: API Test Client for CRUD API Gateway using Pyresttes and Syntribos | 43 |
| Description ----- | 43 |
| Technologies/Libraries ----- | 43 |
| Docker ----- | 44 |
| AWS ECR----- | 44 |
| AWS ECS----- | 44 |
| PyRestTest ----- | 45 |
| Syntribos ----- | 45 |
| Set-Up ----- | 45 |
| Project milestones----- | 46 |
| Task 01: Build Docker image for PyRestTest.----- | 46 |
| Task 02: Publish build images to Elastic Container Registry (ECR) ----- | 48 |
| Task 03: Pull images from ECR and deploy using ECS ----- | 48 |
| Error and solution ----- | 49 |
| Build image error: ----- | 49 |
| AWS login failed: ----- | 49 |
| ECS task definition error: ----- | 49 |
| References ----- | 49 |

Sprint 01: 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, micro services, and event-driven server less 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, server less, 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
alias 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
```

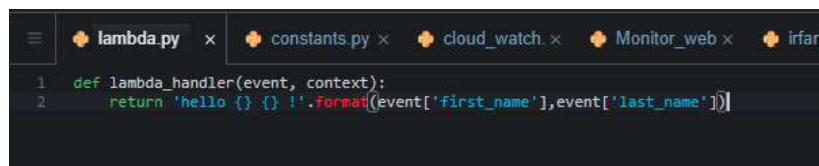
- Create a new folder resource and add a new lambda.py file.

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-name". code for Hello Lambda function is given below.

A screenshot of a code editor with several tabs at the top: 'lambda.py', 'constants.py', 'cloud_watch.py', 'Monitor_web.py', and 'Irfan...'. The 'lambda.py' tab is active, showing a Python function definition:

```
1 def lambda_handler(event, context):
2     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.

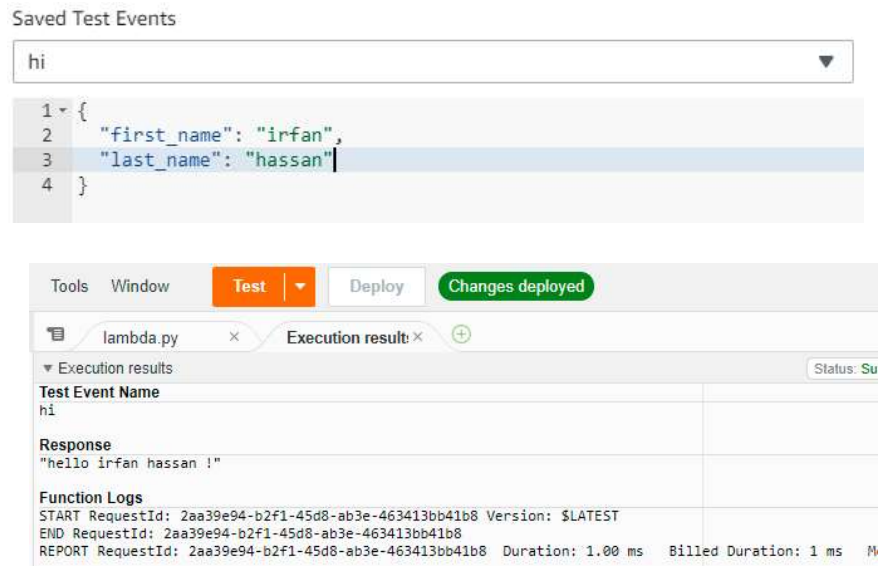


Figure 3: Testing hellolambda

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.

```

6
7 def lambda_handler(event, context):
8     value = dict()
9     cloudwatch = CloudWatch_PutMetric();
10    avail = availability_value()
11    Dimensions=[
12        {'Name': 'URL', 'Value': constant_URL}
13    ]
14    cloudwatch.put_data(constant_URL_NameSpace, constant_URL_Availbilty, Dimensions, avail)
15
16    latency = latency_value()
17    cloudwatch.put_data(constant_URL_NameSpace, constant_URL_Latency, Dimensions, latency)
18    value.update({"availability":avail,"latency":latency})
19    return value
20
21 def availability_value():
22     http = urllib3.PoolManager()
23     response = http.request("GET", constant_URL)
24     if response.status==200:
25         return 1.0
26     else:
27         return 0.0
28
29 def latency_value():
30     http = urllib3.PoolManager()
31     begin = datetime.datetime.now()
32     response = http.request("GET", constant_URL)
33     end = datetime.datetime.now()
34     duration = end - begin
35     latency_sec = round(duration.microseconds * 0.000001,6)
36     return latency_sec
37
38

```

Figure 4: Periodic lambda for availability and latency of webpage

```

1 import boto3
2 import constants
3
4
5 class CloudWatch_PutMetric:
6     def __init__(self):
7         self.client = boto3.client('cloudwatch')
8
9     def put_data(self, Space_Name, Metric_Name, Dimension, Value):
10        response = self.client.put_metric_data(
11            Namespace = Space_Name,
12            MetricData=[{ 'MetricName':Metric_Name, 'Dimensions':Dimension,'Value':Value}]
13        )

```

Figure 5:putting metric on Cloud watch

| | |
|---|--|
| Test Event Name | |
| test | |
| Response | |
| { | |
| "availability": 1, | |
| "latency": 0.302906 | |
| } | |
| Function Logs | |
| START RequestId: a0ce2b6b-5fbc-4d6c-8b59-73f1889a1dc6 Version: \$LATEST | |
| END RequestId: a0ce2b6b-5fbc-4d6c-8b59-73f1889a1dc6 | |
| REPORT RequestId: a0ce2b6b-5fbc-4d6c-8b59-73f1889a1dc6 Duration: 823.00 ms Billed Duration: 823 ms Memory Usage: 128 MB | |
| 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.

```

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
)

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
)

availability_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

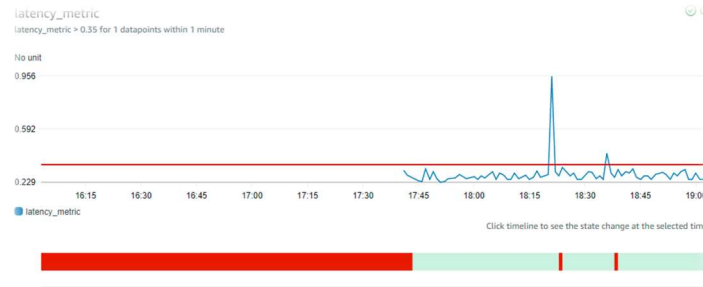


Figure 8: Alarm triggered when threshold breached

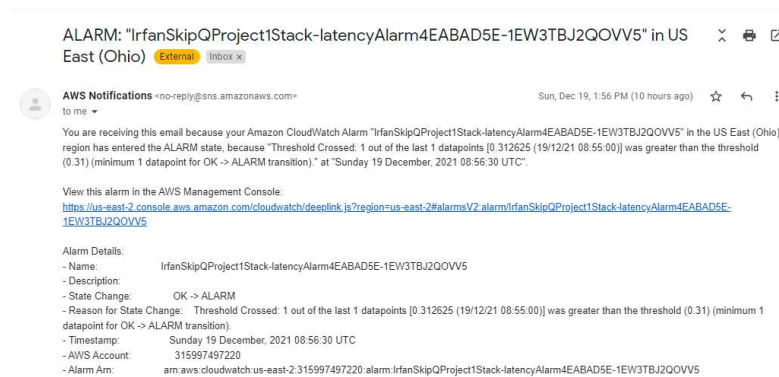


Figure 9: Email notification to the subscriber

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_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

```
dynamodb_lambda.py x
2 import json
3 import constants as constant_
4 client = boto3.client('dynamodb')
5
6 AWS: Add Debug Configuration | AWS: Edit Debug Configuration
7 def lambda_handler(event, context):
8     print(event)
9     client = boto3.client('dynamodb')
10    message = event['Records'][0]['Sns']
11    msg = json.loads(message['Message'])
12    client.put_item(
13        TableName = constant_.table_name,
14        Item={
15            'Timestamp':{'S': message['Timestamp']},
16            'Reason':{'S':msg['NewStateReason']}
17        })
```

Figure 12: Dynamo DB Lambda function

Here is the table we created.

► Irfandynamodb_Table

Expand to query or scan items.

View table details

Items returned (28)

↺

Actions ▼

Create item

<

1

>

⚙

🔍

| <input type="checkbox"/> | Timestamp ▼ | Reason ▼ |
|--------------------------|--------------------------|--|
| <input type="checkbox"/> | 2021-12-25T18:37:21.064Z | Threshold Crossed: 1 out of the last 1 datapoin... |
| <input type="checkbox"/> | 2021-12-25T19:02:16.720Z | Threshold Crossed: 1 out of the last 1 datapoin... |
| <input type="checkbox"/> | 2021-12-25T17:08:16.683Z | Threshold Crossed: 1 out of the last 1 datapoin... |
| <input type="checkbox"/> | 2021-12-25T17:13:16.814Z | Threshold Crossed: 1 out of the last 1 datapoin... |
| <input type="checkbox"/> | 2021-12-25T17:55:16.682Z | Threshold Crossed: 1 out of the last 1 datapoin... |
| <input type="checkbox"/> | 2021-12-25T18:08:16.778Z | Threshold Crossed: 1 out of the last 1 datapoin... |
| <input type="checkbox"/> | 2021-12-25T18:05:21.081Z | Threshold Crossed: 1 out of the last 1 datapoin... |

Figure 13: Alarm details in Dynamo DB Table

11 | Page

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.

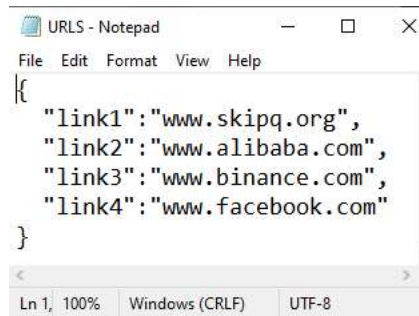


Figure 14: URLs in the JSON file

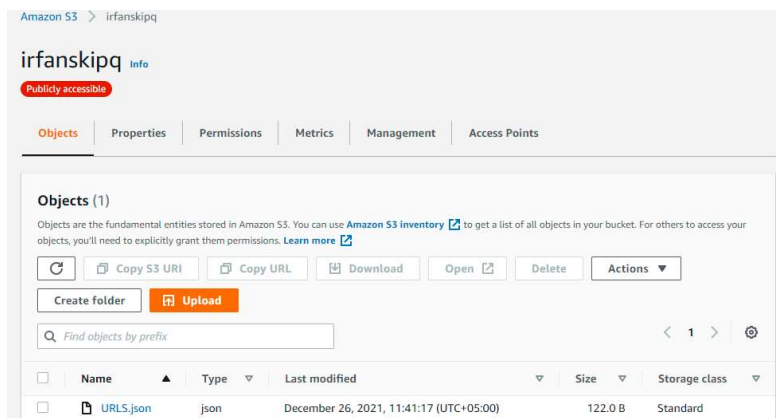


Figure 15:Json file in S3 Bucket

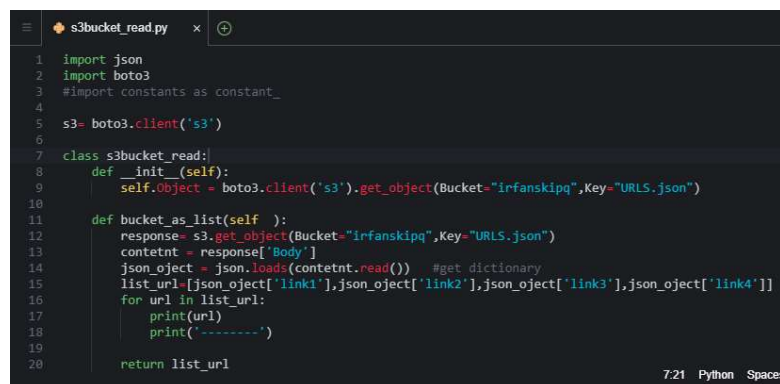


Figure 16: S3 bucket reading

Task 06: Implement Project for customizing URLs List

Our aim for this project was to implement Health Monitoring for a list of Customized 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 where 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 availability and latency
    Dimensions={'URL': list_url[index]}
    availability_metric=cloudwatch.Metric(namespace=constant_.URL_NameSpace,
    metric_name=constant_.URL_Aailibilty,
    dimensions_map=Dimensions,
    period=cdk.Duration.minutes(1),
    label=('availability_metric'+ ' '+list_url[index])
    )
    availability_Alarm=cloudwatch.Alarm(self,
    id ="AvailabilityAlarm"+ " "+list_url[index],
    metric = availability_metric,
    comparison_operator = cloudwatch.ComparisonOperator.LESS_THAN_THRESHOLD,
```

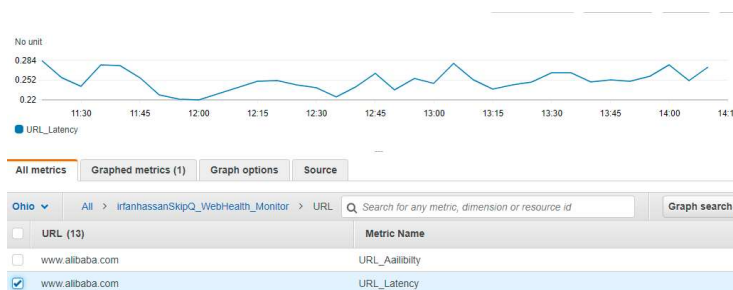
Figure 17: Adding for loop in the Main stack

```
Monitor_webhealth.py x
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 = availability_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({'availability':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.



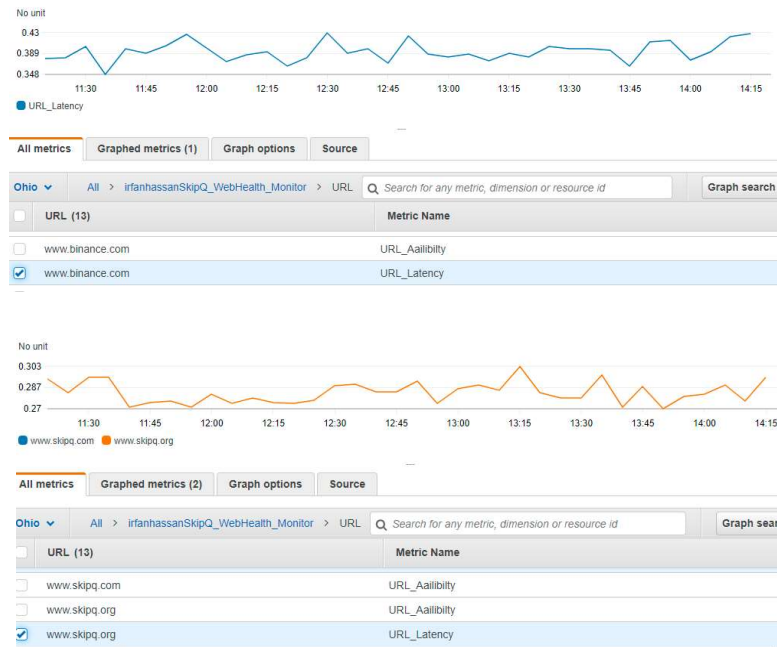


Figure 19: Latency of 3 URLs

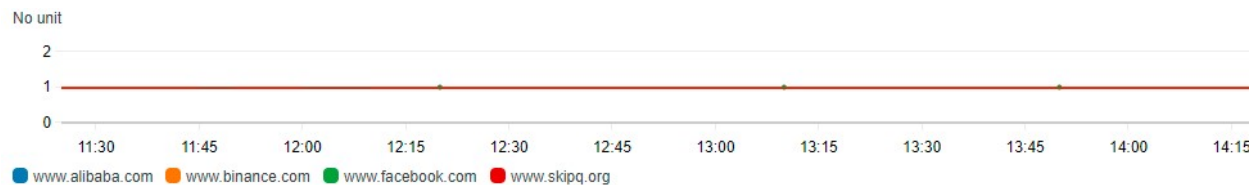


Figure 20: Latency for 4 URL

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](#)

Sprint 02: 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.

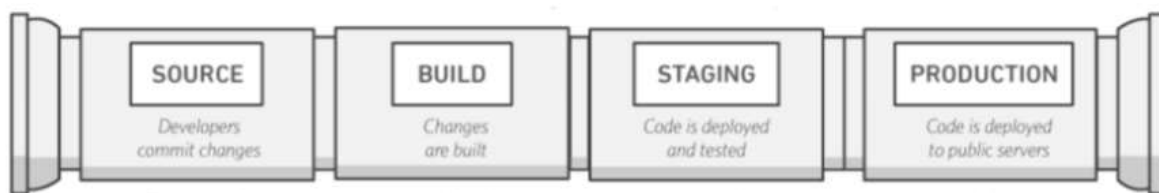


Figure 21: CI/CD Pipeline

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.

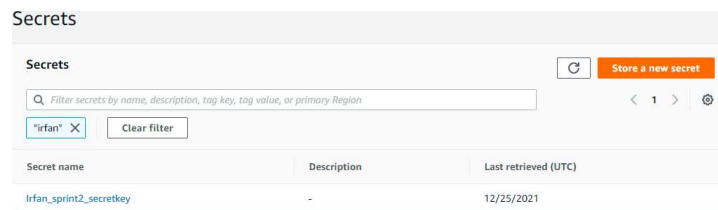


Figure 22: Secret manager

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

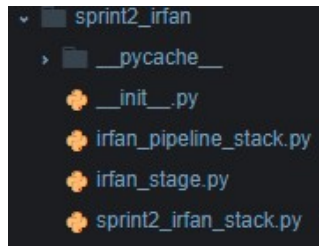


Figure 23: Files in stack folder

- Remove CDK.out from .gitignore file.

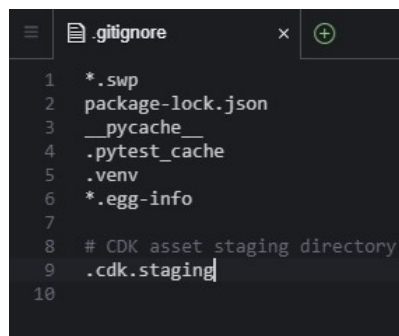


Figure 24: .gitignore file

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

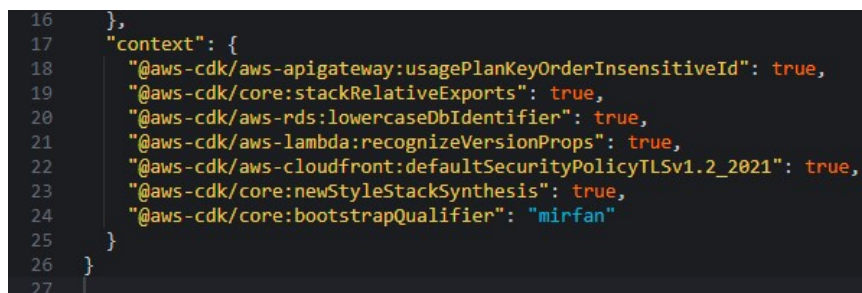


Figure 25: CDK JSON file

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

```

1  #!/usr/bin/env python3
2  import os
3
4  from aws_cdk import core
5
6  from sprint2_irfan.irfan_pipeline_stack import IrfanPipelineStack
7
8
9  app = core.App()
10 IrfanPipelineStack(app, "IrfanPipelineStack", env=core.Environment(account='315997497220', region='us-east-2'))
11
12 app.synth()
13

```

Figure 26: APP.py

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

```

X 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 pipeline (GitHub repository) #####
##
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

```

18 ##### Installing the requirement and Build the Source #####
19 synth = pipelines.ShellStep('synth', input= source,
20 commands = ["cd irfanhassan_skipq2021/Sprint2_irfan", "pip install aws-cdk,aws_cloudwatch_actions==1.135.0",
21 "pip install -r requirements.txt", "npm install -g aws-cdk", "cdk synth" ],
22 primary_output_directory = "Irfanhassan_skipq2021/Sprint2_irfan/cdk.out"
23 )
24 pipeline = pipelines.CodePipeline(self, 'pipeline', synth=synth)
25

```

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.

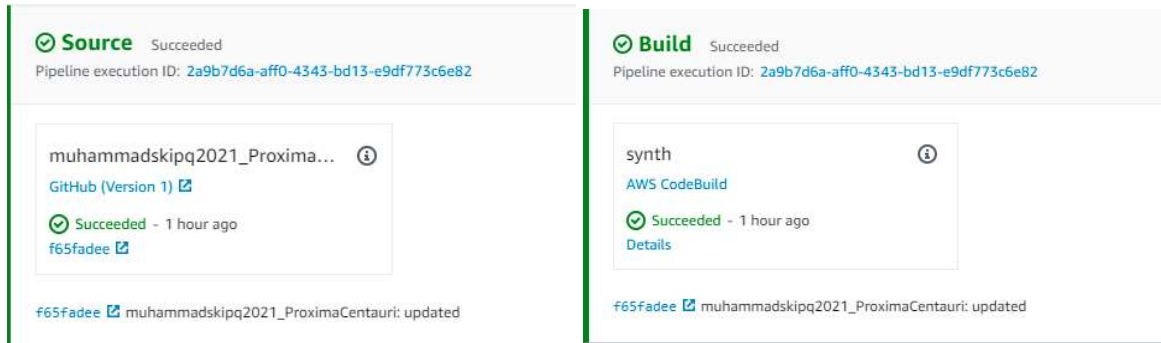


Figure 30: Output (Source and Build)

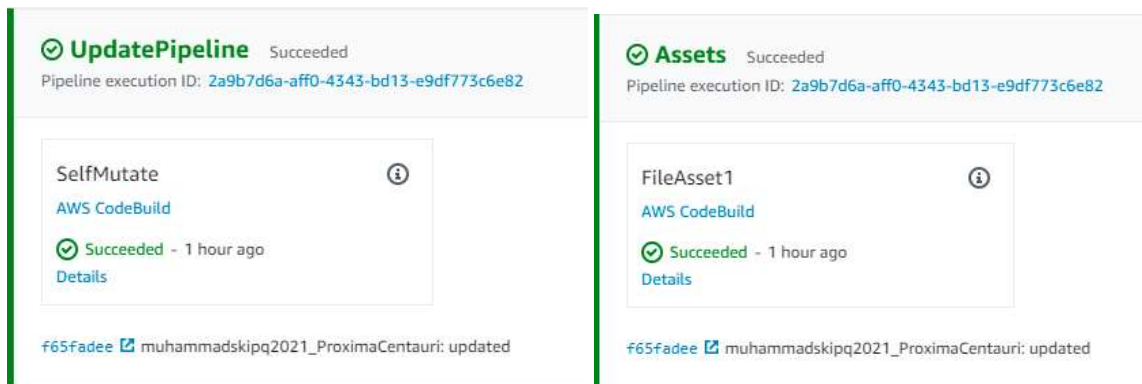


Figure 31: Pipeline updated and Assert are successful

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 stage, unit test and atthe dding to pipeline

```

1 from aws_cdk import core as cdk
2
3 from sprint2_irfan.sprint2_irfan_stack import Sprint2IrfanStack
4
5 class IrfanStage(cdk.Stage):
6     def __init__(self, scope: cdk.Construct, construct_id: str, **kwargs) -> None:
7         super().__init__(scope, construct_id, **kwargs)
8
9         irfan_stack = Sprint2IrfanStack(self, 'irfanskipqstack')

```

Figure 33: Pipeline Stage class

Now again commit the code and push to GitHub.

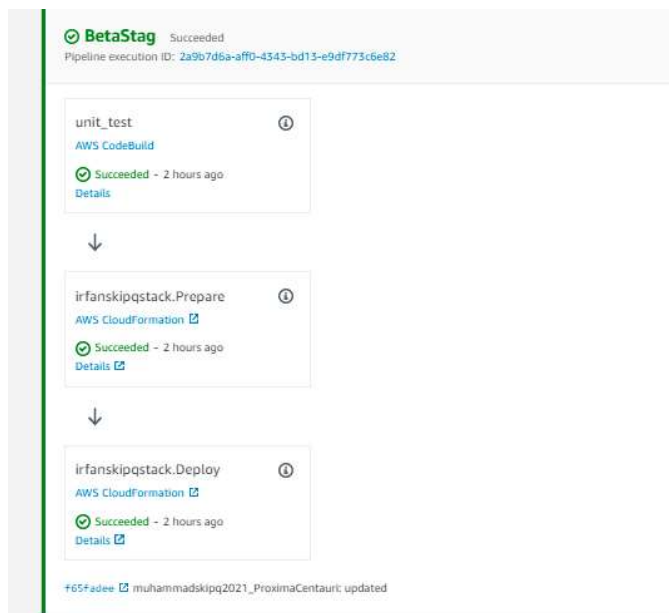


Figure 34: Beta stage is added successfully

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.

```

34 ##### Addign Prodution stage with mannaul approval in Pipeline #####
35 prodstage= IrfanStage(self, "ProdStage", env={'account':'315997497220','region': 'us-east-2'} )
36 pipeline.add_stage(prodstage, pre=[ pipelines.ManualApprovalStep("PromoteToProd") ])
37
38

```

Figure 35: Adding Production stage to pipeline

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

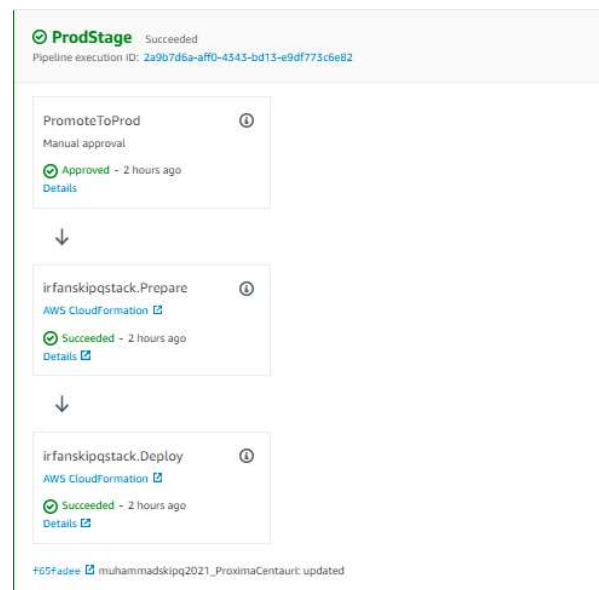


Figure 36: Adding Production stage

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.

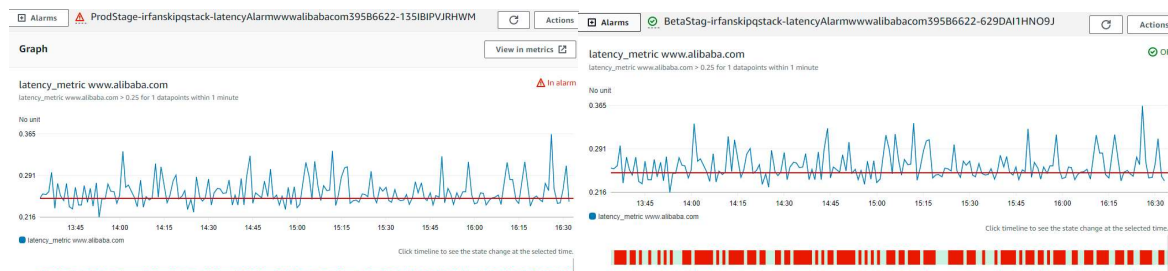


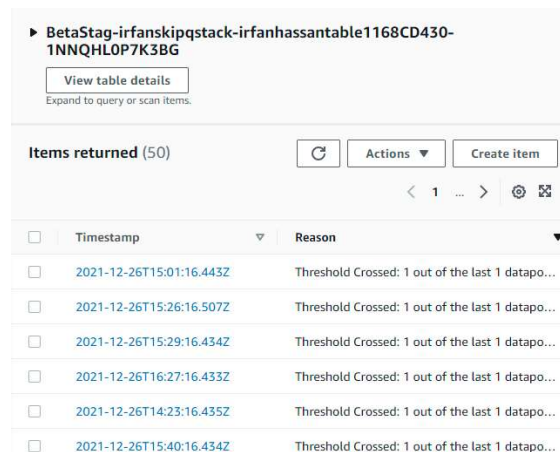
Figure 37: Plot of Latency for alibab.com for Beta and Prod stages

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="BetaStage-irfanskipqstack-irfanhassantable1168CD430-1NNQHL0P7K3BG"
prod_tabel="ProdStage-irfanskipqstack-irfanhassantable1168CD430-7BYVC0V4LAAG"
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.



| | Timestamp | Reason |
|--------------------------|--------------------------|--|
| <input type="checkbox"/> | 2021-12-26T15:01:16.443Z | Threshold Crossed: 1 out of the last 1 datapo... |
| <input type="checkbox"/> | 2021-12-26T15:26:16.507Z | Threshold Crossed: 1 out of the last 1 datapo... |
| <input type="checkbox"/> | 2021-12-26T15:29:16.434Z | Threshold Crossed: 1 out of the last 1 datapo... |
| <input type="checkbox"/> | 2021-12-26T16:27:16.433Z | Threshold Crossed: 1 out of the last 1 datapo... |
| <input type="checkbox"/> | 2021-12-26T14:23:16.435Z | Threshold Crossed: 1 out of the last 1 datapo... |
| <input type="checkbox"/> | 2021-12-26T15:40:16.434Z | Threshold Crossed: 1 out of the last 1 datapo... |

Figure 39: Beta stage Table

ProdStage-irfanskipqstack-irfanhassantable1168CD430-7BYC0V4LAAG

[View table details](#)
Expand to query or scan items.

Items returned (50) [Refresh](#) [Actions](#) [Create item](#)

< 1 > [Filter](#) [Columns](#)

| <input type="checkbox"/> | Timestamp | Reason |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | 2021-12-26T16:35:25.971Z | Threshold Crossed: 1 out of the last 1 datapoi... |
| <input type="checkbox"/> | 2021-12-26T15:52:25.978Z | Threshold Crossed: 1 out of the last 1 datapoi... |
| <input type="checkbox"/> | 2021-12-26T16:07:25.960Z | Threshold Crossed: 1 out of the last 1 datapoi... |
| <input type="checkbox"/> | 2021-12-26T15:46:25.977Z | Threshold Crossed: 1 out of the last 1 datapoi... |
| <input type="checkbox"/> | 2021-12-26T16:25:26.091Z | Threshold Crossed: 1 out of the last 1 datapoi... |
| <input type="checkbox"/> | 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.

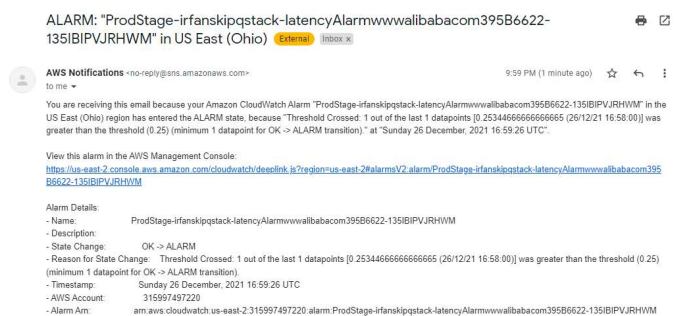


Figure 41: Alarm Notification at Email

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
irfanskip@pipelineStack: 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.

X Irfanskip@pipelineStack failed: Error: Irfanskip@pipelineStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment 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/cloudformation-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/cloudformation-deployments.ts:286:7)
at CloudFormationDeployments.deployStack (/home/ubuntu/.nvm/versions/node/v16.3.0/lib/node_modules/aws-cdk/lib/api/cloudformation-deployments.ts:282:15)
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)
Irfanskip@pipelineStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment been bootstrapped? Please run 'cdk bootstrap' (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.

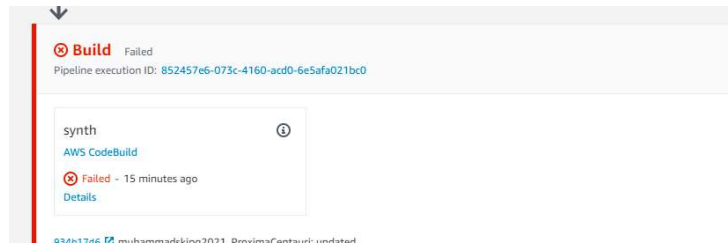


Figure 43: Build failed: Error

- 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.



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.

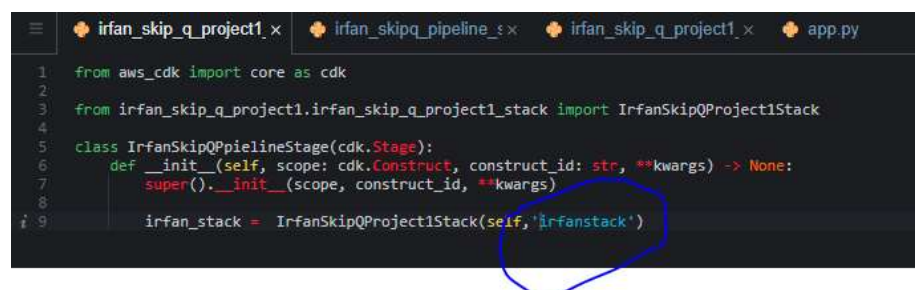


Figure 45: Adding stage: Error

- While running pytest unit test not able to import aws_core. install requirements using "npm install -g aws-cdk" and "pip install -r requirements.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/irfan/lambda_test.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
unittest/lambda_test.py:2: in <module>
    from aws_cdk import core
    ^
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=sTTvZ5ItZG0>
- [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&DAFO=1&PC=U531

Sprint 03: API Gateway endpoint for web crawler

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('urllambda', './resources', 's3_dynamodb_lambda.lambda_handler', db_lambda_role)
url_table = self.create_table(id='urltable', key=db.Attribute(name='URL', type=db.AttributeType.STRING))
url_lambda.grant_full_access(url_table)
url_lambda.add_environment('table_name', url_table.table_name)
table_name=url_table['TableDescription']['TableName']
#####
adding s3bucket event to trigger url lambda
bucket = s3.Bucket(self, "url3bucket")
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

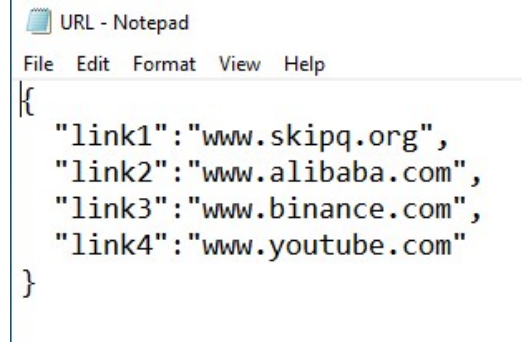
In Lambda Function for S3to dynamo DB, we get bucket name and key. Then read content from JSON file and store the content into dynamo DB table.

Here is code.

```
1 import boto3
2 import os
3 from s3bucket_read import s3bucket_read as bucket
4
5 AWS: Add Debug Configuration | AWS: Edit Debug Configuration
6 def lambda_handler(event,context):
7     value = dict()
8     bucketname = event['Records'][0]['s3']['bucket']['name']
9     filename = event['Records'][0]['s3']['object']['key']
10
11     client = boto3.client('dynamodb')
12     list_url=bucket(bucketname,filename).bucket_as_list()
13     tablename = os.getenv('table_name')#getting table name
14     for url in list_url:
15         client.put_item(Table=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

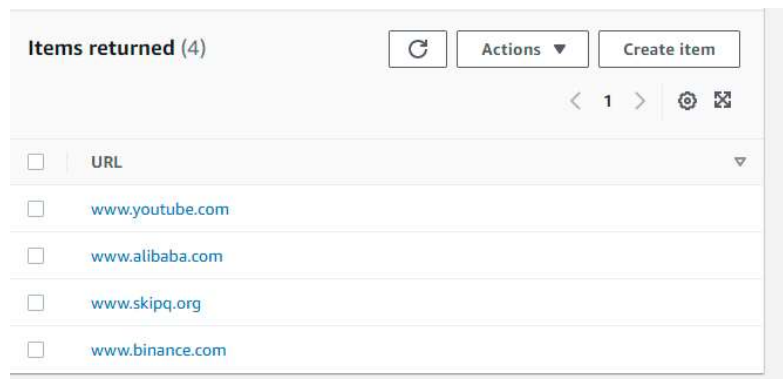


Figure 50: Dynamo DB table after JSON file is uploaded in an S3 bucket

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 add 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") # DELETE 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%20
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/quest
        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

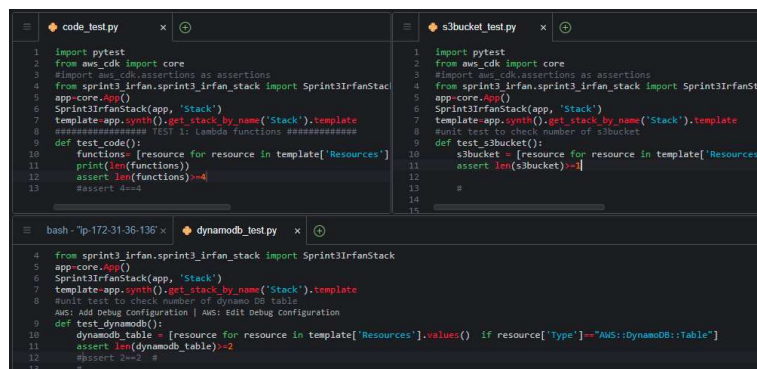
In update Item method, we ask user to enter two URL separated by comma and in lambda function, first we separate the URL using split function and check if URL user want to update is available in table or not. If it is available, then update the URL with new URL and if URL is not available then send notification 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 available 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 available 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
1 import pytest
2 from aws_cdk import core
3 import pytest
4 from sprint3_irfan.sprint3_irfan_stack import Sprint3IrfanStack
5 app=core.App()
6 Sprint3IrfanStack(app, 'Stack')
7 template=app.synth().get_stack_by_name('Stack').template
8 ##### TEST 1: Lambda functions #####
9 def test_code():
10     functions=[resource for resource in template['Resources']]
11     print(len(functions))
12     assert len(functions)==4
13     #assert 4==4

s3bucket_test.py
1 import pytest
2 from aws_cdk import core
3 import pytest
4 from sprint3_irfan.sprint3_irfan_stack import Sprint3IrfanStack
5 app=core.App()
6 Sprint3IrfanStack(app, 'Stack')
7 template=app.synth().get_stack_by_name('Stack').template
8 #unit test to check number of s3bucket
9 def test_s3bucket():
10     s3bucket=[resource for resource in template['Resources']]
11     assert len(s3bucket)==3
12     #
13
14
15

dynamodb_test.py
1 from sprint3_irfan.sprint3_irfan_stack import Sprint3IrfanStack
2 app=core.App()
3 Sprint3IrfanStack(app, 'Stack')
4 template=app.synth().get_stack_by_name('Stack').template
5 #unit test to check number of dynamo db table
6 #AWS: Add debug configuration | AWS: Edit debug configuration
7 def test_dynamodb():
8     dynamodb_table=[resource for resource in template['Resources'].values() if resource['Type']=='AWS::DynamoDB::Table']
9     assert len(dynamodb_table)==2
10     #assert 2==2
11
12
13
```

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

```
s3bucket_test.py x apigateway_test.py x +
1 import pytest
2 import requests
3 import boto3
4 url_table="BetaStag-irfanskipqstack-irfanurlltableF418808E-1W43K10Y935ZF"
5 ##### test 1 to check put method is working in api #####
6 def test_apigateway():
7     url="www.test.com"
8     api_test = requests.put('https://hmmcjcc01f.execute-api.us-east-2.amazonaws.com/prod/',data = url)
9     url_list=read_table()
10    ans=True
11    if url not in url_list:
12        ans=False
13    assert ans
14    ##### test 2 to check delete method is working in API #####
15    AWS: Add Debug Configuration | AWS: Edit Debug Configuration
16    def test_apigateway_delete():
17        url="www.test.com"
18        api_test = requests.delete('https://hmmcjcc01f.execute-api.us-east-2.amazonaws.com/prod/',data = url)
19        url_list=read_table()
20        ans=True
21        if url in url_list:
22            ans=False
23        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.

Method Execution / - ANY - Method Test

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

| | |
|--|---|
| Method | Request: / |
| GET | Status: 200 |
| Path | Latency: 1966 ms |
| No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path. | Response Body |
| Query Strings | ["www.youtube.com", "www.alibaba.com", "www.skipq.org", "www.binance.com"] |
| No query string parameters exist for this method. You can add them via Method Request. | Response Headers |

Figure 59: API Gateway GET Item method

← Method Execution / - ANY - Method Test

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

| | |
|--|--|
| Method | Request: / |
| PUT | Status: 200 |
| Path | Latency: 1821 ms |
| No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path. | Response Body |
| | "The item has been successfully putted into DynamoDB table." |

Figure 60:API Gateway PUT Item method

← Method Execution / - ANY - Method Test

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

| | |
|--|---|
| Method | Request: / |
| DELETE | Status: 200 |
| Path | Latency: 501 ms |
| No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path. | Response Body |
| | "The item has been successfully deleted from DynamoDB table." |

Figure 61:API Gateway DELETE Item method

← Method Execution / - ANY - Method Test

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

| | |
|--|---|
| Method | Request: / |
| POST | Status: 200 |
| Path | Latency: 560 ms |
| No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path. | Response Body |
| | "The item has been successfully updated from DynamoDB table." |

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-dynamodb.html>
- <https://www.youtube.com/watch?v=Ut5CkSz6NR0&t=1064s>
- <https://www.youtube.com/playlist?list=PLL2hISFBmWwx7AFCvrurMhUOJc7kc0ynP>
- <https://dynobase.dev/dynamodb-python-with-boto3/#:~:text=To%20get%20all%20items%20from,the%20results%20in%20a%20loop.>

Sprint 04: Frond-End User Interface for CRUD API Gateway

Description

Building a Front-End user interface for CRUD API Gateway using React JS and Chakra UI. This React APP will allow the user to search, put, delete and get URLs from the Dynamo DB table. The React APP will be integrated into CRUD API Gateway and will send GET requests. Log in access will be enabled using the OAuth method.

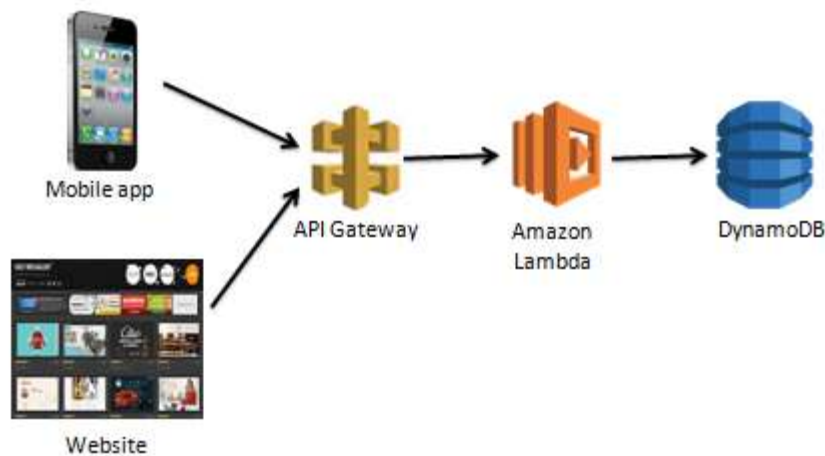


Figure 63: Flow Diagram for Front-end User interface for CRUD API Gateway

Technologies / Software

- Node JS
- Visual Studio Code
- React JS
- Chakra UI
- AWS Amplify

Node JS

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command-line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.

Visual Studio Code

Visual Studio Code is a source-code editor made by Microsoft for Windows, Linux, and Mac OS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

React JS

React (also known as React.js or React JS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. React can be used as a base in the development of single-page or mobile applications.

Chakra UI

Chakra UI is a component-based library. It's made up of basic building blocks that can help you build the front-end of your web application. It is customizable and reusable, and most importantly it supports React Js, along with some other libraries too.

AWS Amplify

AWS Amplify is a set of purpose-built tools and features that lets front-end web and mobile developers quickly and easily build full-stack applications on AWS, with the flexibility to leverage the breadth of AWS services as your use cases evolve. With Amplify, you can configure a web or mobile app backend, connect your app in minutes, visually build a web frontend UI, and easily manage app content outside the AWS console. Ship faster and scale effortlessly—with no cloud expertise needed.

Set-Up

DO following steps to complete the project.

- Install Node JS and visual studio code.
- Run command to create App

```
$ npm create-react-app my-app
```

- Change directory to my-app using this command

```
$ cd my-app
```

- Run this command to install Chakra UI Library” npm i @chakra-ui/react @emotion/react@^11 @emotion/styled@^11 framer-motion@^5”
- Now remove all files from the src folder except App.js and index.js
- Write code in App.js for react app.
- Now check your app by running this command

```
$ npm start
```

- Once your app is done. Build an app using this command.

```
$ npm run build
```

- Make zip folder from build folder and upload to S3 Bucket/GitHub Repo.
- Go to AWS Amplify service and deploy the zip file from S3 Bucket/GitHub Repo.
- Go to the App link and test your React App.

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: Create Front-end of App

The first task is to create Front-end of App using chakra UI and React Js Libraries. I added the component to my App.

- **Color Mode Icons:** I added Sun/Moon icon to change the color mode of react app from dark to light and vice versa.

- **Headline:** I added a headline to show the User App Description.
- **Input Box:** I added an input box where users can enter an URL to search/put/delete from the dynamo DB table.
- **Search URL Button:** To start search I added a button that enables search when the user clicks on it.
- **Put URL Button:** To add new URL in Dynamo DB table, I added a button that enables put URL when the user clicks on it.
- **Delete URL Button:** To delete URL from Dynamo DB table, I added a button that enables delete URL when the user clicks on it.
- **Get URL Button:** To get all URLs stored in the Dynamo DB table, I added a button that enables to show URLs in pagination when the user clicks on it.

Here is a screenshot of my app.



Figure 64: Front-End User Interface (dark mode)

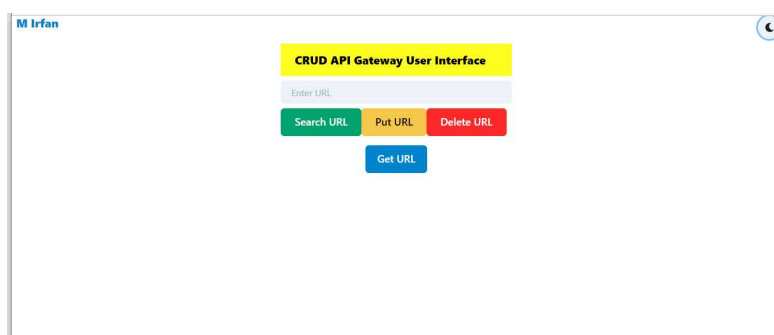


Figure 65: Front-End User Interface (light mode)

Task 02: Integrate the App to API Gateway

After completing the front-end design for the app. Now I have to integrate my front-end to API-gateway to perform search and get requests. I used “Axios” to get data from API Gateway. I am Here being a code screenshot.

```
useEffect(() => {  
  const fetchPosts= async() => {  
    const res=await axios({method: 'GET',  
      headers: { 'Content-Type': 'application/json' },  
      url:"https://hwmcc01f.execute-api.us-east-2.amazonaws.com/prod/"});  
    setPosts(res.data);  
  }  
  fetchPosts();  
},[]);
```

Figure 66: getting data from API Gateway using axios

I also added code for the functionality of each component using conditions like input box, search button, put URL, delete URL and get URLs button.

Task 03: React Pagination and Search Function

Now I have to display the URLs in the form of pages. I used React pagination library to implement the pagination. After getting the data from API Gateway we are storing data in an array. We set the number of URLs. page and then calculated the total number of pages we need. Using the Item function, we display the Current Item for each page on our react app. And then added pagination numbers and signs to allow the user to jump to other pages when he clicks on page number or sign.

Here is the output of Pagination.

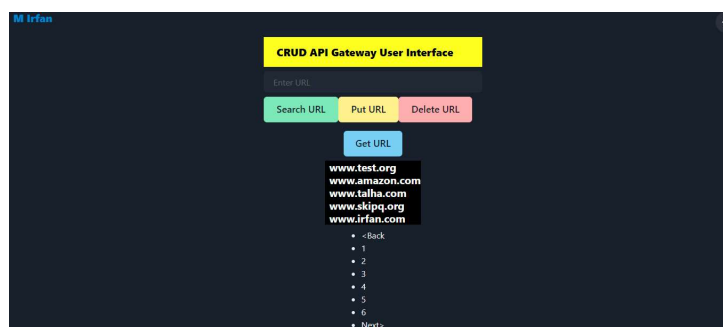


Figure 67: Pagination for displaying URL list to User

Now I have URL data in Array. So I added a function which checks search URL is available in Array or not and displays a message to a user when he clicks on the Search URL button. Here the is output screenshot.

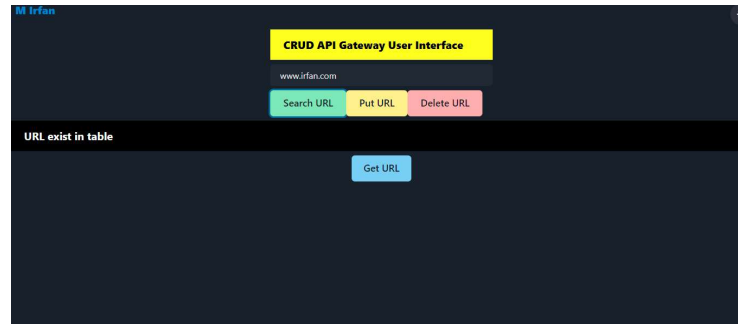


Figure 68: Output result for searching

Task 04: Deploy React App using AWS Amplify

Now we have a working app and we have to deploy it using AWS Amplify. To do this I build the app and then uploaded the zip folder of the build file in the S3 bucket and deploy the build zip file in AWS Amplify Service. As result, we get the Domain link for our React app. Here is the output after deploying the App in AWS amplify.

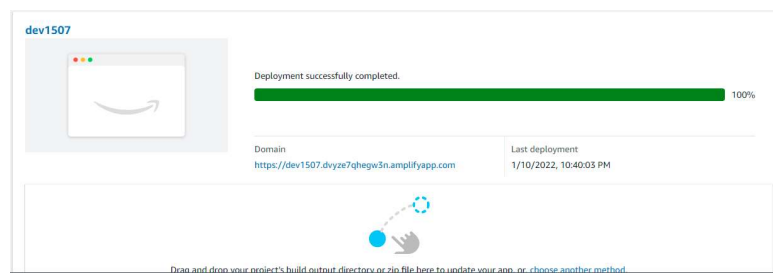


Figure 69: AWS Amplify deploying result

Task 05: Authorization on App using OAuth method

I have to add authorization to react app using the OAuth method. To do this I set a username and password to access react app in the domain.

Access control

Restrict access to your branches with a username and password. [Learn more](#)

Access control settings

Manage access

< 1 >

| Branch name | Access setting | Username | Password |
|-------------|-------------------------|----------------------|----------|
| dev1507 | Global password enabled | irfanhassanskipq2021 | ***** |

Figure 70: Authorization using OAuth method

Error and solution

App name error: Always use the lower case letter for app name while creating react app.

```
C:\Users\Muhammad Irfan Hassa>create-react-app CRUD_API_Gateway
Cannot create a project named "CRUD_API_Gateway" because of npm naming restrictions:
  * name can no longer contain capital letters
Please choose a different project name.
C:\Users\Muhammad Irfan Hassa>create-react-app crud-apigateway
```

Figure 71:App name error

Not define/unknown: install library if you are using in-app or declare variable if you are using variable somewhere.

Block by CORS policy: when using axios get to get data from API Gateway, this error generated. To solve this error, add the extension “CORS unblock” to your default browser.

```
Access to XMLHttpRequest at 'https://4jd8g9kea3.execute-api.us-east-2.amazonaws.com/prod' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

Figure 72:Block by CORS policy



Figure 73: CORS extension

App not working: After deploying, I got this error. I used “HTTP” in my API Gateway link so I changed it to “HTTPS” and it works.

```
useEffect(() => {
  const fetchPosts= async() => {
    const res=await axios({method: 'GET',
      headers: { 'Content-Type': 'application/json' },
      url: 'https://hwmcc01f.execute-api.us-east-2.amazonaws.com/prod/'});
    setPcsts(res.data);
  }
  fetchPosts();
}, []);
```

Figure 74: HTTPS to API Gateway Link

References

- [React tutorial for beginners #15 Get Input box value - YouTube](#)
- https://www.tutorialspoint.com/reactjs/reactjs_pagination.htm
- <https://www.youtube.com/watch?v=IYCa1F-OWmk&t=608s>
- https://www.youtube.com/watch?v=NEYrSUM4Umw&ab_channel=Codevolution
- <https://www.npmjs.com/package/react-paginate>
- <https://www.freecodecamp.org/news/fetch-data-react/>
- [Chakra UI - A simple, modular and accessible component library that gives you the building blocks you need to build your React applications. - Chakra UI \(chakra-ui.com\)](#)
- [React Tutorial \(w3schools.com\)](#)

Sprint 05: API Test Client for CRUD API Gateway using Pyresttes and Syntribos

Description

Create API test client using Pyresttest and Syntribos to exercise the CRUD API Gateway operations (GET, PUT AND DELETE). Build images using Docker and push build images to Elastic Container Registry (ECR). Pull and deploy images using Pipeline and send test data to Cloud watch metrics and add alarm to metrics.

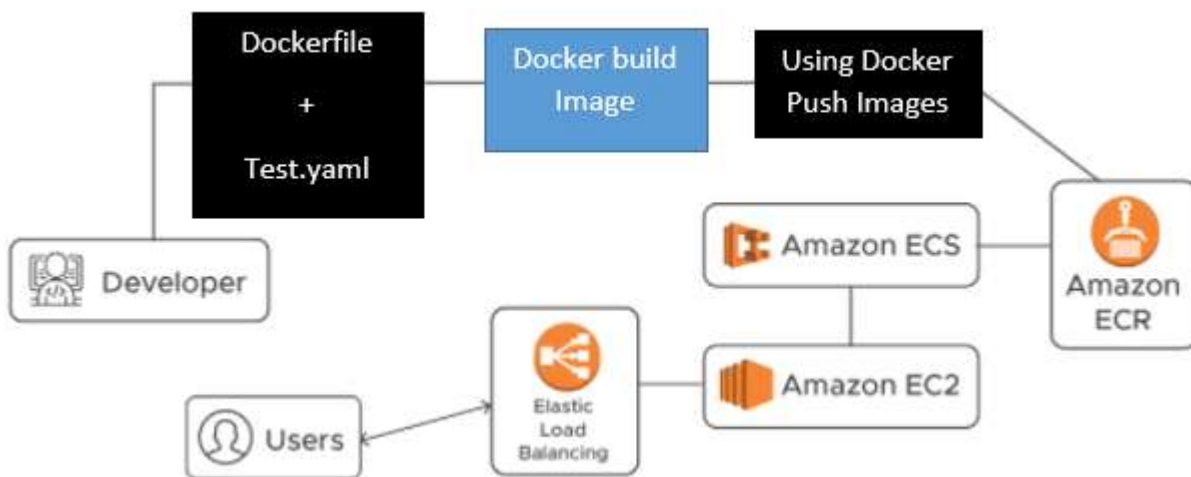


Figure 75: Flow diagram of project

Technologies/Libraries

- Docker
- AWS ECR
- AWS ECS
- PyRestTest
- Syntribos

Docker

Docker, Inc. is an American technology company that develops productivity tools built around Docker, which automates the deployment of code inside software containers.

Purpose of using Docker

Most applications are made to run on different machines for everyone, with or without software engineering knowledge. So, Docker comes in handy when developing such kind of an application.

Docker Container Vs Virtual Machine

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.

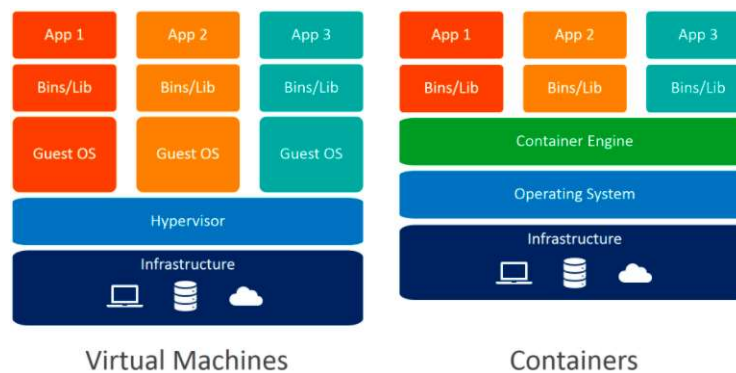


Figure 76: Container Vs Virtual Machine

AWS ECR

Amazon Elastic Container Registry (Amazon ECR) is integrated with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS), which means you can easily store and run container images for applications with either orchestrator. All you need to do is specify the Amazon ECR repository in your task or pod definition for Amazon ECS or Amazon EKS to retrieve the appropriate images for your applications.

AWS ECS

Amazon Elastic Container Service (Amazon ECS) allows you to easily deploy containerized workloads on AWS. The powerful simplicity of Amazon ECS enables you to grow from a single Docker container to managing your entire enterprise application portfolio.

PyRestTest

It's a small Python library with an impressive amount of functionality that goes beyond just checking if a REST service is up to check that it is behaving correctly.

PyRestTest allows you to collect low-level network performance metrics from Curl itself. Benchmarks are based on tests: they extend the configuration elements in a test, allowing you to configure the REST call similarly. However, they do not perform validation on the HTTP response, instead, they collect metrics.

Syntribos

Syntribos is an open-source automated API security testing tool that is maintained by members of the OpenStack Security Project.

Given a simple configuration file and an example HTTP request, Syntribos can replace any API URL, URL parameter, HTTP header, and request body field with a given set of strings. Syntribos iterates through each position in the request automatically. Syntribos aims to automatically detect common security defects such as SQL injection, LDAP injection, buffer overflow, etc.

Set-Up

Here are all steps to set up for this project.

1. First Install Visual Studio, Python, Docker Desktop, and AWS CLI on your PC/Laptop.
2. Open CMD and create a directory using this command and change the directory to that directory.

```
$mkdir example-project
```

```
$cd example-project
```

3. Create a new file with the name Docker file and write code.
4. Create a new file prettiest.yaml and add code to it for your test.
5. Now open CMD and run this command to build a Docker image from a Docker file using this command.

```
$docker build -t image-name .
```

6. Check image is created by using this command

```
$docker images
```

7. If your image is there, then run the image using the below command.

```
$docker run image-name
```

8. If the image is running successfully then move onward.

9. Create an ECR repository and set the name as your image name.

10. Create IAM user and add policy and save excel file having AWS access key ID and AWS access key. Then run this command in CMD to configure AWS.

```
$aws configure
```

11. And give access key ID and Access key and then give other things accordingly to your account and requirements.

12. Go into your ECR repo and click on the option push command. There will be 4 commands. Run all commands one by one and your image will be pushed to ECR.

13. Now add code in stack file to pull the image and deploy using AWS ECS and then deploy pipeline.

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: Build Docker image for PyRestTest.

We created a directory and then add two files (Docker file and pyresttest.yaml). Add code in Pyresttest.yaml as shown below.

```

! PyrestTestyaml
1 ---
2 - config:
3   - testset: "Benchmark tests for Pyresttest Image"
4   #-----
5   #adding benchmark to our API Gateway GET method
6   #-----
7   - benchmark:
8     - name: "GET Items test"
9     - headers: { "ContentType": "application/json" }
10    - url: "https://ndeugavtv6.execute-api.us-east-2.amazonaws.com/prod/" #api gateway link
11    - method: "GET"
12    - warmup_runs: 1
13    - "benchmark_runs": "7" #kust random number
14    - output_format: csv
15    - metrics:
16      - total_time
17  #-----

```

Figure 77: Test bench for Get method on API

In this test we are setting benchmark for API get request. We will run test for 7 times and it will return time for each test. We can add more test by adding more test benchmark in it.

In Docker file, we have to install all required library to run our pyresttest. So here is code for Docker file.

```

Dockerfile > ...
1 FROM python:2-alpine
2 ENV PYCURL_SSL_LIBRARY=openssl
3
4 RUN apk add --no-cache --update openssl curl
5 RUN apk add --no-cache --update --virtual .build-deps build-base python-dev curl-dev
6 RUN pip install jmespath jsonschema pyresttest
7 RUN apk del .build-deps
8
9 WORKDIR /Irfan
10
11 COPY . /Irfan/
12
13 ENTRYPOINT [ "pyresttest" ]
14
15

```

Figure 78: Docker file for pyresttest

Here I am using python2 with alpine. And then install curl, dependencies, and prettiest libraries.

Then we create a working directory and then add all files into our work directory and as an entry point, we set Pyresttest. Then build an image and test it. Here is the output for our test.

```

C:\Users\Muhammad Irfan Hassa\Irfan-Pyresttest>docker run --rm irfan-pyresttest https://ndeugavtv6.execute-api.us-east-2.amazonaws.com/prod/ PyrestTest.yaml
{"failures": 0, "aggregates": [], "group": "Default", "results": {"total_time": [1.024304, 0.931768, 0.956354, 0.897499, 0.839325, 0.636073, 1.129672]}, "name": "GET Items test"}

C:\Users\Muhammad Irfan Hassa\Irfan-Pyresttest>docker images
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
irfan-pyresttest    latest       9af693d196d0   25 hours ago   315MB
315997497220.dkr.ecr.us-east-2.amazonaws.com/hello-world latest       88340bd4c94d   3 days ago     215MB
hello-world         latest       88340bd4c94d   3 days ago     215MB
<none>              <none>       478ffae86a68   3 days ago     169MB

```

Figure 79: Build image and test result

We can see the result after running the test on our build image of pyresttest. We get time for each run of the test.

Task 02: Publish build images to Elastic Container Registry (ECR)

After testing build images in the local machine, we have to publish the images to the elastic container registry. First I created my repository in ECR and then I created a user in IAM getting AWS access key ID and Key.

Now we have to configure AWS in cmd. After configuring run four commands which are given in your ECR repo.

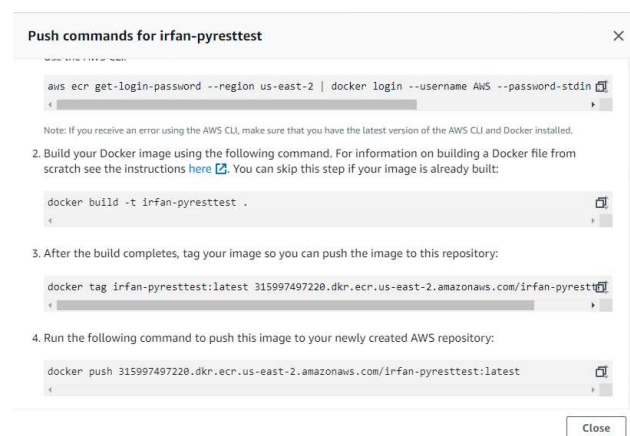


Figure 80: Image push command to ECR

Task 03: Pull images from ECR and deploy using ECS

When image is pushed into ECR repo. Then we have to pull image and deploy the image in CI/CD Pipeline.

```
##### pulling image from ECR repo #####
repo = ecr.Repository.from_repository_name(self, "IrfanRepo", "hello-world")
image = ecs.EcrImage(repo, "latest")

##### deploy image using ECS #####
# Create an ECS cluster
vpc = ec2.Vpc.from_lookup(self, "IrfanVpc", is_default=True) #virtual private cloud#
cluster = ecs.Cluster(self, "IrfanCluster", vpc=vpc)

# Add capacity to it
cluster.add_capacity("IrfanClusterCapacity",
    instance_type=ec2.InstanceType("t2.xlarge"))

task_definition = ecs.Ec2TaskDefinition(self, "TaskDef")
task_definition.add_container("DefaultContainer",
    image=image,
    command=['docker run 315997497220.dkr.ecr.us-east-2.amazonaws.com/hello-world:latest'],
    memory_limit_mib=512
)

# Instantiate an Amazon ECS Service
ecs_service = ecs.Ec2Service(self, "Service",
    cluster=cluster,
    task_definition=task_definition
)
```

Figure 81: Pull Images and deploying using ECS

We first created a virtual private cloud and then ECS cluster using VPC. then add capacity to our cluster. After that we define our task using task definition function and then created EC2Service.

Error and solution

Build image error: When we run “docker run -t image-Name .” it give error that image name should have only lower case letter.

AWS login failed: while pushing build image to ECR. we have to run command to login in AWS through cmd. This issue occurs when we run command.

```
C:\Users\Wuhammad Irfan Hassa>cd hello-world
C:\Users\Wuhammad Irfan Hassa\hello-world>(Get-EC2LoginCommand).Password | docker login --username AWS --password-stdin 315997497228.dkr.ecr.us-east-2.amazonaws.com
Password was unexpected at this time.
```

Figure 82:AWS login failed

To solve this issue, we have to configure AWS. For that we have to create IAM user and then run command AWS configure and give access ID and access key. Then it will login successfully.

ECS task definition error: when we deploy image through ECS. we have to add task definition to our image. I have this issue because in documentation, we have seen example and there was mistake about argument name.

```
--init
275 task_definition.add_container("DefaultContainer",
276 TypeError: add_container() got an unexpected keyword argument 'memory_limit_mi_b'
277 Subprocess exited with error 1
278
279 [Container] 2022/01/18 15:30:16 Command did not exit successfully cdk synth exit status 1
280 [Container] 2022/01/18 15:30:16 Phase complete: BUILD State: FAILED
281 [Container] 2022/01/18 15:30:16 Phase context status code: COMMAND_EXECUTION_ERROR Message: Error while executing command:
cdk synth. Reason: exit status 1
282 [Container] 2022/01/18 15:30:16 Entering phase POST_BUILD
```

Figure 83:ECS task definition error

So we open documentation and then check argument and resolve issue.

References

- [Learn Docker in 12 Minutes 📺 - YouTube](#)
- [Docker Compose in 12 Minutes - YouTube](#)

- [docker basics - YouTube](#)
- [Installing or updating the latest version of the AWS CLI - AWS Command Line Interface](#)
- <https://blog.tinystacks.com/ecs-serverless-or-not-fargate-vs-ec2-clusters>
- [Amazon ECS Construct Library — AWS Cloud Development Kit 1.140.0 documentation](#)
- <https://blog.clairvoyantsoft.com/deploy-and-run-docker-images-on-aws-ecs-85a17a073281>