Proxima Centauri

Documentation

Written by:

Muhammad Irfan Hassan

Trainee @SkipQ2021

Table of Contents

Sprint1

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.

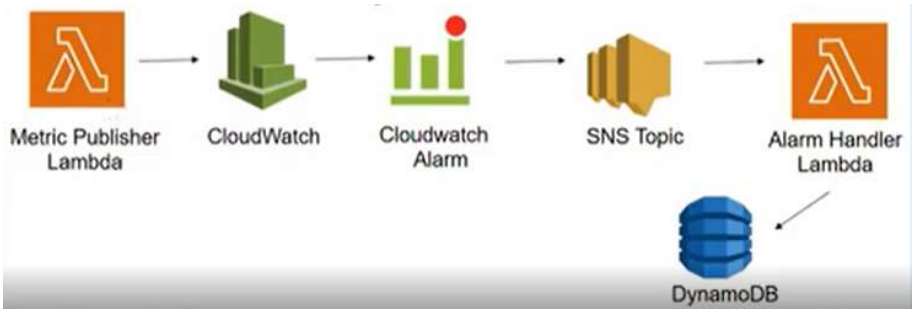


Figure 1: Web Health Monitor CDK Application

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.

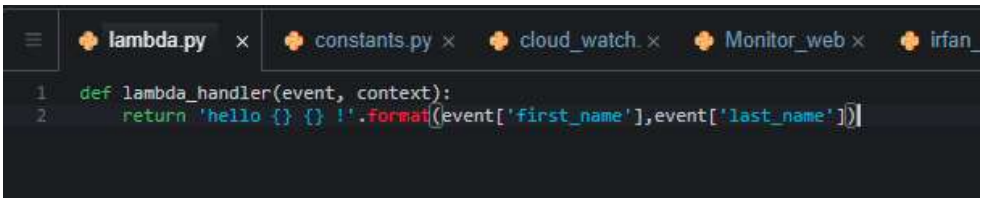


Figure 2: Hello Lambda function

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

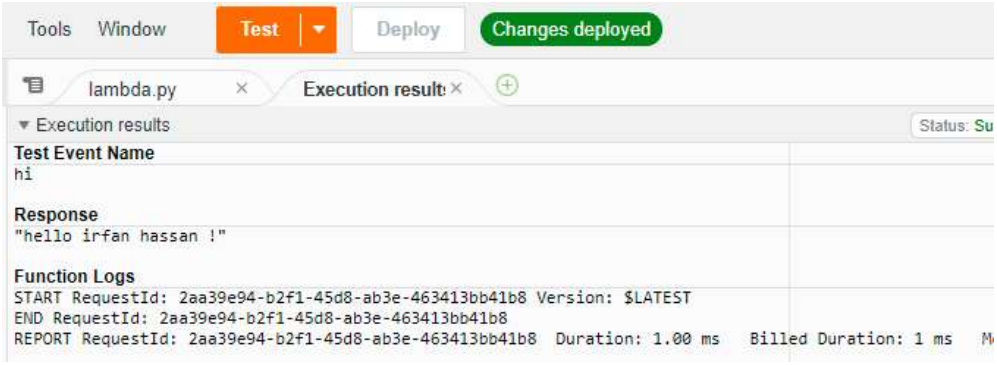


Figure 3:Testing hellolambda

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.

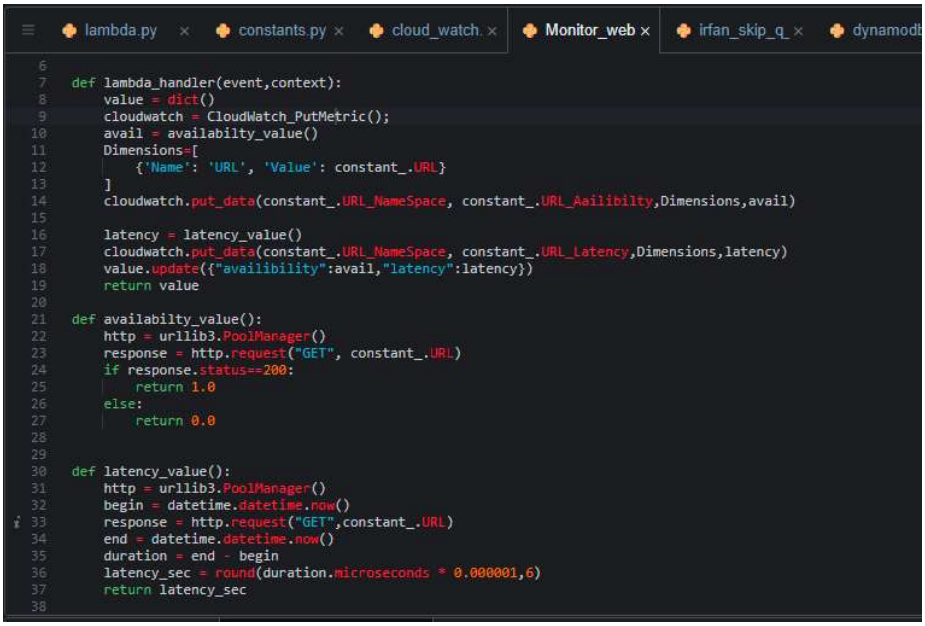


Figure 4: Periodic lambda for availability and latency of webpage

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

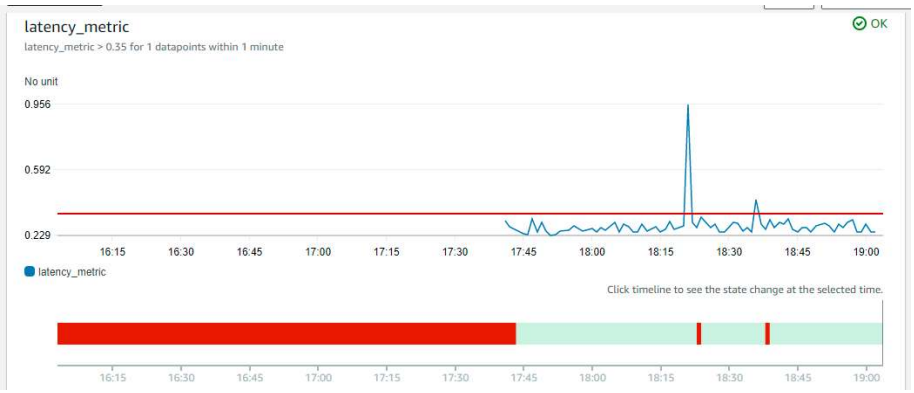


Figure 8: Alarm triggered when threshold breached

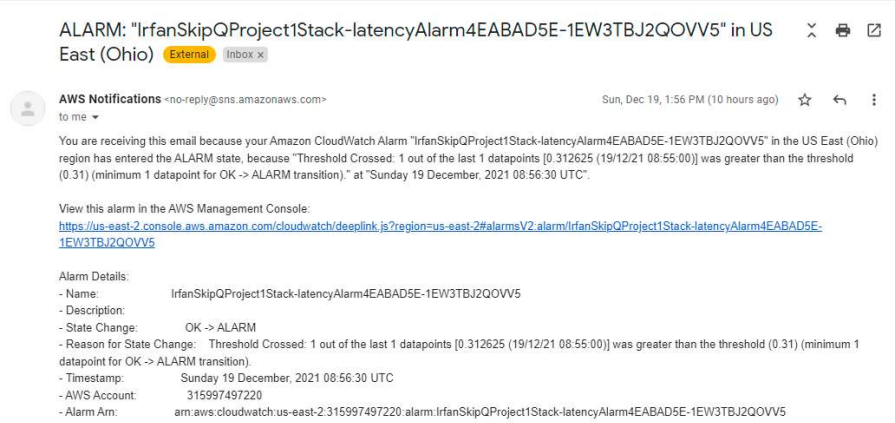


Figure 9: Email notification to the subscriber

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

```python
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.

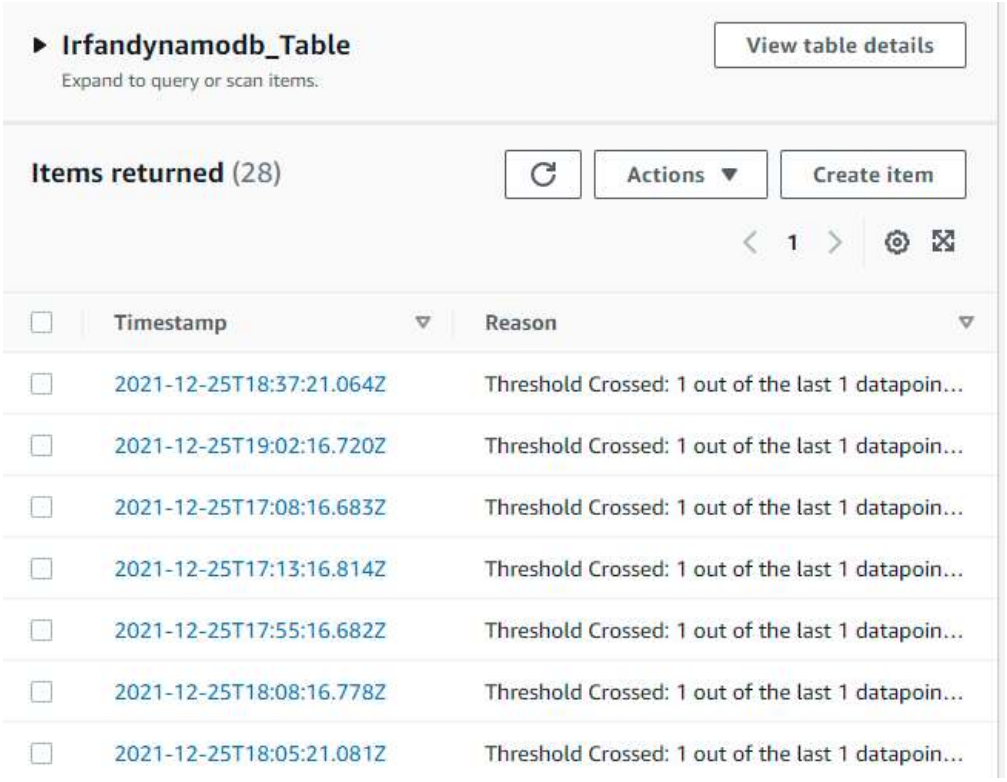


Figure 13: Alarm details in Dynamo DB Table

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.

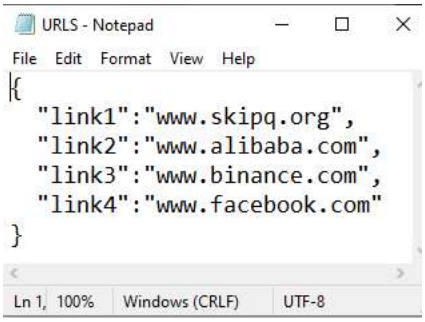


Figure 14: URLS in json file

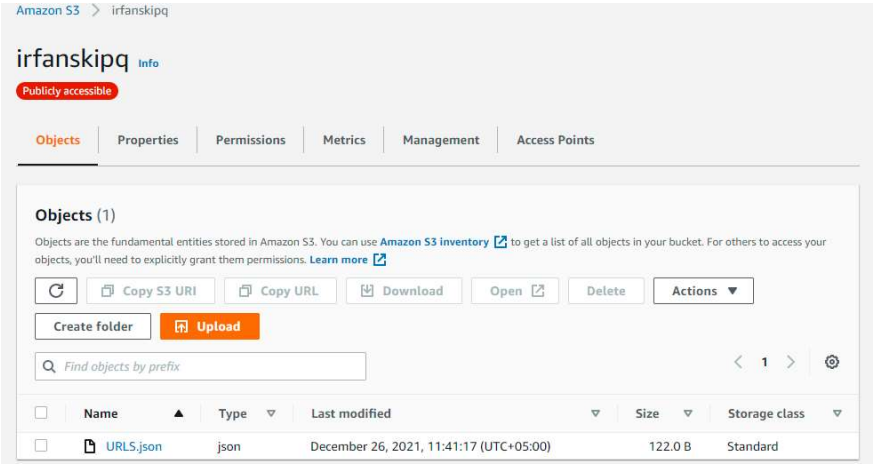


Figure 15:Json file in S3 Bucket

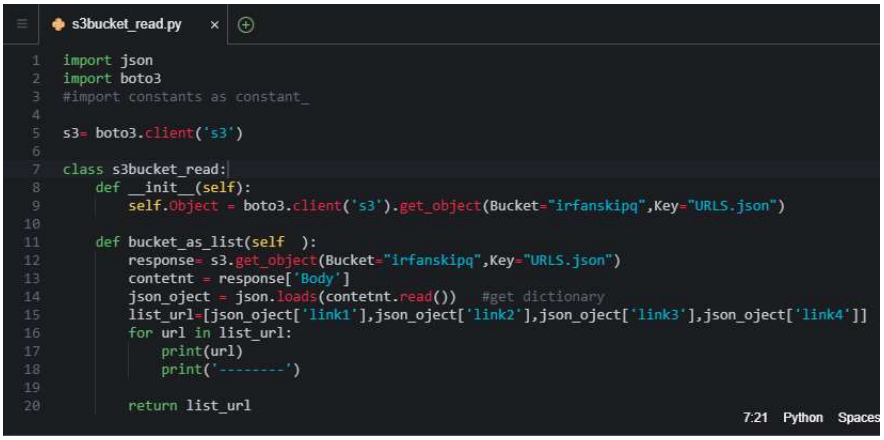


Figure 16: S3 bucket reading

Task 05: Implement Project for customize URLs List

Our aim for this project was to implement Health Monitoring for a list Customize webpages (json file 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 of 4 webpages. Here 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,
```

122:17 Py

Figure 17: Adding for loop in Main stack

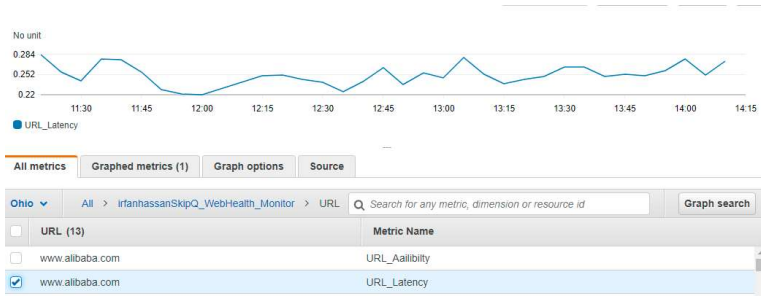
Monitor_webhealth.py ×

```
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.



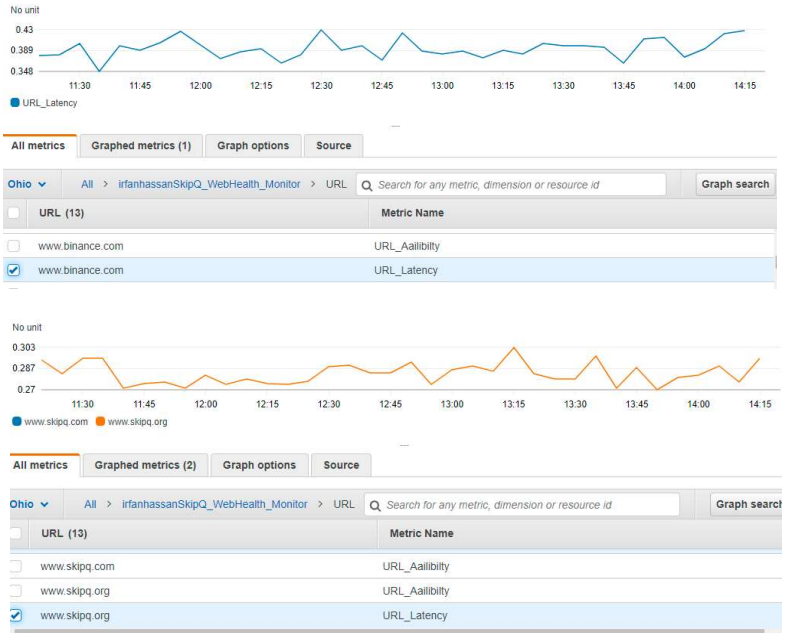


Figure 19: Latency of 3 URLs

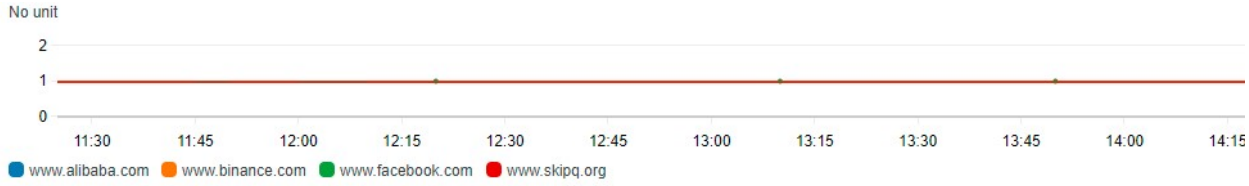


Figure 20: Latency for 4 URL

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

Project Description

Creating multi-stage pipeline having Beta/Gamma and Prod stage using CDK. the source from my GitHub repository for CI/CD Pipeline. Pipeline 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 and then build the code after installing all requirement. In the Beta stage, it runs the unit test and integration test. After passing through the unit test, it deploys the source code. In Production stage, it asks for manual approval then deploys the source code to AWS server.

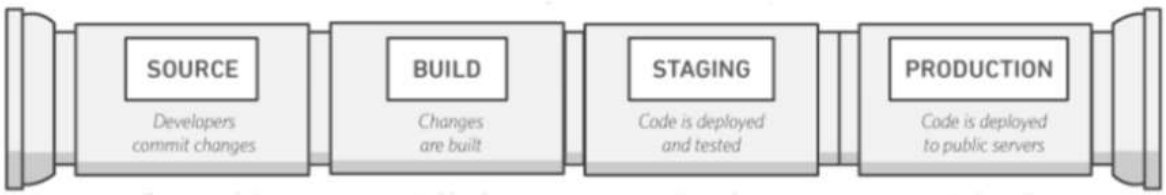


Figure 21: CI/CD Pipeline

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 or with your own custom plugin.

Secret Manager

AWS Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources. It allows to user to save key and then access key when he 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 to setup 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.

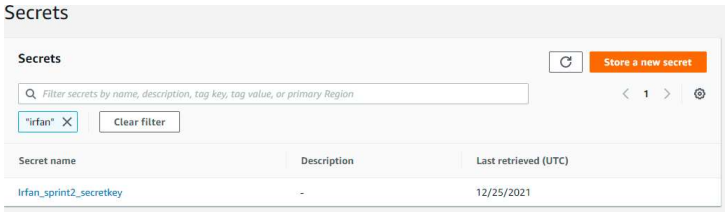


Figure 22: Secret manager

- Create pipeline stack and pipeline_satges files in project_stack folder.

Figure 23: Files in stack folder

- Remove cdk.out from. gitignore file.

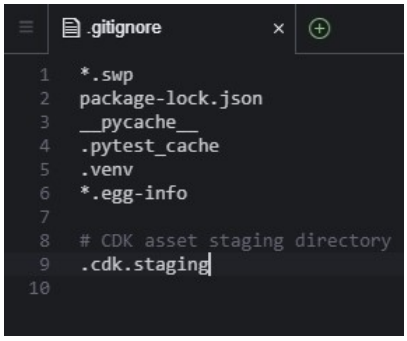


Figure 24: .gitignore file

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

Figure 25: cdk.json file

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

Figure 26: APP.py

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

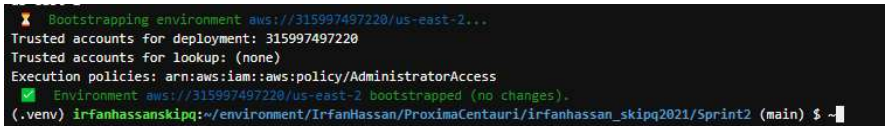


Figure 27: Bootstrap don

Project milestones

This project is divided into subtasks. Each Subtask is discussing with details.

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

First we have to create source for pipeline. GitHub will be 3rd party we will be integrating to our pipeline as source. After adding source build the source. Code for Source is shown below. Add repository in repo string, set branch and then add name of secret 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

```
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)
```

Figure 29: Building the Source

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

If it run successfully then go in Code Pipeline and check your pipeline. You will get these result if pipeline is run successfully.

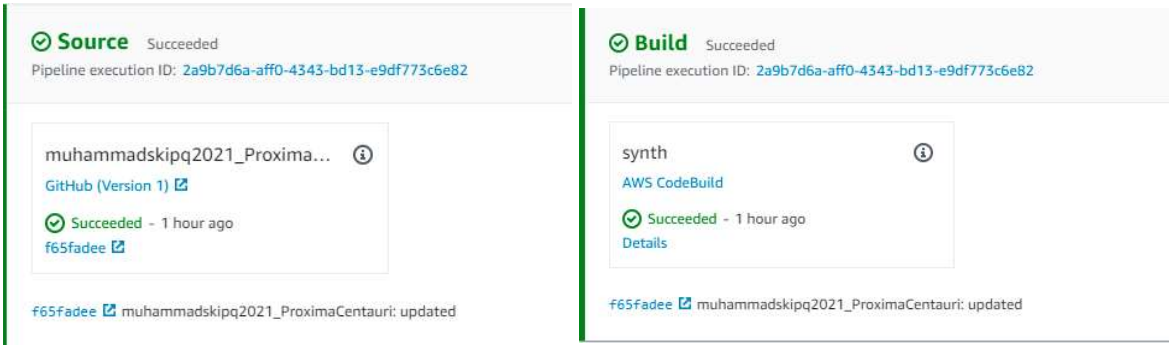


Figure 30: Output (Source and Build)

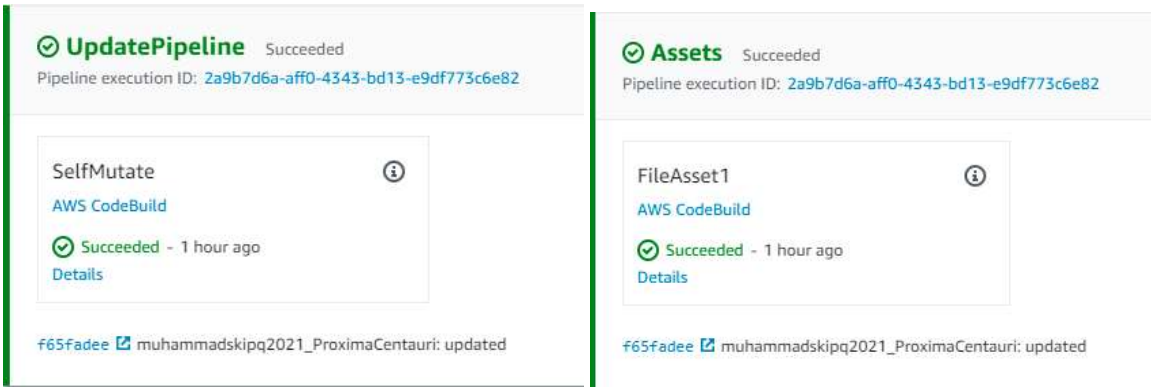


Figure 31: Pipeline updated and Assert are successful

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 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 adding to pipeline

irfan_stage.py

```
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. You will get these result on Code Pipeline.

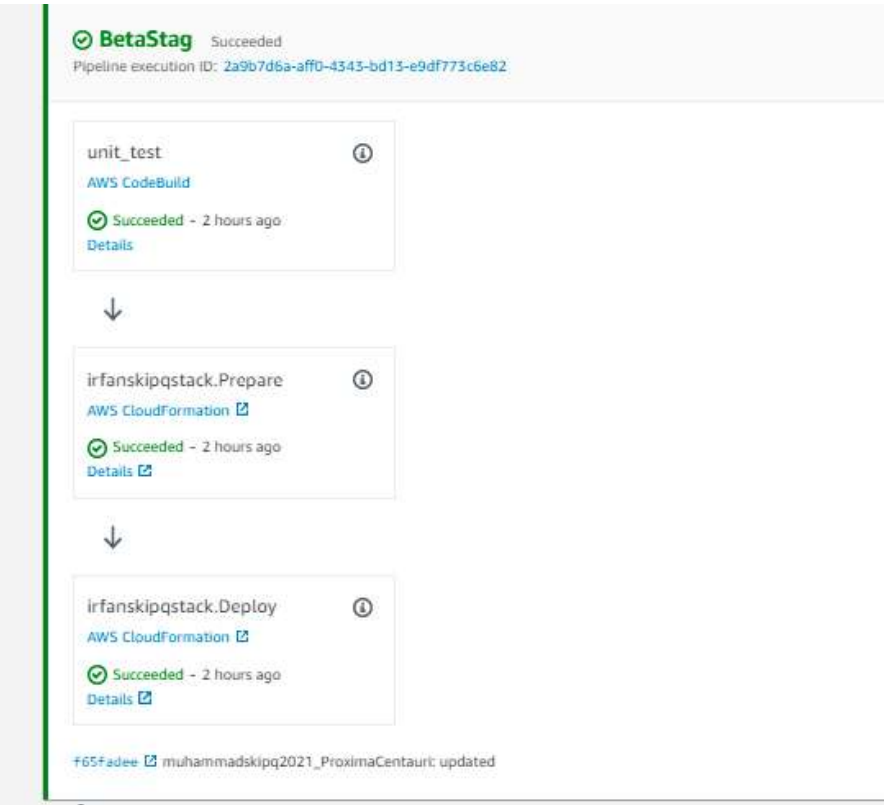


Figure 34: Beta stage is added successfully

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.

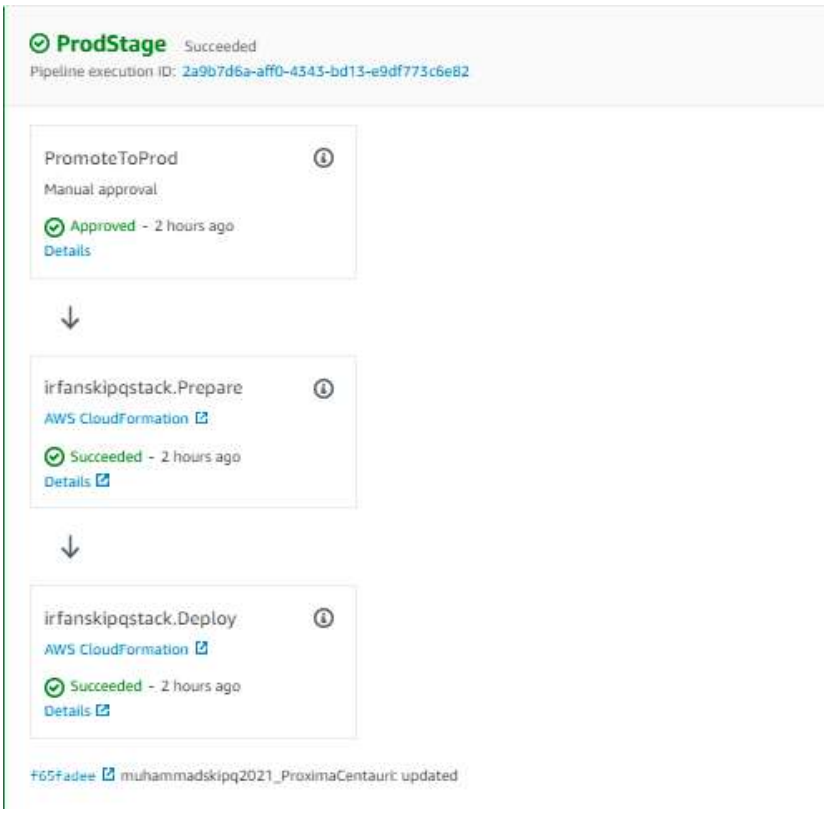


Figure 36: Adding Production stage

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.

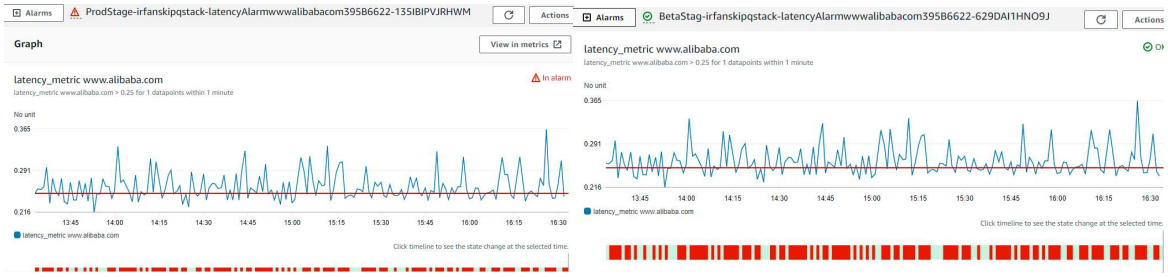


Figure 37: Plot of Latency for alibab.com for Beta and Prod stages

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.

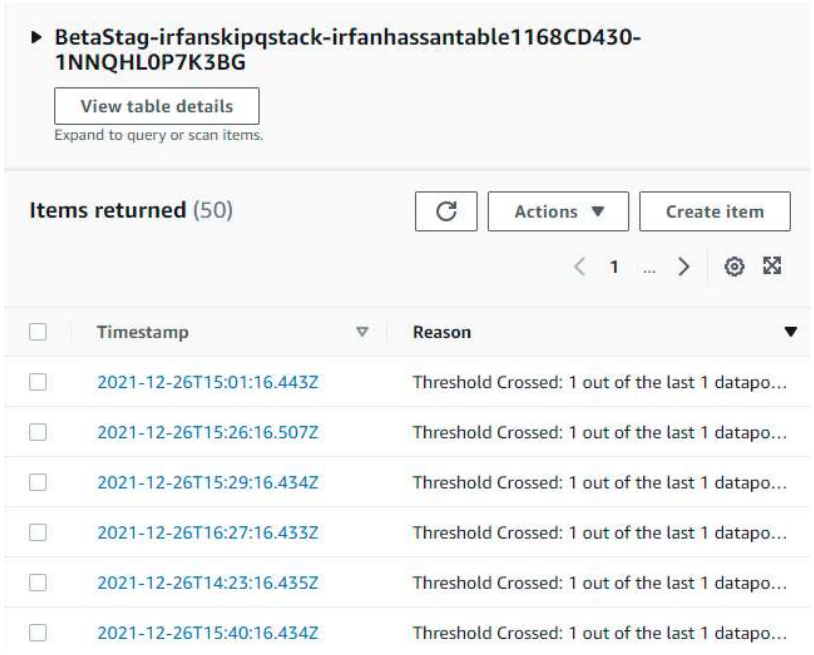


Figure 39: Beta stage Table

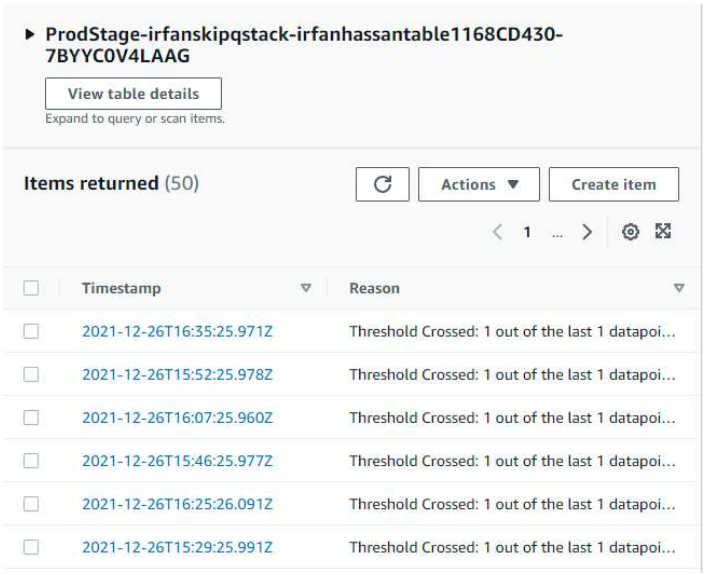


Figure 40: Prod Stage Table

Email Notification are also received when alarm trigger.

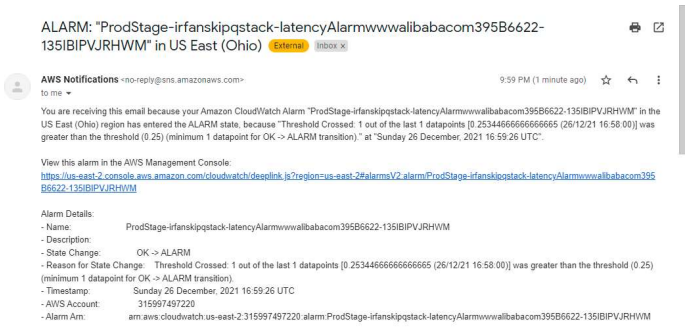


Figure 41: Alarm Notification at Email

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.

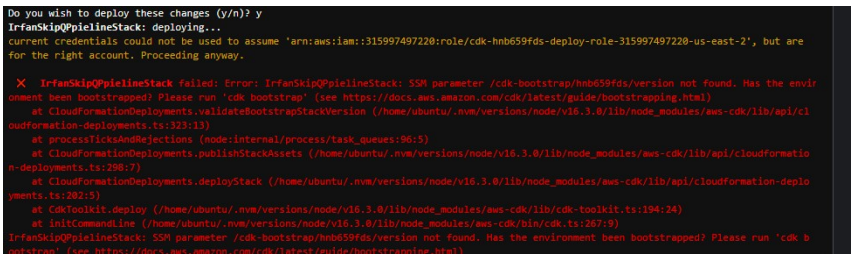


Figure 42: while deploying: Error

- 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.

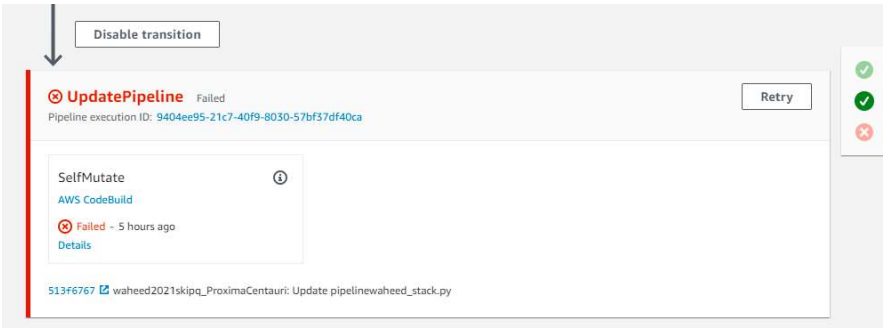


Figure 44:Update Failed: Error

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

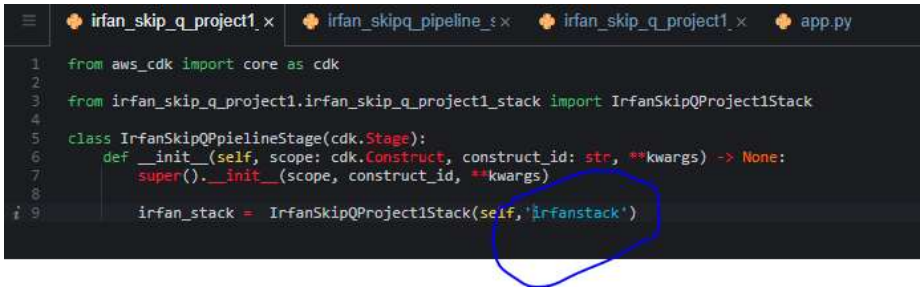


Figure 45: Adding stage: Error

- While running pytest unit test not able to import aws_core. install requirements using "npm install -g aws-cdk " and "pip install -r 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.

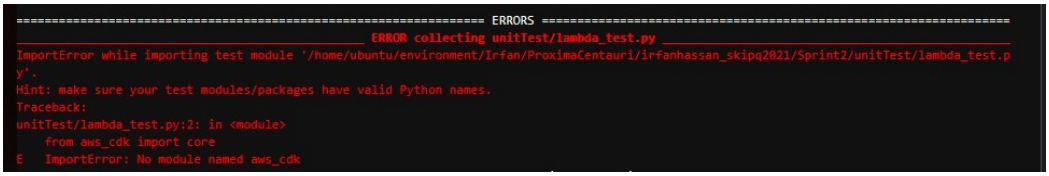


Figure 46: Pytest Unit test : Error

- 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&DAF0=1&PC=U531