SkipQ – Pegasus (Cohort-6)

Documentation

Documented by: Muhammad Faizan Ikram

Table of Contents

Project Document

1 TRAINING OVERVIEW

2 SPRINT 4

2.1 OBJECTIVE

The objective of this Sprint4 is to build up on Sprint3 and build a public server-less CRUD API Gateway endpoint for the web crawler to create/read/update/delete the target URLs list containing the list of websites/webpages to crawl. The core objectives are listed as follows:

- Create and populate a URL table in DynamoDB.
- Implement CRUD REST commands on DynamoDB entries.
- Create Lambda handler to process API Gateway requests.
- Create public server-less REST API Gateway endpoint.
- Use CI/CD to automate multiple deployment stages (prod vs beta).
- Extend tests in each stage to cover the CRUD operations and DynamoDB read/write time.
- Manage README files and run-books in markdown on GitHub.

2.2 TECHNOLOGIES USED

2.2.1 AWS CodeBuild

AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don’t need to provision, manage, and scale your own build servers. CodeBuild scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue. You can get started quickly by using prepackaged build environments, or you can create custom build environments that use your own build tools. With CodeBuild, you are charged by the minute for the compute resources you use.

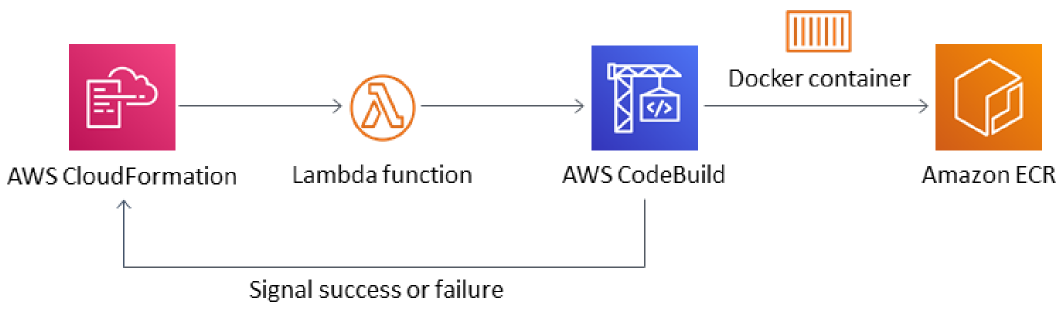


Figure 1: CodeBuild workflow [1]

2.2.2 AWS CodePipeline

AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. CodePipeline 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. This enables you to rapidly and reliably deliver features and updates. You can easily integrate AWS CodePipeline with third-party services such as GitHub or with your own custom plugin. With AWS CodePipeline, you only pay for what you use. There are no upfront fees or long-term commitments.

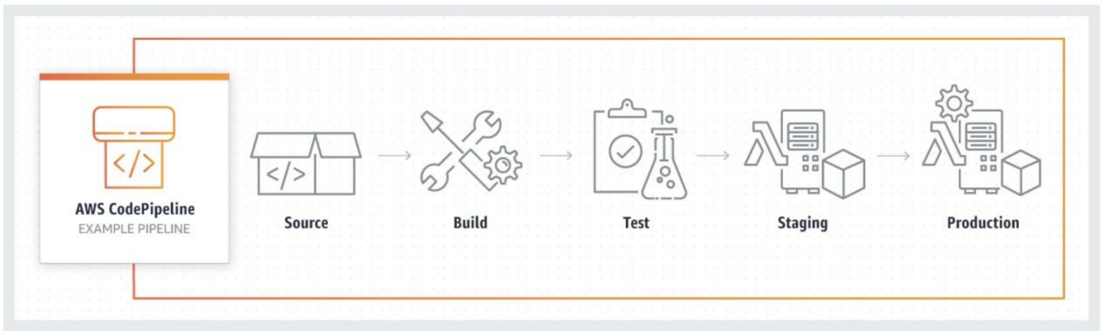


Figure 2: CodePipeline Process [2]

2.2.3 AWS CloudWatch

Amazon CloudWatch is a monitoring and observability service built for DevOps engineers, developers, site reliability engineers (SREs), IT managers, and product owners. CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, and optimize resource utilization. CloudWatch collects monitoring and operational data in the form of logs, metrics, and events. You get a unified view of operational health and gain complete visibility of your AWS resources, applications, and services running on AWS and on-premises. You can use CloudWatch to detect anomalous behavior in your environments, set alarms, visualize logs and metrics side by side, take automated actions, troubleshoot issues, and discover insights to keep your applications running smoothly.

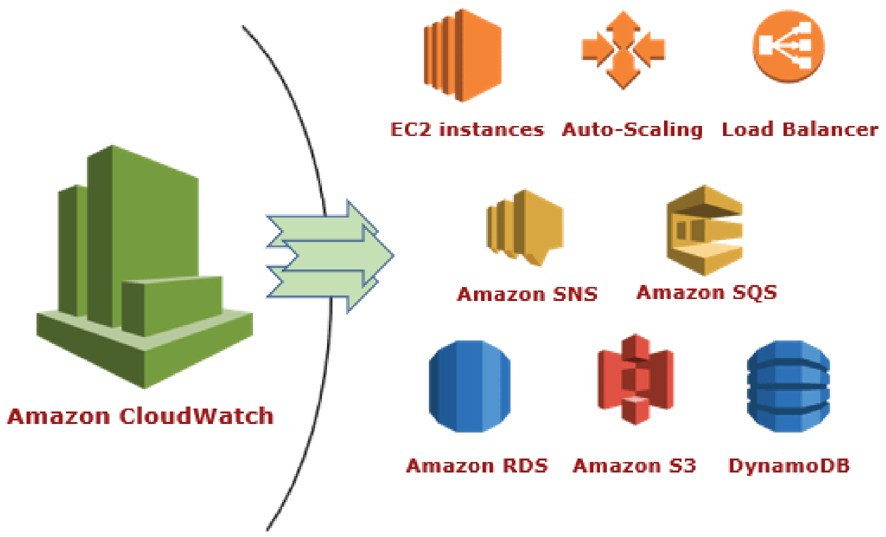


Figure 3: Resource monitored by CloudWatch [3]

2.2.4 AWS DynamoDB

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

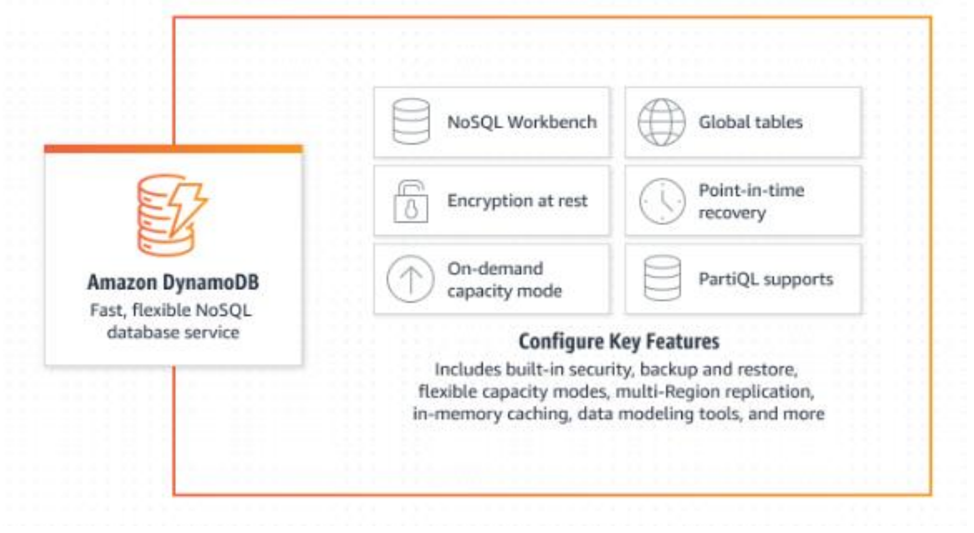

Amazon DynamoDB
Fast, flexible NoSQL database service
NoSQL Workbench
Global tables
Encryption at rest
Point-in-time recovery
On-demand capacity mode
PartiQL supports
Configure Key Features
Includes built-in security, backup and restore, flexible capacity modes, multi-Region replication, in-memory caching, data modeling tools, and more

Figure 4: AWS DynamoDB [4]

2.2.5 AWS API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

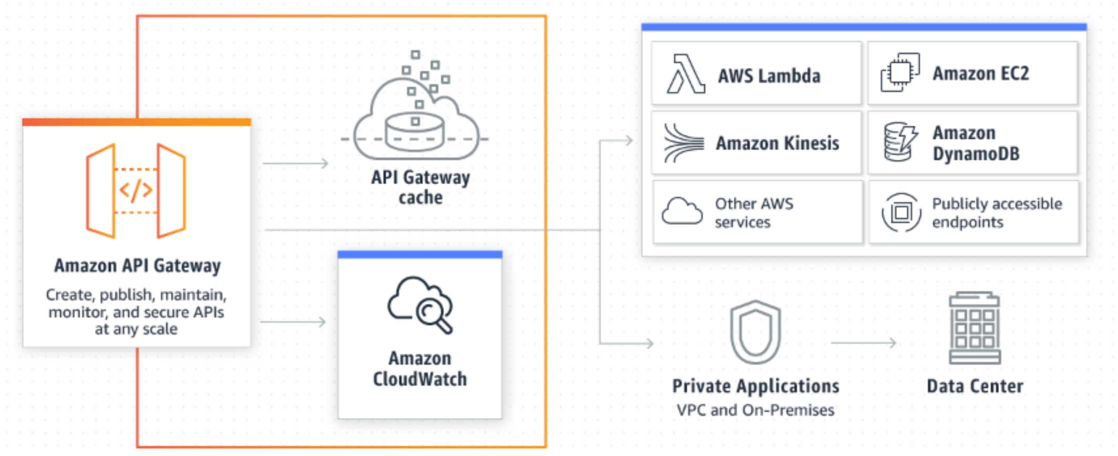


Figure 5AWS API Gateway [5]

.

2.2.6 Github

GitHub is a web-based interface that uses Git, the open source version control software that lets multiple people make separate changes to web pages at the same time. It allows for real-time collaboration, encourage teams to work together to build and edit their code and site content. GitHub allows multiple developers to work on a single project at the same time, reduces the risk of duplicative or conflicting work, and can help decrease production time. With GitHub, developers can build code, track changes, and innovate solutions to problems that might arise during the site development process simultaneously. Non-developers can also use it to create, edit, and update projects.

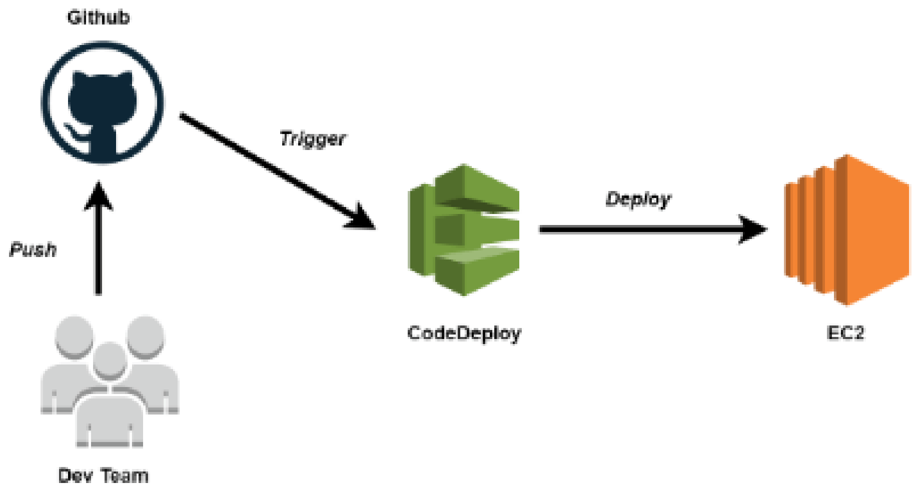
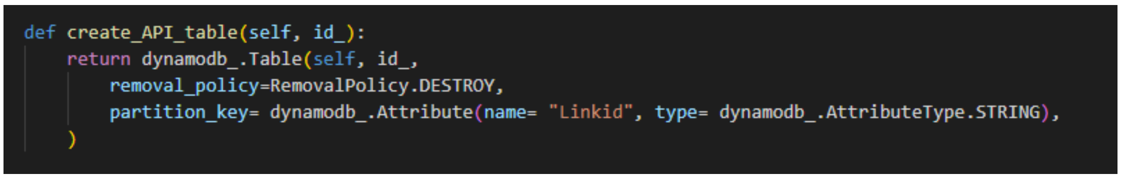


Figure 6: GitHub Integration with EC2 [6]

2.3 Sprint Breakdown

2.3.1 Task – 1: Create and populate a URL table in DyamoDB

In this task, we had to create a DynamoDB table to populate some urls from the list that we were already using in the constant file. We created the table using a separate function since we needed to pass it separate partition key and table name. We populated the url from our constant file in the lambda_handler so whenever the API invokes the lambda, the url list will be added initially.



```
def create_API_table(self, id_):
    return dynamodb_.Table(self, id_,
        removal_policy=RemovalPolicy.DESTROY,
        partition_key= dynamodb_.Attribute(name= "Linkid", type= dynamodb_.AttributeType.STRING),
    )
```

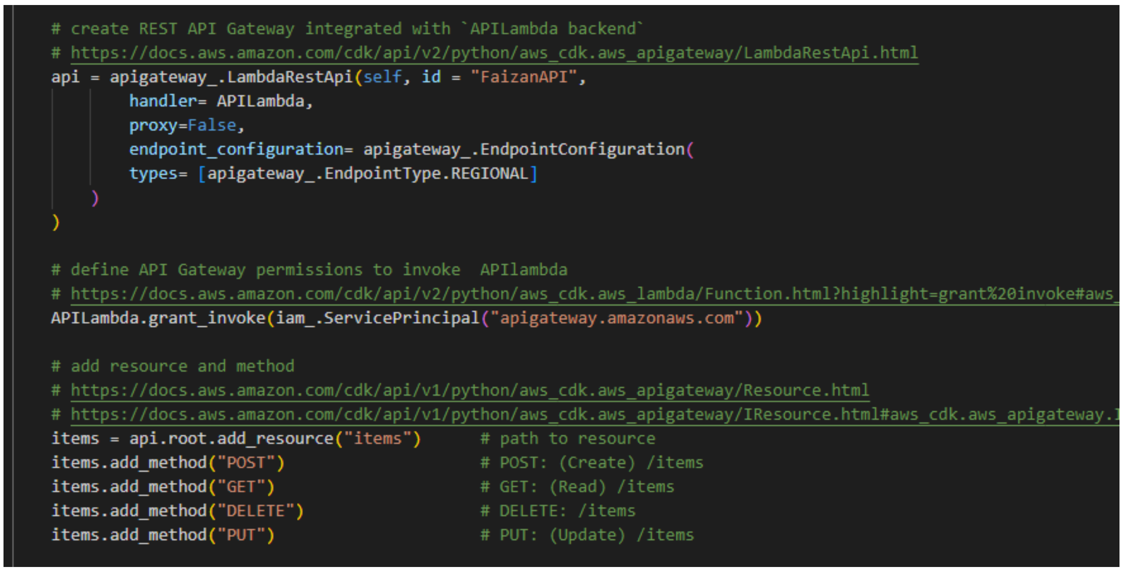
2.3.2 Task – 2: Create Lambda handler to process API Gateway requests

In this task, we created the lambda_handler which will be used to process the API Gateway request every time a client request hits the lambda function. We have populated the initial URLs using this lambda handler.

2.3.3 Task – 3: Create public server-less REST API Gateway endpoint.

In this task, we need to create and a public server-less API Gateway using REST methods and implement the GET/PUT/DELETE/POST methods (also known as CRUD operations) using this API for the lambda function.

Code:



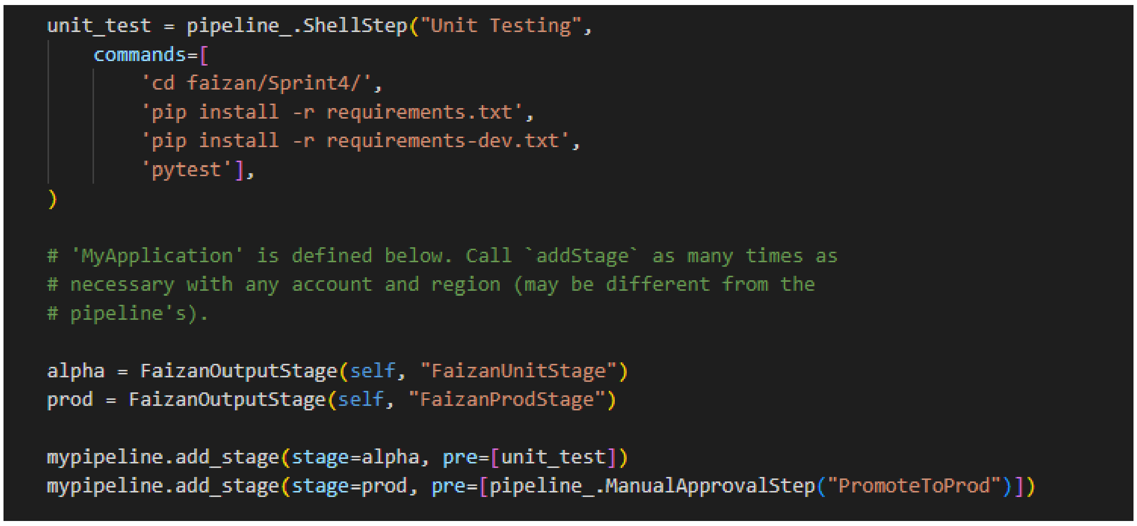
```
# create REST API Gateway integrated with `APILambda backend`
# https://docs.aws.amazon.com/cdk/api/v2/python/aws_cdk.aws_apigateway/LambdaRestApi.html
api = apigateway_.LambdaRestApi(self, id = "FaizanAPI",
        handler= APILambda,
        proxy=False,
        endpoint_configuration= apigateway_.EndpointConfiguration(
        types= [apigateway_.EndpointType.REGIONAL]
    )
)

# define API Gateway permissions to invoke  APIlambda
# https://docs.aws.amazon.com/cdk/api/v2/python/aws_cdk.aws_lambda/Function.html?highlight=grant%20invoke#aws_
APILambda.grant_invoke(iam_.ServicePrincipal("apigateway.amazonaws.com"))

# add resource and method
# https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_apigateway/Resource.html
# https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_apigateway/IResource.html#aws_cdk.aws_apigateway.I
items = api.root.add_resource("items")       # path to resource
items.add_method("POST")                     # POST: (Create) /items
items.add_method("GET")                      # GET: (Read) /items
items.add_method("DELETE")                   # DELETE: /items
items.add_method("PUT")                      # PUT: (Update) /items
```

Figure 7: Code snippet for stages

2.3.4 Task – 4: Use CI/CD to automate multiple deployment stages (prod vs beta)

In this task, we need to create and add different stages for the pipeline to make it a multi-stage pipeline architecture. The "Alpha" stage was added for the unit testing and "Prod" stage was added for the manual approval of the application to go into deployment.

Code:

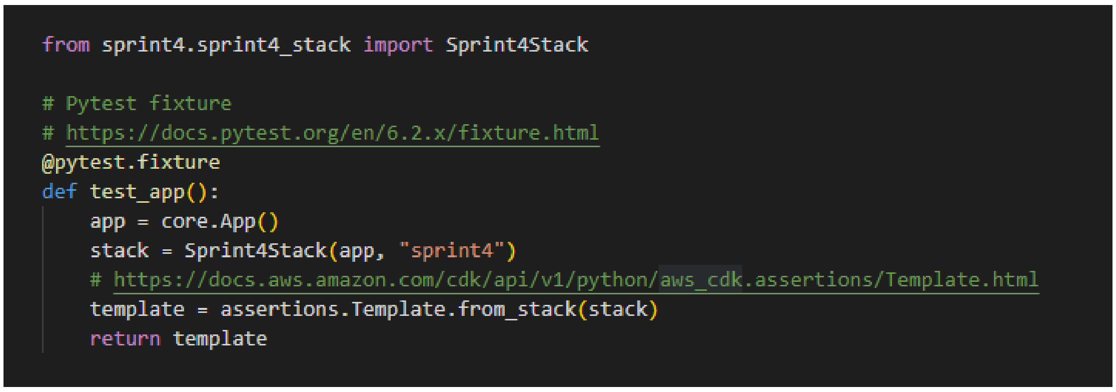


```
unit_test = pipeline_.ShellStep("Unit Testing",
    commands=[
        'cd faizan/Sprint4/',
        'pip install -r requirements.txt',
        'pip install -r requirements-dev.txt',
        'pytest'],
)

# 'MyApplication' is defined below. Call `addStage` as many times as
# necessary with any account and region (may be different from the
# pipeline's).

alpha = FaizanOutputStage(self, "FaizanUnitStage")
prod = FaizanOutputStage(self, "FaizanProdStage")

mypipeline.add_stage(stage=alpha, pre=[unit_test])
mypipeline.add_stage(stage=prod, pre=[pipeline_.ManualApprovalStep("PromoteToProd")])
```

2.3.5 Task – 5: Extend tests in each stage to cover the CRUD operations and DynamoDB read/write time.

In this task, we created 6-7 unit cases for the application using assertions module of AWS which include the testing of resources count, parameters presence, conditions check, resource properties etc. In these test, we checked the resource count and resource properties that were defined in the Cloud Formation template and verified that the tests are clear and application is ready.

Code:



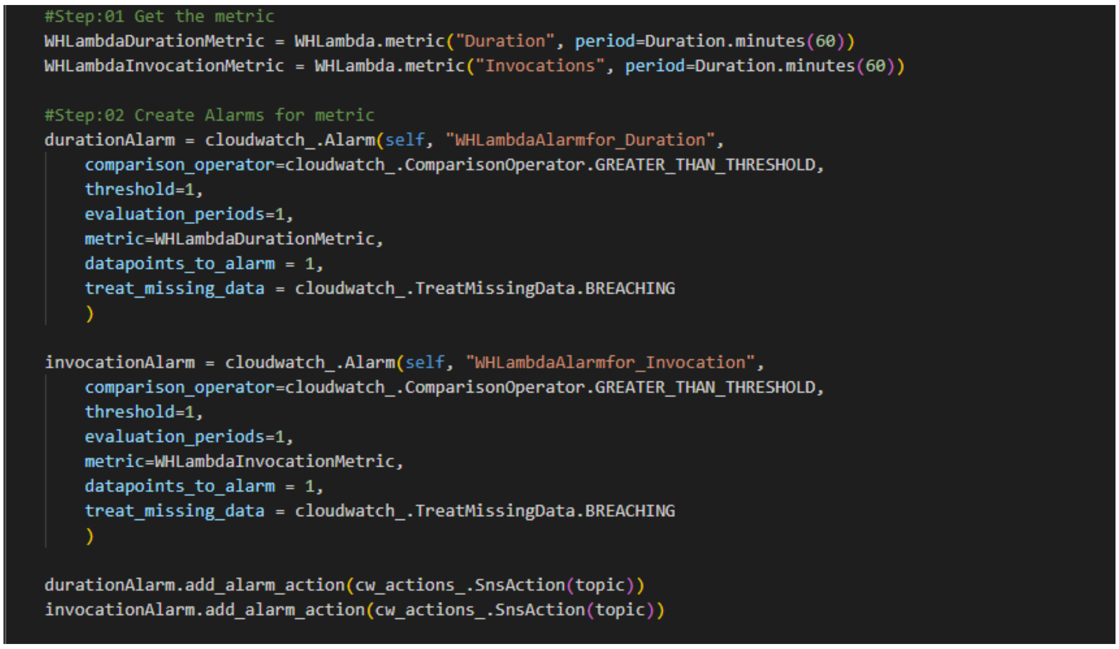
```python
from sprint4.sprint4_stack import Sprint4Stack

# Pytest fixture
# https://docs.pytest.org/en/6.2.x/fixture.html
@pytest.fixture
def test_app():
    app = core.App()
    stack = Sprint4Stack(app, "sprint4")
    # https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.assertions/Template.html
    template = assertions.Template.from_stack(stack)
    return template
```

Figure 8: Sample code snippet of a unit test

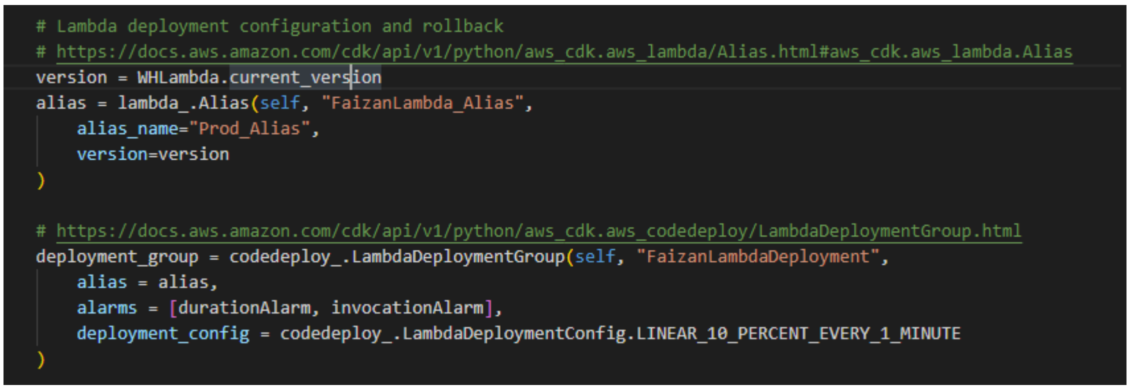
2.3.6 Task – 5: Create metric and alarm and configure Lambda deployment and rollback

In this task, we created metrics and alarms for the "Duration" and "Invocation" of the Lambda for our web health monitoring application. We also created deployment group for the Lambda deployment configuration and rollback so that the application can be deployed in Blue-Green technique and also rollback to previous version if there is any alarm generated.

Code:



```python
#Step:01 Get the metric
WHLambdaDurationMetric = WHLambda.metric("Duration", period=Duration.minutes(60))
WHLambdaInvocationMetric = WHLambda.metric("Invocations", period=Duration.minutes(60))

#Step:02 Create Alarms for metric
durationAlarm = cloudwatch_.Alarm(self, "WHLambdaAlarmfor_Duration",
    comparison_operator=cloudwatch_.ComparisonOperator.GREATER_THAN_THRESHOLD,
    threshold=1,
    evaluation_periods=1,
    metric=WHLambdaDurationMetric,
    datapoints_to_alarm = 1,
    treat_missing_data = cloudwatch_.TreatMissingData.BREACHING
    )

invocationAlarm = cloudwatch_.Alarm(self, "WHLambdaAlarmfor_Invocation",
    comparison_operator=cloudwatch_.ComparisonOperator.GREATER_THAN_THRESHOLD,
    threshold=1,
    evaluation_periods=1,
    metric=WHLambdaInvocationMetric,
    datapoints_to_alarm = 1,
    treat_missing_data = cloudwatch_.TreatMissingData.BREACHING
    )

durationAlarm.add_alarm_action(cw_actions_.SnsAction(topic))
invocationAlarm.add_alarm_action(cw_actions_.SnsAction(topic))
```



```
# Lambda deployment configuration and rollback
# https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_lambda/Alias.html#aws_cdk.aws_lambda.Alias
version = WHLambda.current_version
alias = lambda_.Alias(self, "FaizanLambda_Alias",
    alias_name="Prod_Alias",
    version=version
)

# https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_codedeploy/LambdaDeploymentGroup.html
deployment_group = codedeploy_.LambdaDeploymentGroup(self, "FaizanLambdaDeployment",
    alias = alias,
    alarms = [durationAlarm, invocationAlarm],
    deployment_config = codedeploy_.LambdaDeploymentConfig.LINEAR_10_PERCENT_EVERY_1_MINUTE
)
```

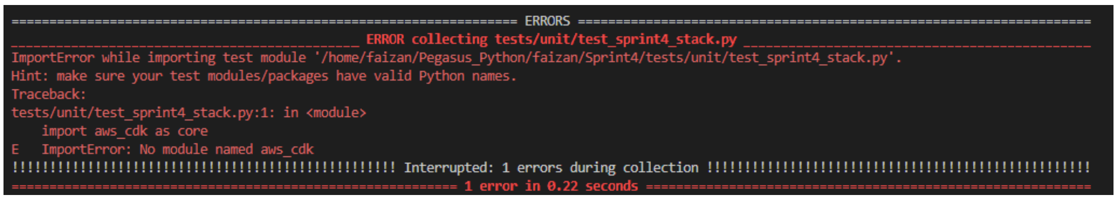
Figure 9: Code snippet for metric, alarm and lambda configuration

2.4 Issues/Errors and Troubleshooting

2.4.1 ImportError: No module named aws_cdk

- **Description**:

  I was getting this error that no module found named aws_cdk although I have imported it with libraries.



```
==================================================================== ERRORS ====================================================================
______________________________________________ ERROR collecting tests/unit/test_sprint4_stack.py ______________________________________________
ImportError while importing test module '/home/faizan/Pegasus_Python/faizan/Sprint4/tests/unit/test_sprint4_stack.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
tests/unit/test_sprint4_stack.py:1: in <module>
    import aws_cdk as core
E   ImportError: No module named aws_cdk
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 1 errors during collection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
========================================================== 1 error in 0.22 seconds ==========================================================
```

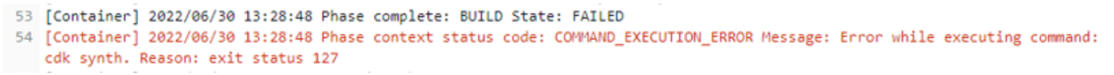
- **Solution**:

  1. Deactivate virtual environment using command ***deactivate***
  2. Uninstall pytest using ***sudo apt purge python-pytest*** or ***pip3 uninstall pytest***
  3. Activate the venv using command ***pip install -r requirements.txt***
  4. Install pytest using ***pip install pytest***
  5. Run pytest

2.4.2 BUILD State: FAILED

- **Description**:

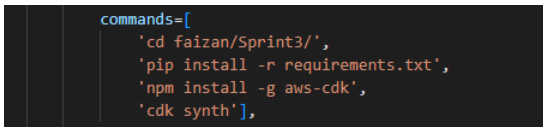
  I was getting this error while deploying my pipeline on AWS. The build stage in pipeline got failed while executing the *cdk synth* command.



```
[Container] 2022/06/30 13:28:48 Phase complete: BUILD State: FAILED
[Container] 2022/06/30 13:28:48 Phase context status code: COMMAND_EXECUTION_ERROR Message: Error while executing command:
cdk synth. Reason: exit status 127
```

- **Solution**:

  1. For correct execution of *cdk synth*, add the aws-cdk command and the commands in pipeline should be given in following order:



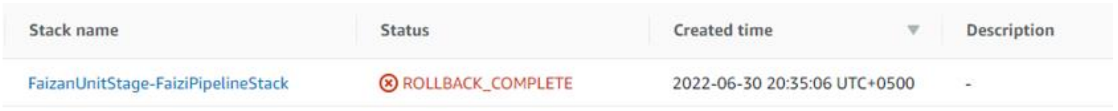
```
commands=[
    'cd faizan/Sprint3/',
    'pip install -r requirements.txt',
    'npm install -g aws-cdk',
    'cdk synth'],
```

  2. Make sure to push your code on GitHub for pipeline.

2.4.3 ROLLBACK_COMLETE: FAILED

- **Description**:

  I was getting this error while deploying my pipeline on AWS. The test deploy stage in pipeline got failed because of stack creation and alarm generation it goes on ROLL BACK and stack doesn't update itself.



Stack name	Status	Created time	Description
FaizanUnitStage-FaiziPipelineStack	ROLLBACK_COMPLETE	2022-06-30 20:35:06 UTC+0500	-

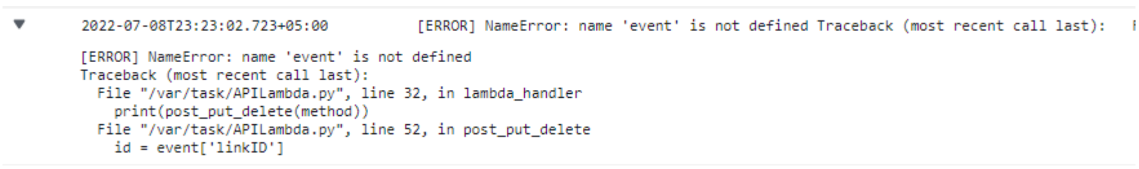
- **Solution**:

  1. Delete your old stack
  2. Destroy stack from CLI
  3. Synth and Deploy again

2.4.4 NameError: name 'event'

Description:

I was getting this error while using Rest API. The API does not receive any event have we have to define event manually and so the method should also be defined while giving inputs to API.



```
2022-07-08T23:23:02.723+05:00        [ERROR] NameError: name 'event' is not defined Traceback (most recent call last):

[ERROR] NameError: name 'event' is not defined
Traceback (most recent call last):
  File "/var/task/APILambda.py", line 32, in lambda_handler
    print(post_put_delete(method))
  File "/var/task/APILambda.py", line 52, in post_put_delete
    id = event['linkID']
```

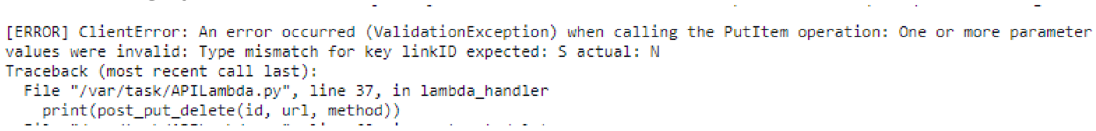
- **Solution**:

  1. I used LambdaRestApi that gives valid events.
  2. You can also use event if you define your own event.

2.4.5 ValidationException: PutItem operation

Description:

I was getting this error because I gave integer as value in partition key but the partition key format is string by default



```
[ERROR] ClientError: An error occurred (ValidationException) when calling the PutItem operation: One or more parameter
values were invalid: Type mismatch for key linkID expected: S actual: N
Traceback (most recent call last):
  File "/var/task/APILambda.py", line 37, in lambda_handler
    print(post_put_delete(id, url, method))
```

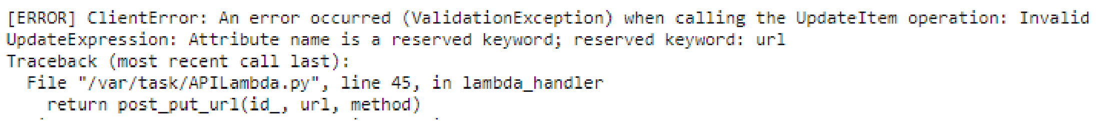
- **Solution**:

  1. Use string format for your partition key.

2.4.6 ValidationException: UpdateItem operation – Reserve Keyword Error

Description:

I was getting this error because I used a reserve keyword of Boto3 DynamoDB service in my partition key and other item keys.



```
[ERROR] ClientError: An error occurred (ValidationException) when calling the UpdateItem operation: Invalid
UpdateExpression: Attribute name is a reserved keyword; reserved keyword: url
Traceback (most recent call last):
  File "/var/task/APILambda.py", line 45, in lambda_handler
    return post_put_url(id_, url, method)
```

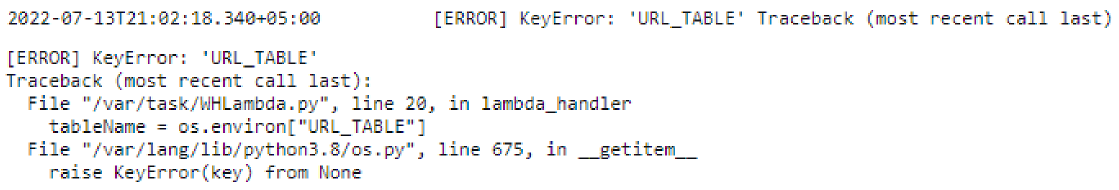
- **Solution**:

  1. Use ***UpdateExpression*** with ***ExpressionAttributeNames*** in UpdateItem and define your key as # Attribute name.

2.4.7 KeyError: URL_TABLE

Description:

I was getting this error because I forgot to pass the environment variable to Lambda file.



```
2022-07-13T21:02:18.340+05:00         [ERROR] KeyError: 'URL_TABLE' Traceback (most recent call last)

[ERROR] KeyError: 'URL_TABLE'
Traceback (most recent call last):
  File "/var/task/WHLambda.py", line 20, in lambda_handler
    tableName = os.environ["URL_TABLE"]
  File "/var/lang/lib/python3.8/os.py", line 675, in __getitem__
    raise KeyError(key) from None
```

- **Solution**:

  1. Make sure to add environment variable to the lambda handler.

2.5 FUNCTION AND CLASSES DOCUMENTATION

2.5.1 Create_lambda:

createLambda(self, id, asset, handler)	
Description	Creates the lambda function
Parameters	id (str): id for the lambda function asset (str): asset folder where the py file is located handler (str): WHLambda.lambda_handler role: myRole
Returns	lambda function

2.5.2 Lambda_handler: (for availability and latency)

lambda_handler(event, context)	
Description	Creates the lambda function to get availability and latency
Parameters	event: it gets the parameters from lambda function context: context returns the information of availability and latency values in dictionary format.
Returns	Availability, latency values in dictionary format

2.5.3 Lambda_handler: (for DynamoDB table)

lambda_handler(event, context)	
Description	Creates the lambda function to put data in dynamoDB table
Parameters	event: it gets the data in the form of JSON format from the table. context: context returns the information records to of each invocation and prints them in log
Returns	Availability, latency values in dictionary format

2.5.4 Lambda_handler: (for URL table)

lambda_handler(event, context)	
Description	Creates the lambda function to perform CRUD operations
Parameters	event: it gets the data in the form of JSON format from the table. context: context returns the information records to of each invocation and prints them in log
Returns	None

2.5.5 Create_table:

create_table(self, id_)	
Description	Creates the table in the Dynamo DB
Parameters	Id = string name
Returns	dynamo_db table
create_API_table(self, id_)	
Description	Creates the table in the Dynamo DB
Parameters	Id = string name
Returns	dynamo_db table

2.5.6 Create_role:

lambda_handler(self)	
Description	Creating role for policies
Parameters	None
Returns	Iam.Role

2.5.7 getAvailability:

getAvailability(url):	
Description	Calculate the values of availability for the urls
Parameters	url = string
Returns	Availability in Boolean 0 or 1

2.5.8 getLatency:

getLatency(url)	
Description	Calculate the values of latency for the urls
Parameters	url = string
Returns	Latency in seconds

2.5.9 put_data:

put_data(self, namespace, metricName, dimensionPair, value)	
Description	Get the reuired data from generated event and puts them in DynamoDB table with proper indexing

Parameters	nameSpace(string):
	metricName(string):
	dimensionPair(string):
	value(int):

Returns	None

2.5.10 post_get_put_delete

post_get_put_delete(id, url, method):	
Description	Get the reuired data from generated event and puts them in DynamoDB table with proper indexing according to CRUD operations.

Parameters	id = string
	url = string
	method = string

Returns	Json response

3 REFERENCES

1. https://tkssharma-devops.gitbook.io/devops-training/devops-01-continuous-integration/aws-code-pipeline-ci-cd/aws-ci-cd-tools/aws-code-build
2. https://k21academy.com/amazon-web-services/deploy-aws-codepipeline/
3. https://medium.com/@abhibvp003/using-cloudwatch-alarms-monitor-aws-resources-7f38b5f5fddd
4. https://aws.amazon.com/dynamodb/
5. https://aws.amazon.com/api-gateway/
6. https://medium.com/hackernoon/continuous-deployment-with-aws-codedeploy-github-d1eb97550b82
7. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.Errors.html