SkipQ – Pegasus (Cohort-6)

Documentation

Documented by: Muhammad Faizan Ikram

Table of Contents

Project Document

1 Training Overview

2 Sprint 3

2.1 Objective

The objective of this Sprint3 is to build up on Sprint2 and create a multi-stage pipeline having Beta/Gamma and Prod stage using CDK and also deploy the project code in one or multiple regions. The core objectives are listed as follows:

- The Each stage must have bake Times, code-review, and test blockers.
- Write unit/integration tests for the web crawler.
- Emit CloudWatch metrics and alarms for the operational health of the web crawler, including memory and time-to-process each crawler run.
- Automate rollback to the last build if metrics are in alarm. Manage README files and run-books in markdown on GitHub.

2.2 Technologies Used

2.2.1 AWS CI/CD Pipeline

A pipeline is a process that drives software development through a path of building, testing, and deploying code, also known as CI/CD. By automating the process, the objective is to minimize human error and maintain a consistent process for how software is released. Tools that are included in the pipeline could include compiling code, unit tests, code analysis, security, and binaries creation. For containerized environments, this pipeline would also include packaging the code into a container image to be deployed across a hybrid cloud. CI/CD is the backbone of a DevOps methodology, bringing developers and IT operations teams together to deploy software. As custom applications become key to how companies differentiate, the rate at which code can be released has become a competitive differentiator.

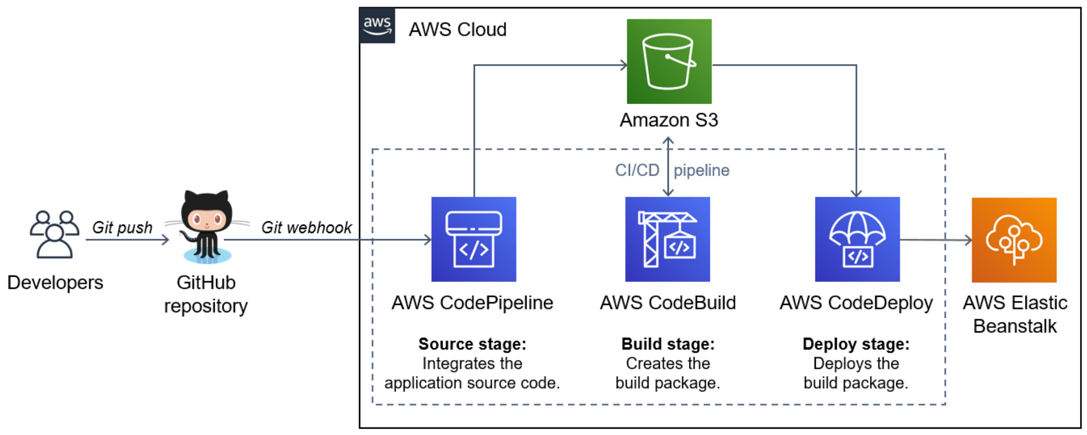


Figure 1: CICD Pipeline [1]

2.2.2 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.

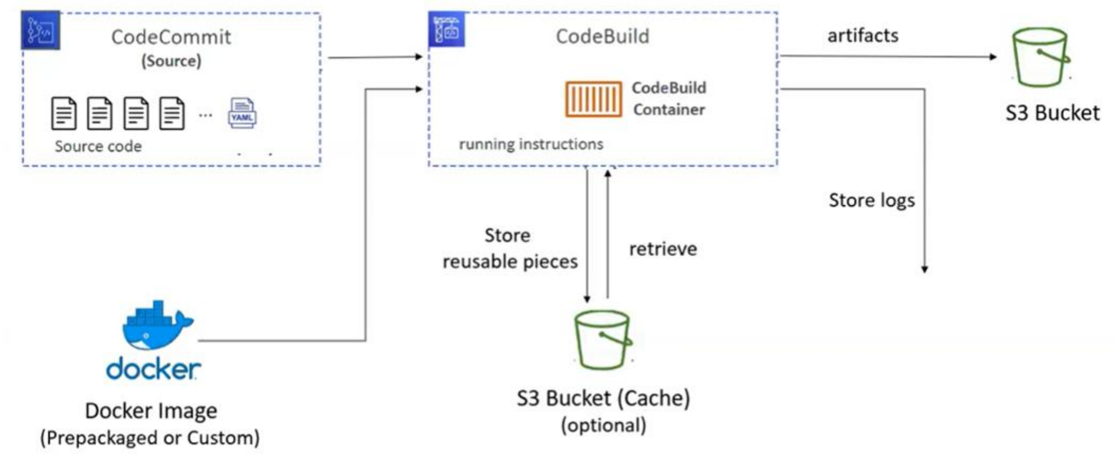


Figure 2: CodeBuild workflow [2]

2.2.3 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.

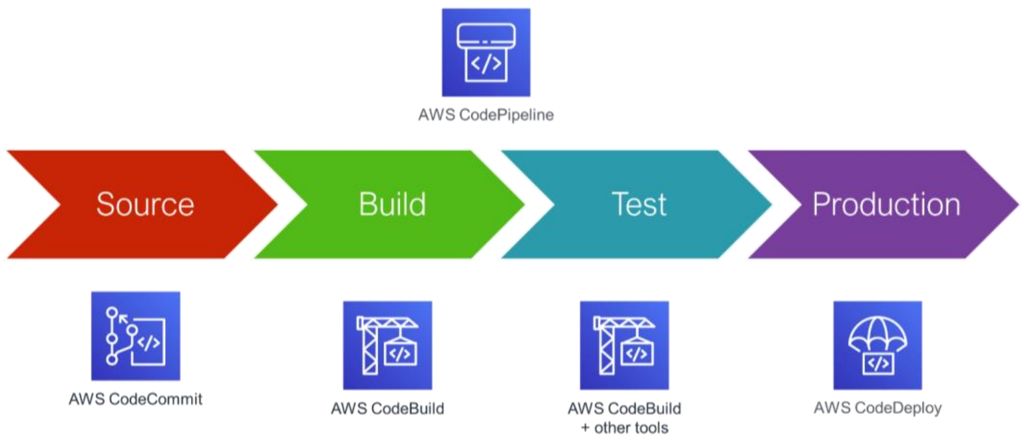


Figure 3: CodePipeline Process [3]

2.2.4 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.

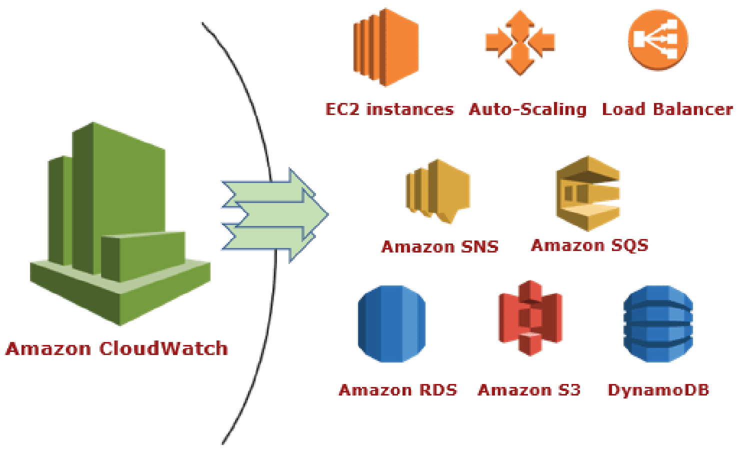


Figure 4: Resource monitored by CloudWatch [4]

2.2.5 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.

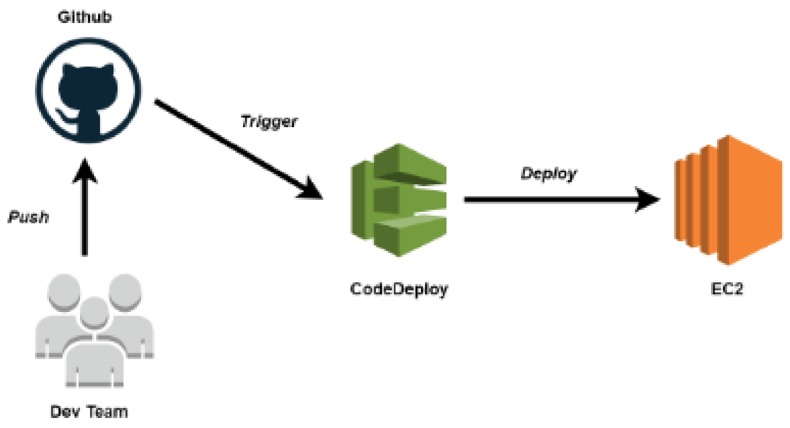


Figure 5: GitHub Integration with EC2 [5]

2.3 Sprint Breakdown

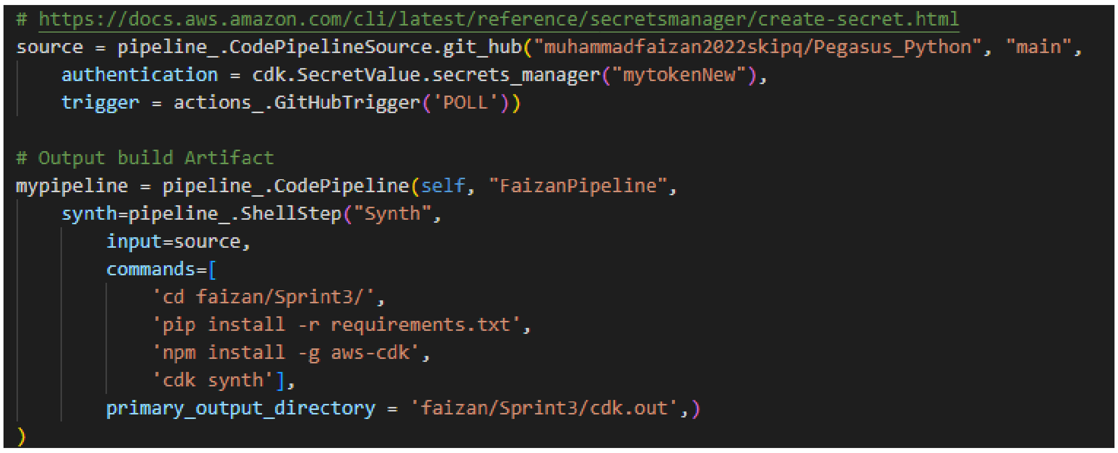
2.3.1 Task – 1: Create a multi-stage pipeline architecture

In this task, we had to create a multi-stage pipeline architecture having Beta/Gamma and Prod stage using CDK. Currently, it was directly pointing from the app.py to the main stack file. So we created the architecture in which app calls the pipeline stack which has source and stages. This pipeline stack then calls the pipeline stage file which was linked to the end application stack file.

2.3.2 Task – 2: Add source and build artifact

In this task, we defined our source (GitHub repository) for the pipeline to access the CommitId of a GitHub source in the synth and get the code to build it using CDK. The output build artifact was created using CodePipeline which builds the source code using ShellStep.

Code:



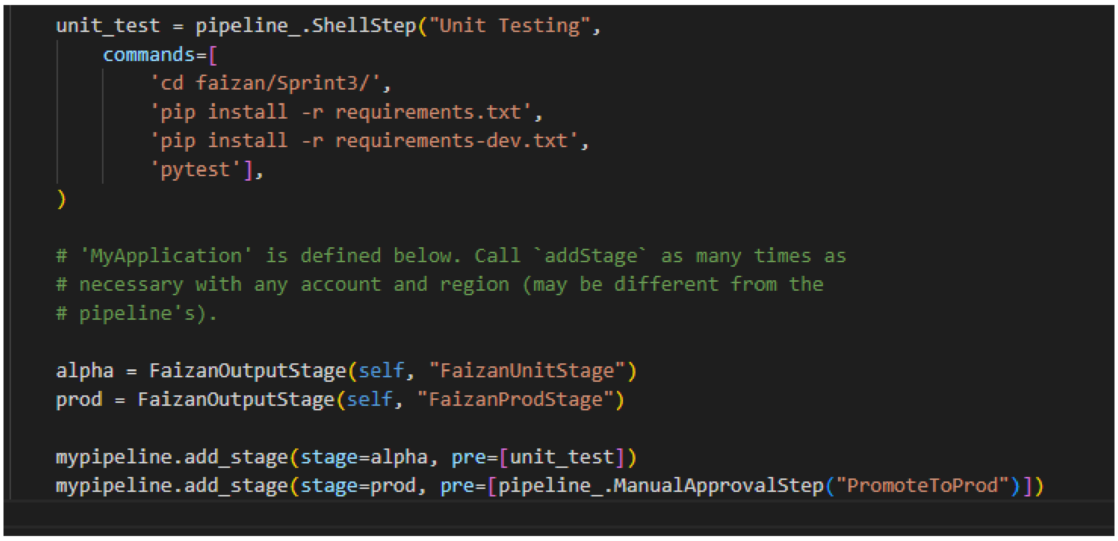
```
# https://docs.aws.amazon.com/cli/latest/reference/secretsmanager/create-secret.html
source = pipeline_.CodePipelineSource.git_hub("muhammadfaizan2022skipq/Pegasus_Python", "main",
    authentication = cdk.SecretValue.secrets_manager("mytokenNew"),
    trigger = actions_.GitHubTrigger('POLL'))

# Output build Artifact
mypipeline = pipeline_.CodePipeline(self, "FaizanPipeline",
    synth=pipeline_.ShellStep("Synth",
        input=source,
        commands=[
            'cd faizan/Sprint3/',
            'pip install -r requirements.txt',
            'npm install -g aws-cdk',
            'cdk synth'],
        primary_output_directory = 'faizan/Sprint3/cdk.out',)
)
```

Figure 6: Code snippet of source and build artifact

2.3.3 Task – 3: Create and add alpha/beta/gamma and manual approval stages

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:



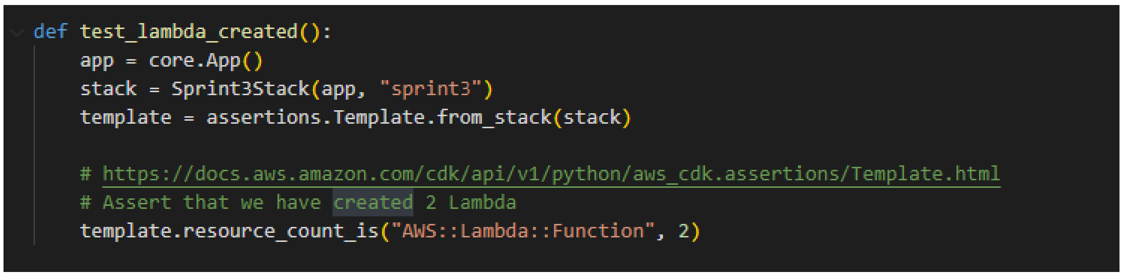
```
    unit_test = pipeline_.ShellStep("Unit Testing",
        commands=[
            'cd faizan/Sprint3/',
            'pip install -r requirements.txt',
            'pip install -r requirements-dev.txt',
            'pytest'],
    )

    # '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")])
```

Figure 7: Code snippet for stages

2.3.4 Task – 4: Create unit tests

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:



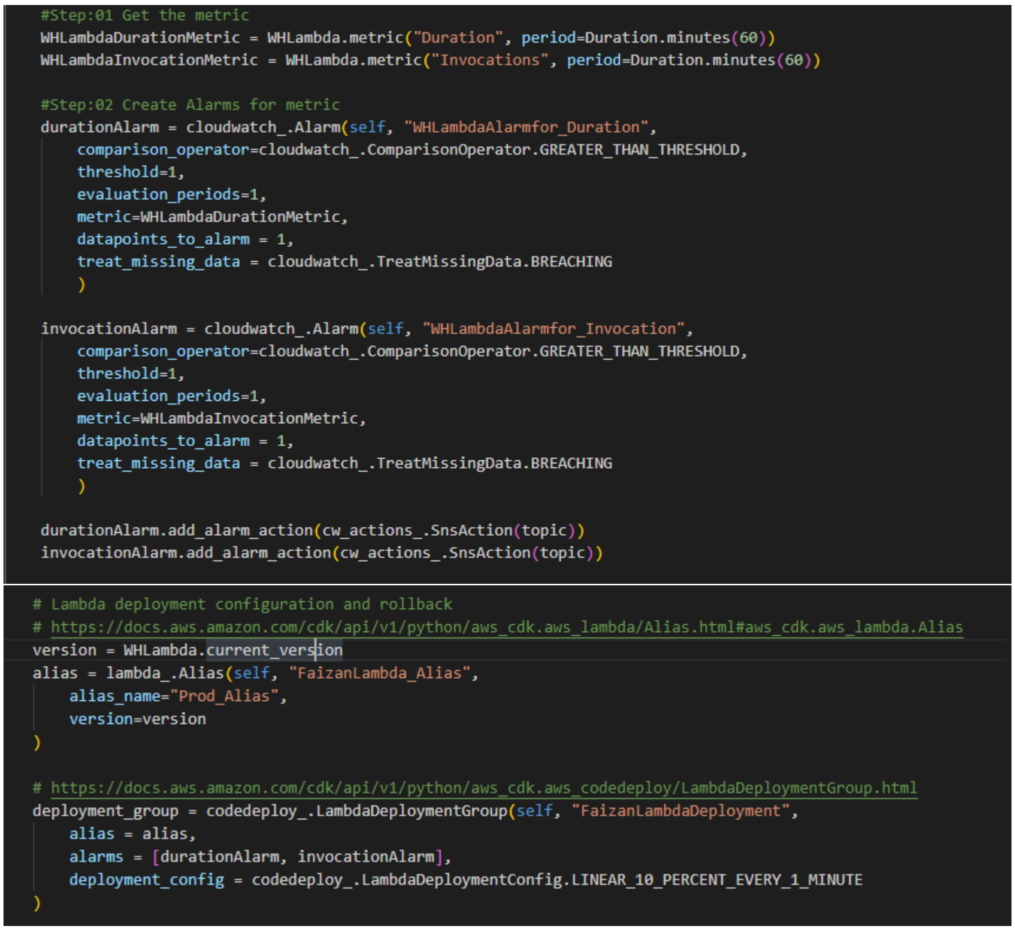
```python
def test_lambda_created():
    app = core.App()
    stack = Sprint3Stack(app, "sprint3")
    template = assertions.Template.from_stack(stack)

    # https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.assertions/Template.html
    # Assert that we have created 2 Lambda
    template.resource_count_is("AWS::Lambda::Function", 2)
```

Figure 8: Sample code snippet of a unit test

2.3.5 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))
```

```python
# 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
)
```

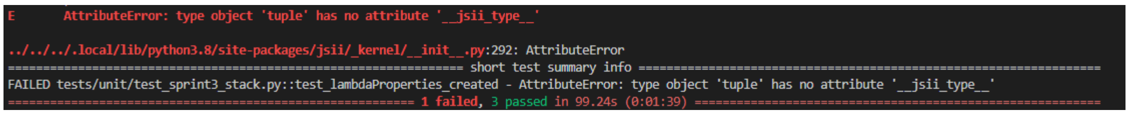
Figure 9: Code snippet for metric, alarm and lambda configuration

2.4 Issues/Errors and Troubleshooting

2.4.1 AttributeError: type object 'tuple' has no attribute '__jsii_type__'

- **Description**:

  I was receiving this error because I added wrong properties in test function for the DynamoDB table.



```
E       AttributeError: type object 'tuple' has no attribute '__jsii_type__'

../../../.local/lib/python3.8/site-packages/jsii/_kernel/__init__.py:292: AttributeError
================================================================ short test summary info ================================================================
FAILED tests/unit/test_sprint3_stack.py::test_lambdaProperties_created - AttributeError: type object 'tuple' has no attribute '__jsii_type__'
========================================================== 1 failed, 3 passed in 99.24s (0:01:39) ==========================================================
```

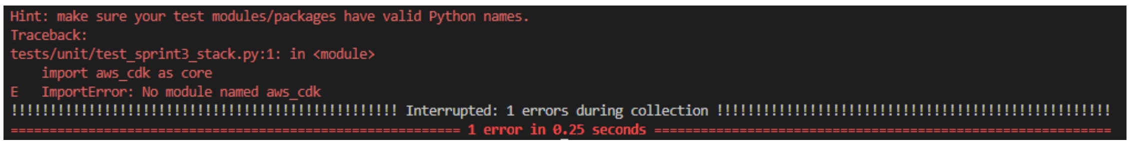
- **Solutions**:

  1. Add the attributes that were needed for test.
  2. Pass them in dictionary format {} as second argument for the test of specific properties.

2.4.2 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.



```
Hint: make sure your test modules/packages have valid Python names.
Traceback:
tests/unit/test_sprint3_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 0.25 seconds ============================================================
```

- **Solution**:

  1. Activate virtual environment using command *pip install -r requirements.txt*
  2. Install the requirements in virtual environment using command *pip install -r requirements.txt*
  3. Add the path to bin using command *export PATH=$PATH:$(npm get prefix)/bin*

2.4.3 FaizanPipeline should be created in the scope of a Stack, but no Stack found

- **Description**:

  I was getting this error because I didn't provide scope for the pipeline stack file. The scope for the service should be clearly described.

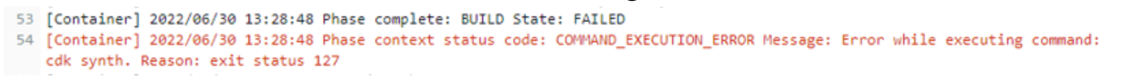
- **Solution**:

  1. I added self as scope because I was working on the same stack file. So the scope was *self*.

2.4.4 BUILD State: FAILED

- **Description**:

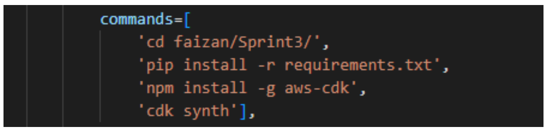
  I was getting this error while deploying my pipeline on AWS. The build stage in pipeline got failed while executing the *cdk synth* command. I didn't provide aws-cdk installation command and when added it, the order of commands was wrong.



```
[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**:

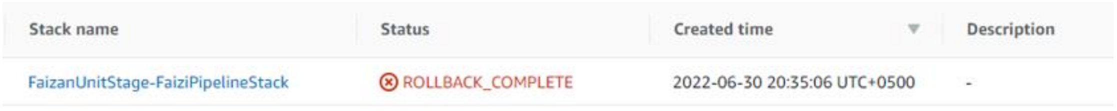
  1. For correct execution of *cdk synth*, add the aws-cdk command and the commands in pipeline should be given in following order:


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

2.4.5 ROLLBACK_COMLETE: FAILED

- **Description**:

  I was getting this error while deploying my pipeline on AWS. The test deploy stage in pipeline got failed because I added multiple test cases in one function and also there was issue in metric definitions for the duration and invocations.



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

- **Solution**:

  1. Add unit tests separately and create separate functions for the different tests.
  2. Check the definition of metric and duration defined.

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 Create_table:

lambda_handler(event,context)	
Description	Creates the table in the Dynamo DB
Parameters	None
Returns	dynamo_db table

2.5.5 Create_role:

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

2.5.6 getAvailability:

GetAvailability(url)	
Description	Calculate the values of availability for the urls
Parameters	None
Returns	Availability in Boolean 0 or 1

2.5.7 getLatency:

GetLatency(url)	
Description	Calculate the values of latency for the urls
Parameters	None
Returns	Latency in seconds

2.5.8 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

3 REFERENCES

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