Production Grade Programming

AWS DevOps

Documentation

By

Trainee: Shanawar Ali Chouhdry

In Partial Fulfillment

Of the Requirements for the graduation

from SKIPQ

Table of Contents

Sprint 1:

Create a web health monitoring application that triggers alarms when thresholds are breached and gives notification on email and update alarm information on alarm

Flow Diagram:

1. Lambda sends web health metric data to Cloud Watch.
2. Sns take a message from Cloud watch if alarm triggers to email.
3. Sns send a notification to lambda with the message as a payload.
4. Lambda save message in DynamoDB.

AWS:

Amazon web service is an online platform that provides scalable and cost-effective cloud computing solutions. AWS is a broadly adopted cloud platform that offers several on-demand operations like compute power, database storage, content delivery, etc., to help corporates scale and grow.

AWS-CDK:

The AWS Cloud Development Kit (AWS CDK) is an open-source software development framework to define your cloud application resources using familiar programming languages.

AWS-SDK:

AWS SDK (software development kit) helps simplify your coding by providing JavaScript objects for AWS services. It allows developers to access AWS from JavaScript code that runs directly in the browser, and it includes access to AWS components like Amazon S3, Amazon SNS, Amazon SQS, DynamoDB, and more.

Libraries:

```
# IMPORTING LIBRRARIES
from aws_cdk import (
    core as cdk,
    aws_lambda as lambda_,
    aws_events as events_,
    aws_events_targets as targets_,
    aws_iam,
    aws_cloudwatch as cloudwatch_,
    aws_sns as sns,
    aws_sns_subscriptions as subscriptions_,
    aws_cloudwatch_actions as actions_,
    aws_dynamodb as dynamodb_,
)
from resources import constants as constants
import os
```

Task 1: Create Hello Lambda Function

Description:

You organize your code into Lambda Functions. Lambda runs your function only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time that you consume—there is no charge when your code is not running.

Code:

Create Lambda:

```
################### CREATE LAMBDA FUNCTION ################
  def create_lambda(self,newid,asset,handler,role):
      return lambda_.Function(self, id=newid,
      runtime=lambda_.Runtime.PYTHON_3_6,
      handler=handler,
      code=lambda_.Code.from_asset(asset),
      role=role)
###########################################################
```

Invoke the Lambda function:

Invoke your Lambda function using the sample event data provided in the console.

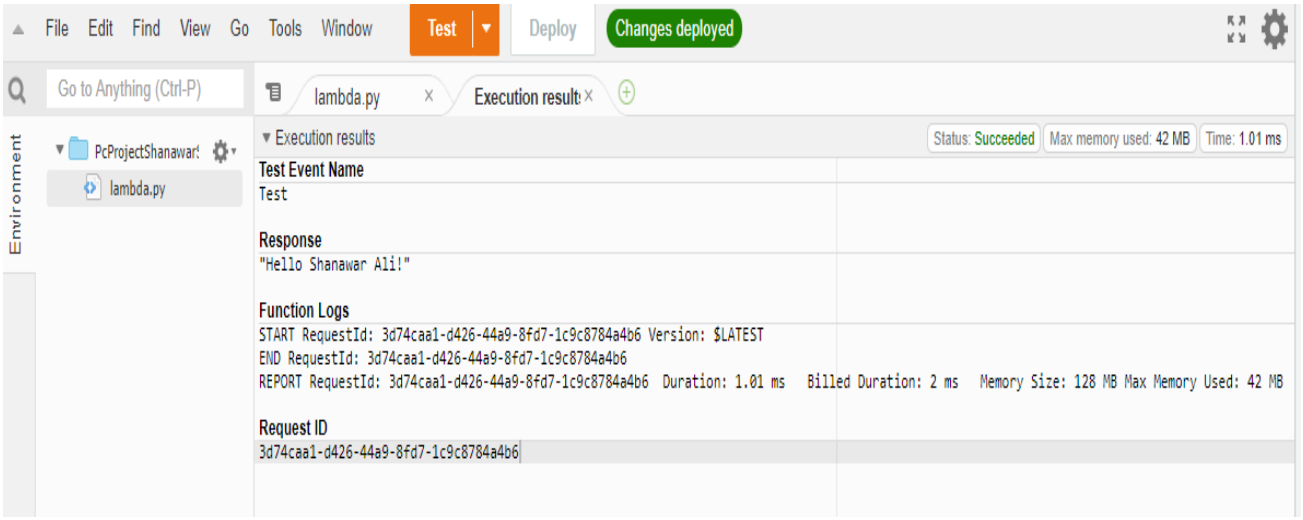
To invoke a function

1. After selecting your function, choose the Test tab.
2. In the Test event section, choose New event. In Template, leave the default hello-world option. Enter a Name for this test and note the

following sample event template:

```
# LAMBDA handler is the method in function code that processes events
def lambda_handler(event,context):
    return 'Hello {} {}!'.format(event['first name'],event['last name'])
```

Test Results:



Task 2: Monitor Web Health Metrics on CloudWatch

Description:

Amazon CloudWatch is a component of Amazon Web Services that provides monitoring for AWS resources and the customer applications running on the Amazon infrastructure.

CloudWatch enables real-time monitoring of AWS resources. The application automatically collects and provides metrics for CPU utilization, latency, and request count. Users can also stipulate additional metrics to be monitored, such as memory usage, transaction volumes, or error rates.

Get Availability:

Get availability of URL by checking to get URL. If the response is 200, it returns 1(available).

Get Latency:

Get Latency works by calculating the time difference between getting the website.

```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_availability():
    http=urllib3.PoolManager()
    response=http.request("GET",constants.URL_to_Monitor)
    if response.status==200:
        return 1.0
    else:
        return 0.0

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def get_latency():
    http=urllib3.PoolManager()
    start=datetime.datetime.now()
    response=http.request("GET",constants.URL_to_Monitor)
    end=datetime.datetime.now()
    diff=end-start
    latency_sec=round(diff.microseconds * 0.000001,6)
    return latency_sec
```

Cloudwatch_PutData:

Boto3:

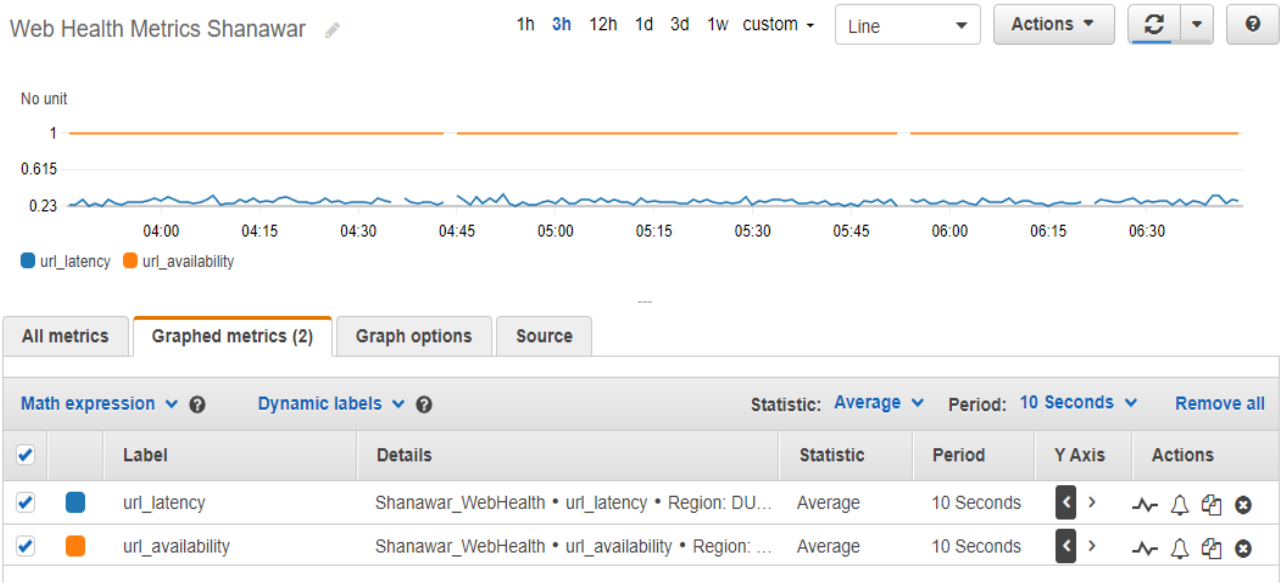
Boto 3 is the name of the Python SDK for AWS. It allows you to directly create, update, and delete AWS resources from your Python scripts.

Putting Data:

We have to put data to CloudWatch metrics so that we can see CloudWatch metrics on the AWS CloudWatch.

```python
import boto3
import constants as constants

class CloudWatch_PutMetric:
    def __init__(self):
        self.client=boto3.client('cloudwatch')
    def put_data(self, nameSpace,metricName, dimensions,value):
        response= self.client.put_metric_data(
            Namespace=nameSpace,
            MetricData=[
                {
                    'MetricName':metricName,
                    'Dimensions':dimensions,
                        'Value':value
                }
                ]
            )
```

CloudWatch Dashboard:



TASK 3: Updating DynamoDB Table

Description: Alarm raised should go to email using SNS and also stored in DynamoDB Table.

Periodic Lambda: You have to create periodic lambda to see it over a longer continuous period.

Create DynamoDB Table:

The CreateTable operation adds a new table to your account. In an AWS account, table names must be unique within each Region. That is, you can have two tables with the same name if you create the tables in different Regions.

CreateTable is an asynchronous operation. Upon receiving a CreateTable request, DynamoDB immediately returns a response with a TableStatus of CREATING. After the table is created, DynamoDB sets the TableStatus to ACTIVE. You can perform read and write operations only on an ACTIVE table.

```
# CREATING DYNAMODB TABLE
table_name="ShanawarAlarmTable"
dbtable = dynamodb_.Table(self, "ShanawarDBTable",table_name=table_name,
partition_key=dynamodb_.Attribute(name="MessageID", type=dynamodb_.AttributeType.STRING))
dbtable.grant_read_write_data(DBLambda)
DBLambda.add_environment('table_name',table_name)
```

DBLambda:

The event is triggered when the alarm is raised. Then Message and Timestamp are retrieved from the event and put onto the DynamoDB table.

```
import boto3,os
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(event, context):
    # BOTO3 CLIENT
    db=boto3.client('dynamodb')
    # SEPARATE MESSAGE ID and TIMESTAMP
    Message = event['Records'][0]['Sns']['MessageId']
    Timestamp = event['Records'][0]['Sns']['Timestamp']
    table_name=os.getenv('table_name')
    # UPDATE TABLE WITH ITEMS
    db.put_item(TableName=table_name,Item={
        'MessageID':{'S':Message},
        'TimeStamp':{'S':Timestamp}
    })
```

Email Subscription:

SNS will send email subscription

Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:
arn:aws:sns:us-east-2:315997497220:PcProjectShanawarStack-WebHealthEmailingbyShanawar7531AF37-DASND0BID31X:e5bec63f-bf9d-4676-8213-e1f4d1b86cfc

If it was not your intention to subscribe, click here to unsubscribe.

AWS Notifications <no-reply@sns.amazonaws.com> 9:11 AM (6 minutes ago)

to me

You are receiving this email because your Amazon CloudWatch Alarm "PcProjectShanawarStack-LatencyAlarm5394FC57-1SC63GNWC478G" in the US East (Ohio) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [0.300699 (20/12/21 04:10:00)] was greater than the threshold (0.28) (minimum 1 datapoint for OK -> ALARM transition)." at "Monday 20 December, 2021 04:11:57 UTC".

View this alarm in the AWS Management Console:
https://us-east-2.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-2#alarmsV2:alarm/PcProjectShanawarStack-LatencyAlarm5394FC57-1SC63GNWC478G

Alarm Details:
- Name: PcProjectShanawarStack-LatencyAlarm5394FC57-1SC63GNWC478G
- Description:
- State Change: OK -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [0.300699 (20/12/21 04:10:00)] was greater than the threshold (0.28) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Monday 20 December, 2021 04:11:57 UTC

...

Latency Alarms:

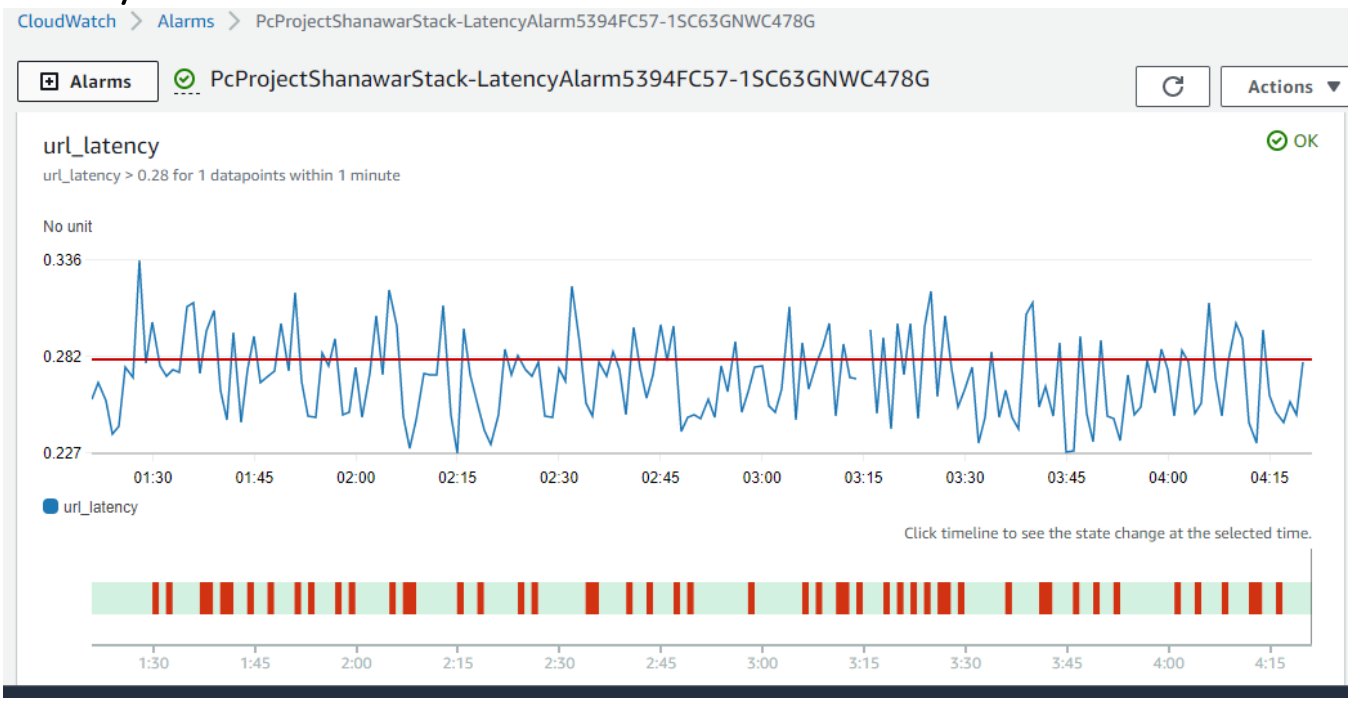
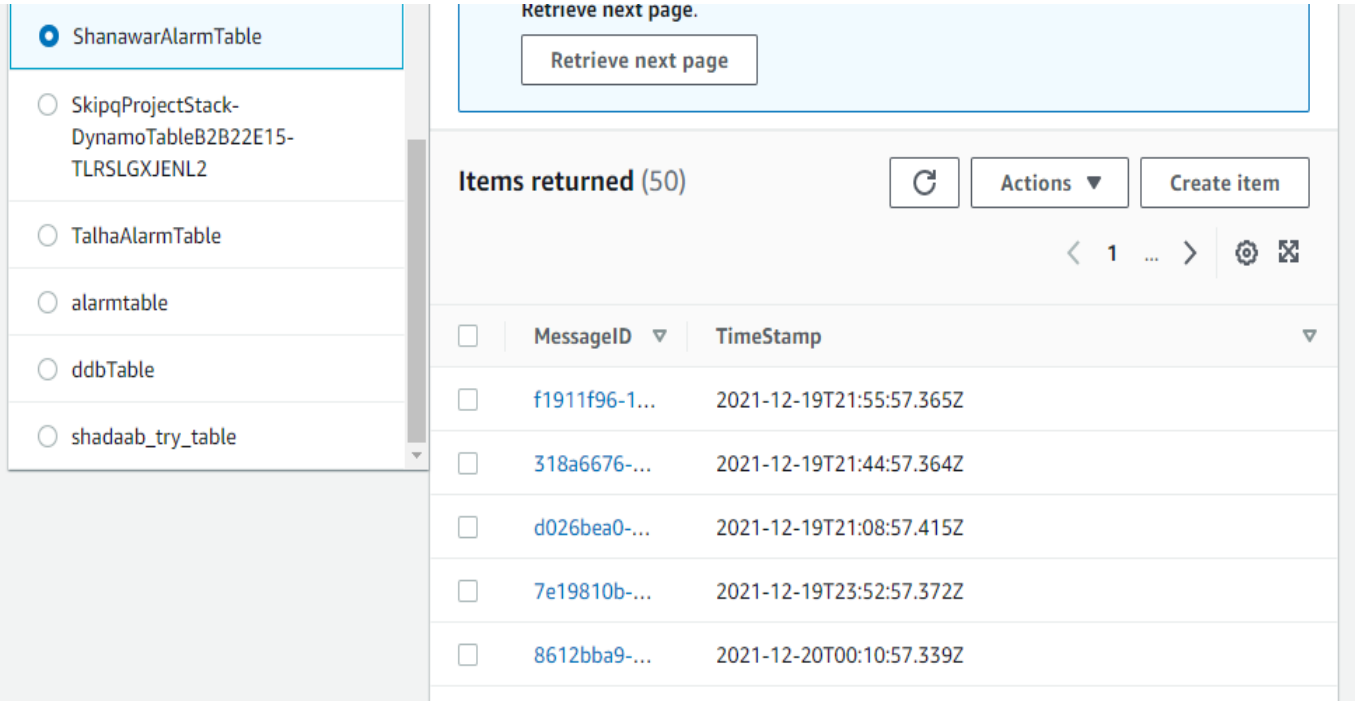


Table:



Problems Faced

- Updating python to python3. Solving using bashrc file but alias python = python3

- Unable to update DynamoDB table

```
dbtable.grant_read_write_data(DBLambda)
DBLambda.add_environment('table_name',table_name)
```

- Unable to receive notifications

```
assumed_by=aws_iam.CompositePrincipal(
    aws_iam.ServicePrincipal("lambda.amazonaws.com"),
    aws_iam.ServicePrincipal("sns.amazonaws.com")
    ),
managed_policies=[
    aws_iam.ManagedPolicy.from_aws_managed_policy_name('service-role/AWSLambdaBasicExecutionRole'),
    aws_iam.ManagedPolicy.from_aws_managed_policy_name('CloudWatchFullAccess'),
    aws_iam.ManagedPolicy.from_aws_managed_policy_name("AmazonDynamoDBFullAccess"),
    aws_iam.ManagedPolicy.from_aws_managed_policy_name("AmazonSNSFullAccess")
    ])
return lambdaRole
```

- Unable to retrieve data. I then realized that the DynamoDB lambda handler will only get values when the alarm is triggered.

Sprint 2

Description:

Create multi-stage pipeline having Beta/Gamma and Prod stage using CDK. Deploy the project code in 1 Region. 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.

Concepts:

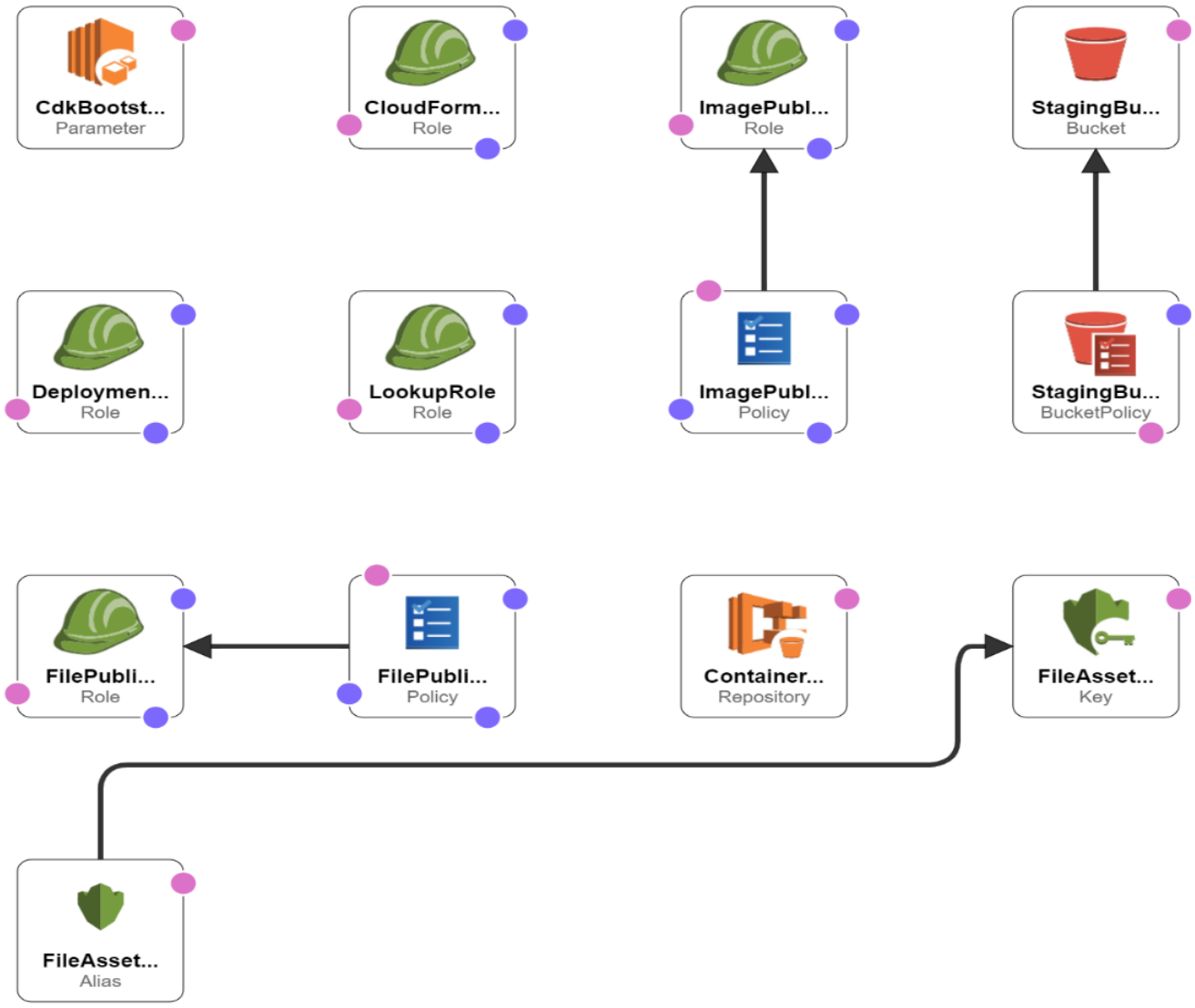
- Introduction to Cl/CD
- Learn AWS services: CodePipeline for build and test, CodeDeploy for CD
- Integrate AWS CodePipeline with GitHub
- Learn automated testing using PyTest running
- Build operational CloudWatch metrics for a web crawler
- Write rollback automation allowing rollback to the last build
- Setup beta and prod environments in CodePipeline and deploy using CodeDeploy

Designer Template:

AWS CloudFormation Designer (Designer) is a graphic tool for creating, viewing, and modifying AWS CloudFormation templates. With Designer, you can diagram your template resources using a drag-and-drop interface, and then edit their details using the integrated JSON and YAML editor. Whether you are a new or an experienced AWS CloudFormation user, AWS CloudFormation Designer can help you quickly see the interrelationship between a template's resources and easily modify templates.

Pipeline Toolkit Stack:

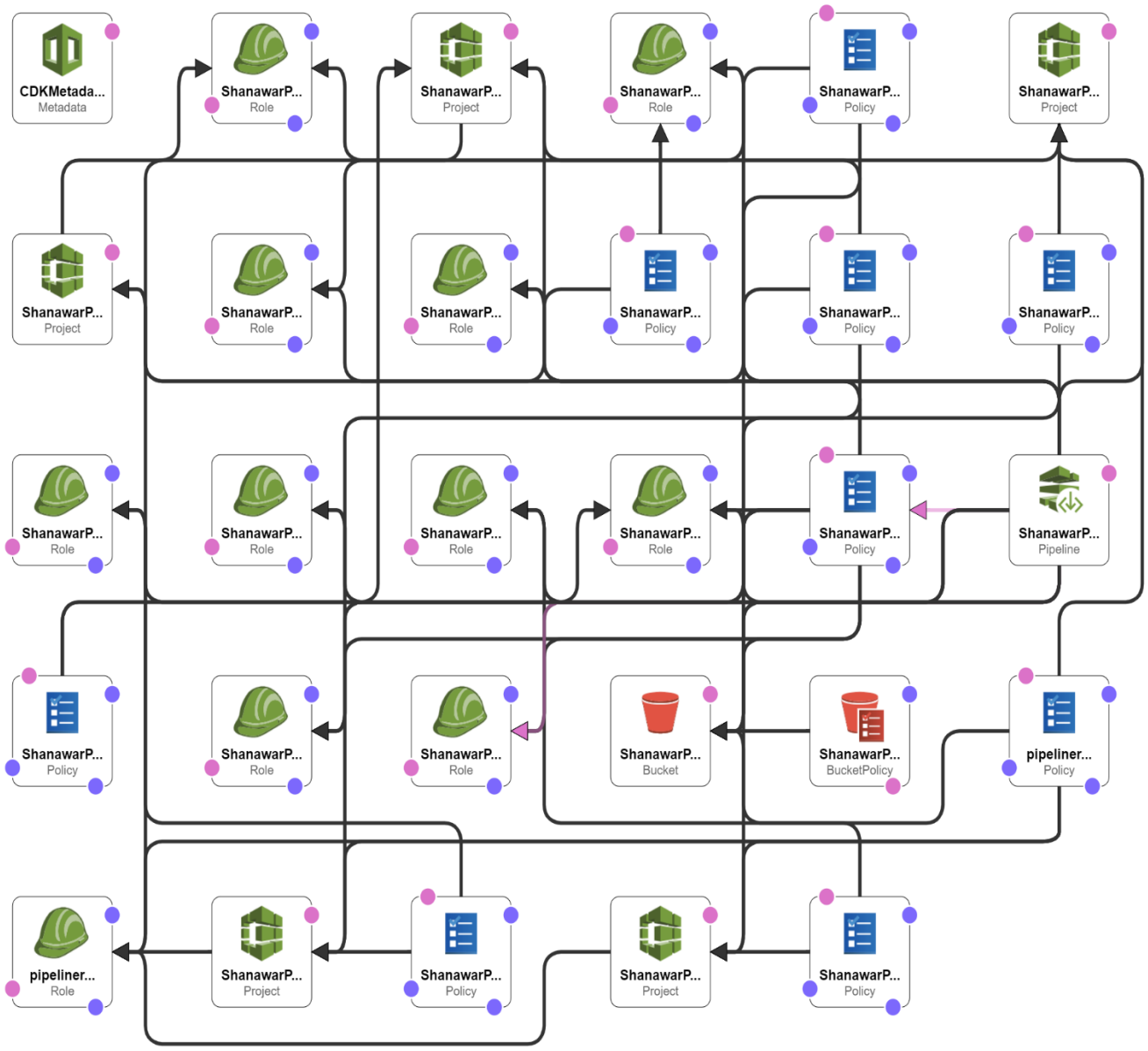
Deploying AWS CDK apps into an AWS environment (a combination of an AWS account and region) may require that you provision the resources the AWS CDK needs to perform the deployment. These resources include an Amazon S3 bucket for storing files and IAM roles that grant permissions needed to perform deployments. The process of provisioning these initial resources is called bootstrapping.



Pipeline Stack:

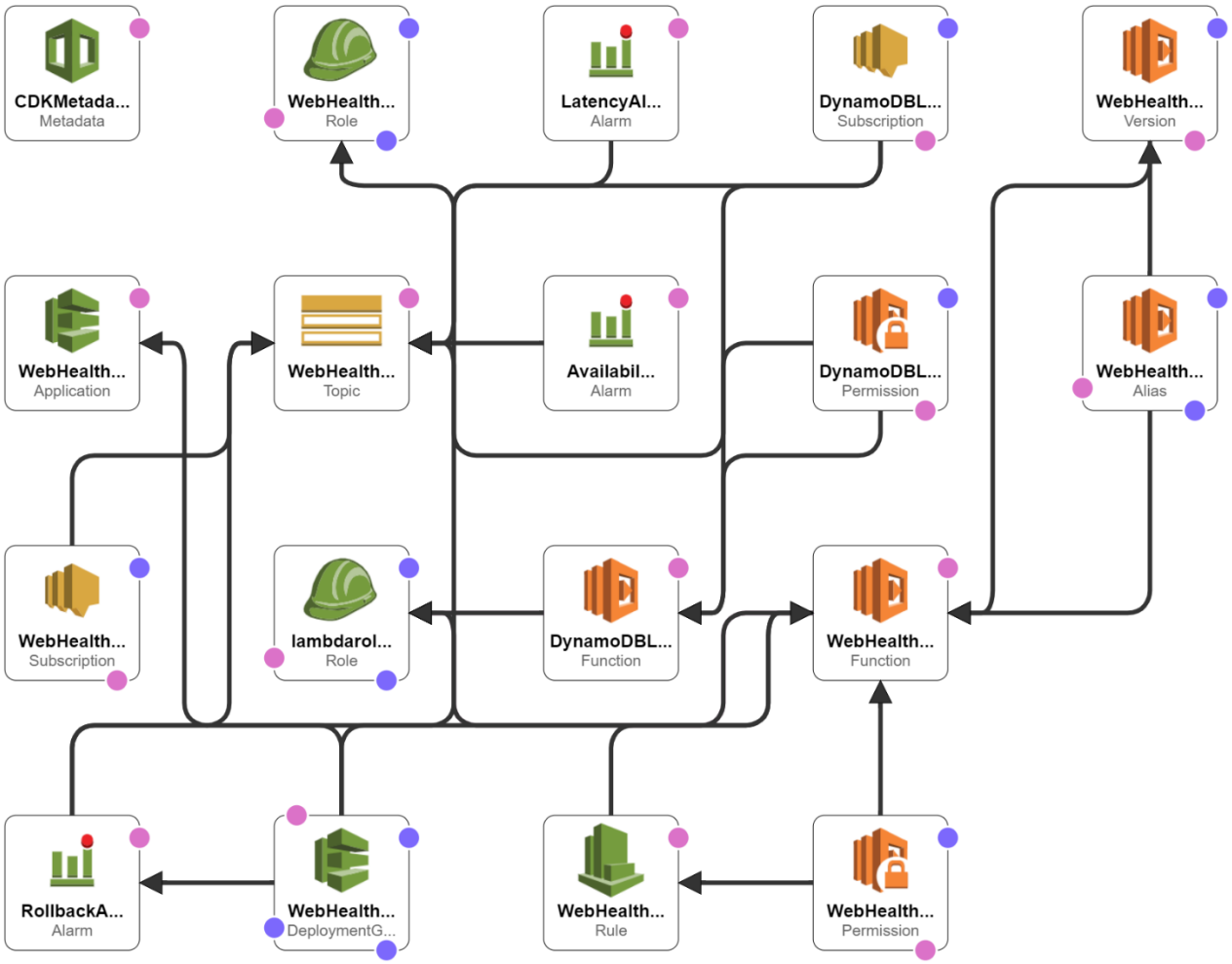
The pipeline stack determines the flow structure of the pipeline and self-updates actions whenever new code is pushed to the source repository. It contains resources like :

- Update pipeline action role
- Self mutate pipeline role
- S3 Artifact buckets
- Roles and policies for access
- Stages unit and integration tests

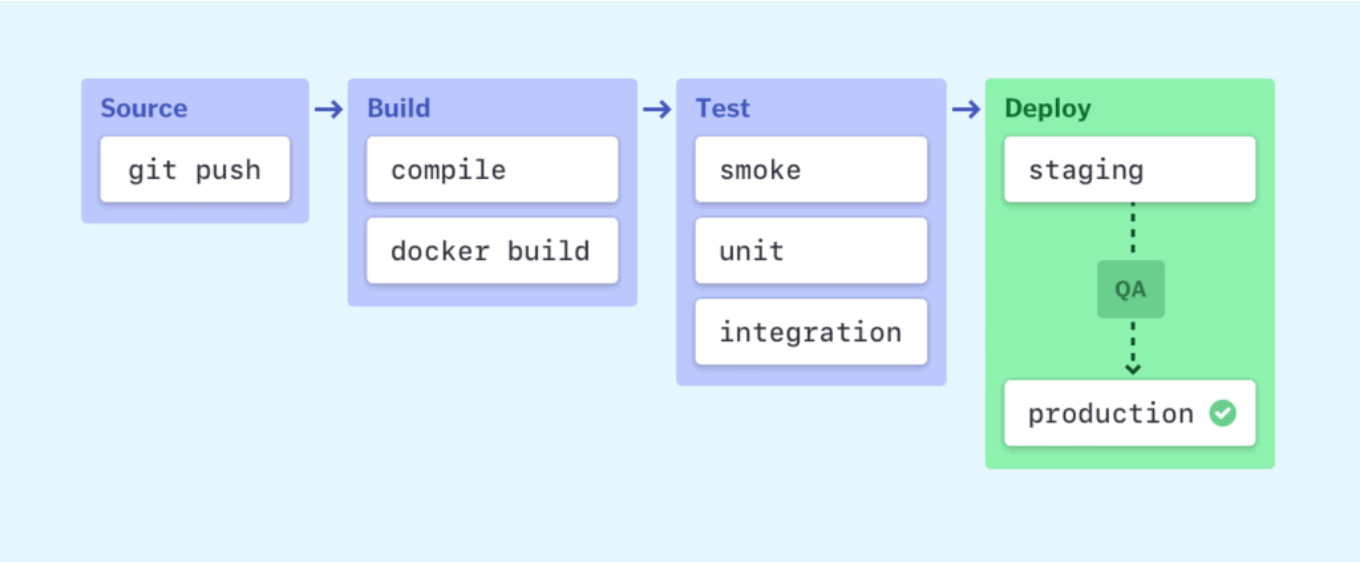


Production Stage Stack:

Production Stage contains all alarms, lambda functions, and SNS topics used in the project.



Building Pipeline:



Source:

The Source Stage is the first step of any CI/CD pipeline and it represents your source code. This stage is in charge of triggering the pipeline based on new code changes (i.e. git push or pull requests). In this sprint, we will be using GitHub as the source provider, but CodePipeline also supports S3, CodeCommit and Amazon ECR as source providers.

```
class PipelineStack(core.Stack):
    def __init__(self,scope:core.Construct,id:str,**kwargs):
        super().__init__(scope,id,**kwargs)
        pipelineroles=self.createrole()
        iamPolicy=aws_iam.PolicyStatement(resources=['*'],actions=['iam:*'])
        stsPolicy=aws_iam.PolicyStatement(resources=['*'],actions=['sts:*'])

        ############## STEP1: SOURCE is Github ##############
        source =pipelines.CodePipelineSource.git_hub(repo_string='shanawar2021skipq/ProximaCentauri-1',branch='main',
        authentication=core.SecretValue.secrets_manager('pipeline/shanawar',
        json_field="shanawarsecret"),
        trigger=cpactions.GitHubTrigger.POLL)
```

BUILD STAGE

The build stage is where your Serverless application gets built and packaged. We are going to use AWS CodeBuild as the Build provider for our pipeline. It is worth mentioning that CodePipeline also supports other providers like Jenkins, TeamCity, or CloudBees.

Why AWS CodeBuild?

AWS CodeBuild is a great option because you only pay for the time where your build is running, which makes it very cost effective compared to running a dedicated build server 24 hours a day when you really only build during office hours. It is also container-based which means that you can bring your own Docker container image where your build runs, or use a managed image provided by CodeBuild.

Add the build stage

```
############## STEP2: BUILD ###############
synth = pipelines.CodeBuildStep('Shanawar_synthesizing',input=source,
commands=["cd shanawar/sprint3","pip install -r requirements.txt", "npm install -g aws-cdk", "cdk synth"],
primary_output_directory="shanawar/sprint3/cdk.out",
role=pipelineroles,
role_policy_statements=[iamPolicy,stsPolicy]
)
```

Testing:

Integrating automated testing into a system is a crucial part of any pipeline workflow.

But shortening a feedback loop is one of the not-so-secret tips to creating an effective CI/CD pipeline. The shorter the time that developers spend waiting on test results, the more time they'll have to address bug fixes and make code changes.

Automated testing allows for scalability—an unskippable part of software development for companies looking to expand or grow. In fact, there really is no reason not to implement automated testing with a CI/CD pipeline.

```
unit_test = pipelines.CodeBuildStep(
    'unit_tests',input=source,
    commands=["cd shanawar/sprint2","pip install -r requirements.txt", "npm install -g aws-cdk", "pytest unit_tests"],
    role=pipelineroles,
    role_policy_statements=[iamPolicy,stsPolicy]
    )

integration_test = pipelines.CodeBuildStep(
    'integration_tests',input=source,
    commands=["cd shanawar/sprint2","pip install -r requirements.txt", "npm install -g aws-cdk", "pytest integration_tests"],
    role=pipelineroles,
    role_policy_statements=[iamPolicy,stsPolicy]

    )
```

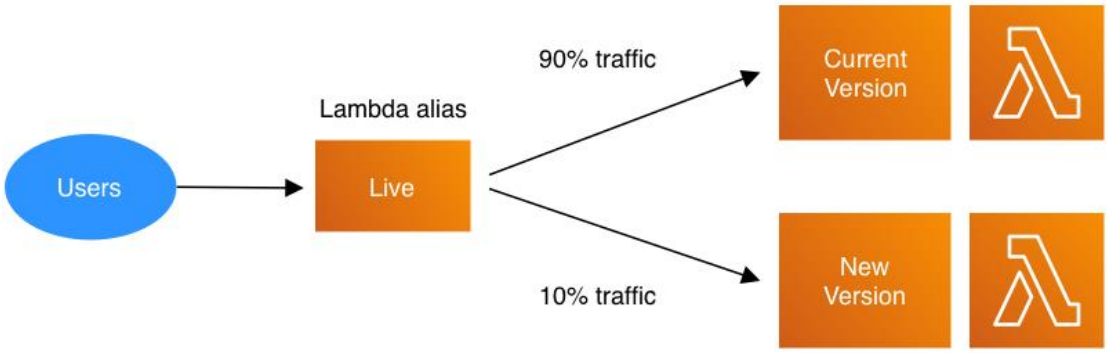
DEPLOY STAGE

The Deploy Stage is where your application and all its resources are created an in an AWS account. The most common way to do this is by using CloudFormation changesets to deploy. This means that this stage will have two actions: the *CreateChangeSet* and the *ExecuteChangeSet*.

```
    pipeline.add_stage(beta, pre=[unit_test],post=[pipelines.ManualApprovalStep("Post-Beta Check")])
    pipeline.add_stage(gamma, pre=[integration_test],post=[pipelines.ManualApprovalStep("Post-Gamma Check")])
################## STEP4: PROD ###################
    pipeline.add_stage(prod)
```

CANARY DEPLOYMENTS:

A Canary Deployment is a technique that reduces the risk of deploying a new version of an application by slowly rolling out the changes to a small subset of users before rolling it out to the entire customer base.




```
alias = lambda_.Alias(self, "Shanawar_WebHealthLambdaAlias"+construct_id,alias_name= 'Shanawar'+construct_id,version=WebHealthLambda.current_version)#)

"""
Parameters
scope (Construct) –

id (str) –

alias (Alias) – Lambda Alias to shift traffic.

alarms (Optional[Sequence[IAlarm]]) – The CloudWatch alarms associated with this Deployment Group.

auto_rollback (Optional[AutoRollbackConfig]) – The auto-rollback configuration for this Deployment Group. Default: - default AutoRollbackConfig.

deployment_config (Optional[ILambdaDeploymentConfig]) – The Deployment Configuration this Deployment Group uses. Default: LambdaDeploymentConfig.CANARY_10PERCENT_5MINUTES

"""
# Linear: Traffic is shifted in equal increments with an equal number of minutes between each increment.
# linear options specify the percentage of traffic that's shifted in each increment and the number of minutes between each increment.

codedeploy.LambdaDeploymentGroup(self, "Shanawar_WebHealthLambda_DeploymentGroup",
alias=alias,
deployment_config=codedeploy.LambdaDeploymentConfig.LINEAR_10_PERCENT_EVERY_1_MINUTE,
alarms=[rollback_alarm]
)
```

Rollback Metric and Alarm:

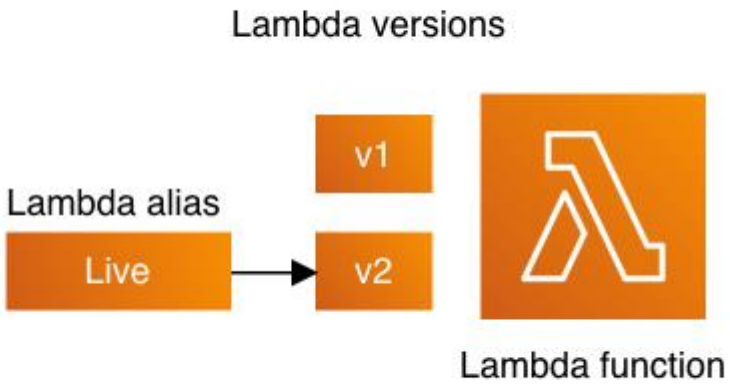
```
# DEFININING ROLLBACK METRIC
rollback_metric=cloudwatch_.Metric(
namespace='AWS/Lambda',
metric_name='Duration',
dimensions_map={'FunctionName':WebHealthLambda.function_name},
period= cdk.Duration.minutes(1))

# DEFININING ROLLBACK ALARM
rollback_alarm= cloudwatch_.Alarm(self,
id="RollbackAlarm",
metric= rollback_metric,
comparison_operator=cloudwatch_.ComparisonOperator.GREATER_THAN_THRESHOLD,
datapoints_to_alarm=1,
evaluation_periods=1,
threshold=800) # THRESHOLD IS IN MILLISECONDS

rollback_alarm.add_alarm_action(actions_.SnsAction(newtopic))
```

The concepts of blue/green and canary deployments have been around for a while and have been well-established as best-practices for reducing the risk of software deployments. In traditional applications, you slowly and incrementally update the servers in your fleet while simultaneously verifying application health. However, there is somewhat of an impedance mismatch when mapping these concepts to a serverless world. You can't incrementally deploy your software across a fleet of servers when there are no servers!

The answer is that there are a couple of services and features involved in making this possible.

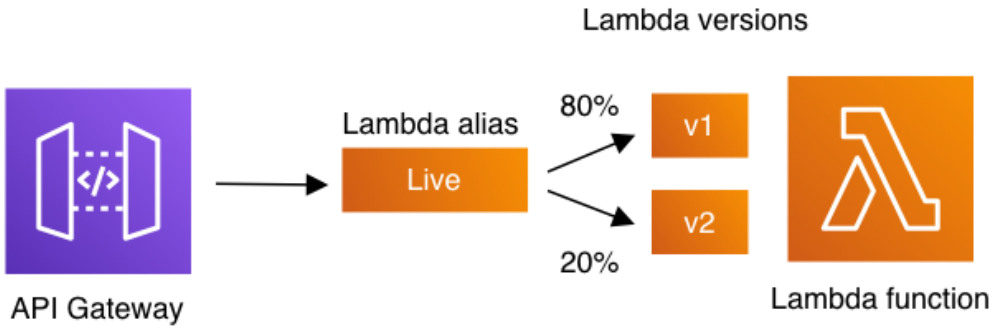
Lambda versions and aliases

AWS Lambda allows you to publish multiple versions of the same function. Each version has its own code and associated dependencies, as well as its own function settings (like memory allocation, timeout and environment variables). You can then refer to a given version by using a Lambda Alias. An alias is nothing but a name that can be pointed to a given version of a Lambda function.



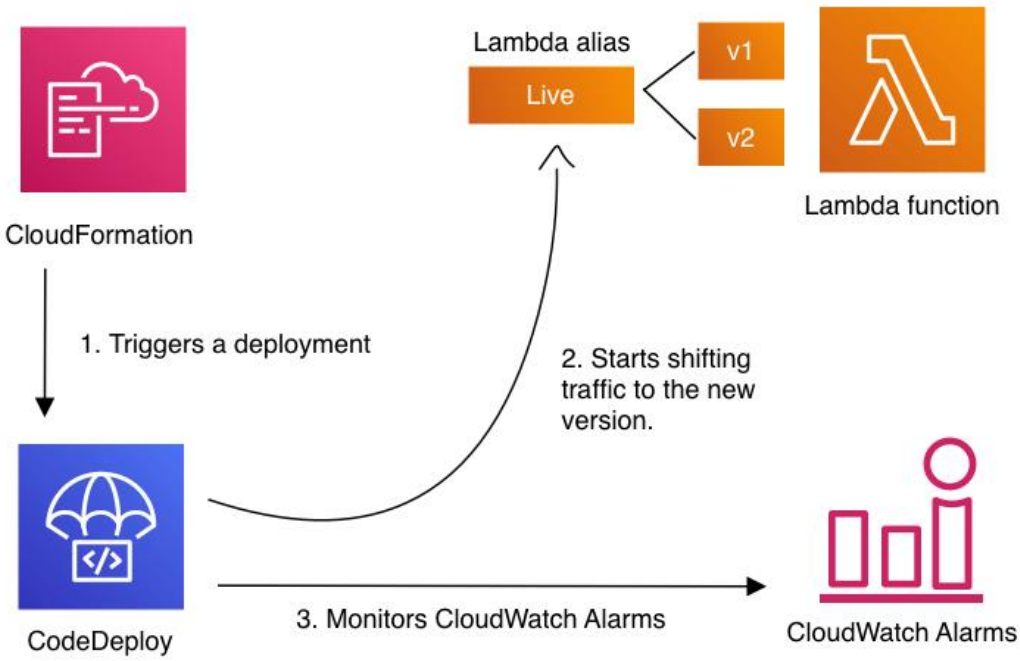
Traffic shifting with Lambda aliases

With the introduction of alias traffic shifting, it is now possible to trivially implement canary deployments of Lambda functions. By updating additional version weights on an alias, invocation traffic is routed to the new function versions based on the weight specified. Detailed CloudWatch metrics for the alias and version can be analyzed during the deployment, or other health checks performed, to ensure that the new version is healthy before proceeding.



Traffic shifting with CodeDeploy

Traffic-shifted deployments can be declared in infra stack, and CodeDeploy manages the function rollout as part of the CloudFormation stack update. CloudWatch alarms can also be configured to trigger a stack rollback if something goes wrong.



Issues:

CDK Bootstrap:

Cdk bootstrap keeps on giving update_rollback_complete error.

The reason was everyone was working with the same account in the same region. That's why it was giving a rollback error.

The issue can be resolved by giving a qualifier and CDK toolkit name when bootstrapping.

Update cdk.json file with these lines:

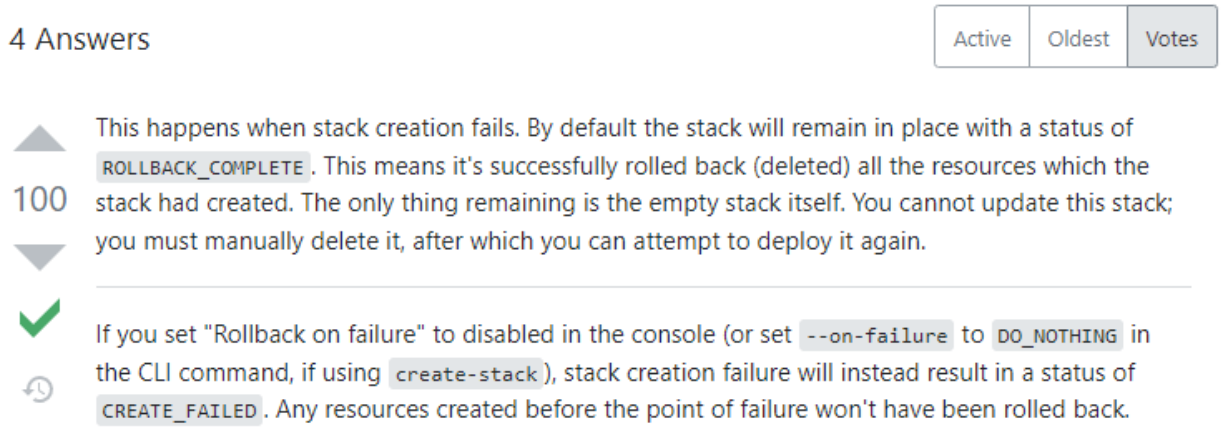
```
  "@aws-cdk/core:newStyleStackSynthesis": true,
  "@aws-cdk/core:bootstrapQualifier": "yourname"
```

Customized cdk command:

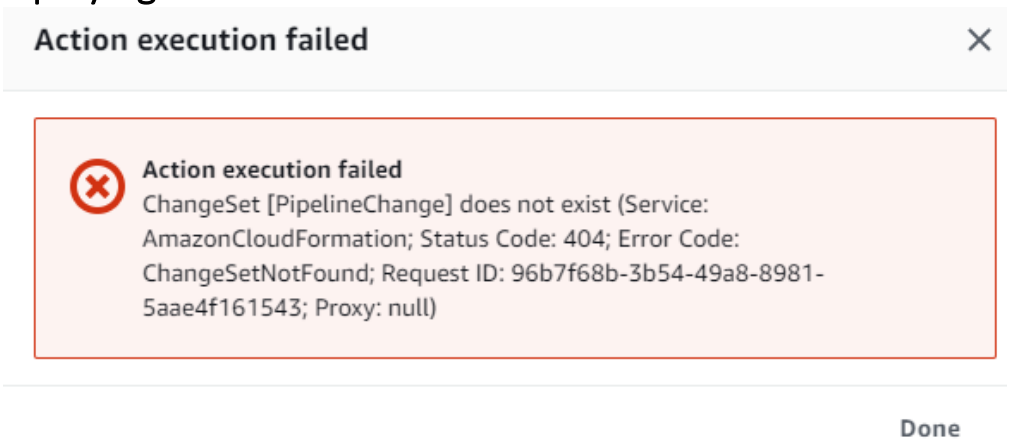
```
cdk bootstrap --qualifier "yourqualifier" --toolkit-stack-name
"yournametoolkit" --cloudformation-execution-policies
arn:aws:iam::aws:policy/AdministratorAccess account_no/your region
```

Rollback Complete Error:

If a rollback error happens to delete the stack and redeploying solves the problems.



Stage Deploying Failed:



Stage deployment can be failed in the pipeline. If the stack for a certain stage is deleted, it doesn't update by just pushing the code. It can only be deployed by the CDK deploy method.

Role Access Error

```
❌  shanawarpipeline failed: AccessDenied: User: arn:aws:sts::315997497220:assumed-role/shanawarpipeline-
ShanawarPipelineUpdatePipelineSel-OCSVCHA909CF/AWSCodeBuild-03897cc5-f2a9-4540-a1a7-35b24a77f64b is not authorized to
perform: cloudformation:GetTemplate on resource: arn:aws:cloudformation:us-east-
2:315997497220:stack/shanawarpipeline/3f5c8960-6662-11ec-9fc1-02261dac6e7c because no identity-based policy allows the
cloudformation:GetTemplate action
    at Request.extractError (/usr/local/lib/node_modules/aws-cdk/node_modules/aws-sdk/lib/protocol/query.js:50:29)
    at Request.callListeners (/usr/local/lib/node_modules/aws-cdk/node_modules/aws-sdk/lib/sequential_executor.js:106:20)
    at Request.emit (/usr/local/lib/node_modules/aws-cdk/node_modules/aws-sdk/lib/sequential_executor.js:78:10)
    at Request.emit (/usr/local/lib/node_modules/aws-cdk/node_modules/aws-sdk/lib/request.js:686:14)
    at Request.transition (/usr/local/lib/node_modules/aws-cdk/node_modules/aws-sdk/lib/request.js:22:10)
```

It's a best practice to be sure of the following:

Note: If you receive errors when running AWS Command Line Interface (AWS CLI) commands, make sure that you're using the most recent AWS CLI version.

Bob has permissions for AssumeRole.

You're signed in to the AWS Account as Bob. For more information, see your AWS account ID and its alias.

If Account_Bob is part of an AWS Organization, there might be a service control policy (SCP) restricting AssumeRole access with Account_Bob or Account_Alice. For more information, see service control policies (SCPs).

create role function and then pass it to the pipeline

```
def createrole(self):
    role=aws_iam.Role(self,"pipeline-role",
    assumed_by=aws_iam.CompositePrincipal(
        aws_iam.ServicePrincipal("lambda.amazonaws.com"),
        aws_iam.ServicePrincipal("sns.amazonaws.com"),
        aws_iam.ServicePrincipal("codebuild.amazonaws.com")
        ),
    managed_policies=[
        aws_iam.ManagedPolicy.from_aws_managed_policy_name('service-role/AWSLambdaBasicExecutionRole'),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name('CloudWatchFullAccess'),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name("AmazonDynamoDBFullAccess"),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name("AwsCloudFormationFullAccess"),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name("AmazonSSMFullAccess"),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name("AWSCodePipeline_FullAccess"),
        aws_iam.ManagedPolicy.from_aws_managed_policy_name("AmazonS3FullAccess")
        ])
    return role
```

Repository Sync Failed

GitHub repo in code pipeline shows that it failed to sync. The reason is when you create a secret key in secret manager, it enters it as JSON format. So, to access GitHub token, another parameter of json_field in source function.

Sprint 3:

Description:

Build a public CRUD API Gateway endpoint for the web crawler to create/read/update/delete the target list containing the list of websites/webpages to crawl. First, move the json file from S3 to a database (DynamoDB). Then implement CRUD REST commands on DynamoDB entries. Extend tests in each stage to cover the CRUD operations and DynamoDB read/write time. Write API documentation and commit to GitHub. Manage README files and runbooks in markdown on GitHub.

Concepts:

- Learn AWS Services: API Gateway, DynamoDB
- Write a RESTful API Gateway interface for web crawler CRUD operations
- Write a Python Function to implement business logic of CRUD into DynamoDB
- Extend tests and prod/beta Cl/CD pipelines in CodeDeploy / CodePipeline
- Use Cl/CD to automate multiple deployment stages (prod vs beta)

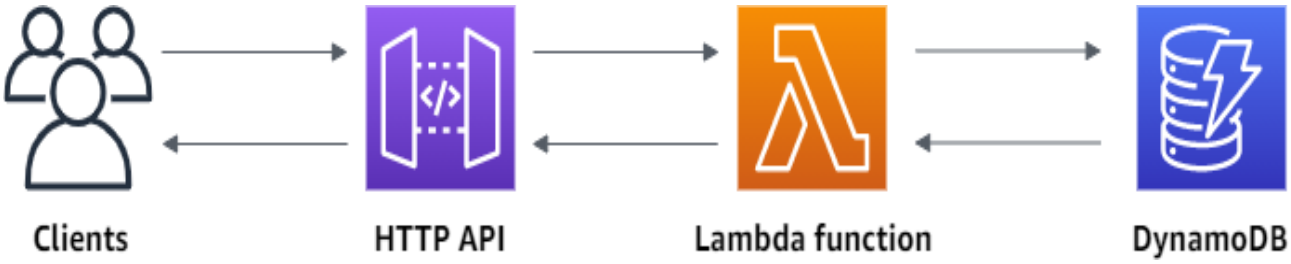
Amazon API Gateway:

Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. As an API Gateway API developer, you can create APIs for use in your own client applications. Or you can make your APIs available to third-party app developers.

API Gateway creates RESTful APIs that:

- Are HTTP-based.
- Enable stateless client-server communication.
- Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

Flow Diagram



Task 1: Update DynamoDB with S3 urls.json file.

Create a table for URLS and then loop through links retreived from bucket and pu them in table using boto3 client.put_item function.

```
AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def lambda_handler(events, context):
    client = boto3.client('dynamodb')


    URLS= s('shanawarbucket','urls.json').get_bucket()
    print('URLS in API LAMBDA ',URLS)

    urltable = os.getenv(key = 'table_name')#getting table name
    print('THE URL TABLE NAME in API LAMBDA:',urltable)

    for link in URLS:
        client.put_item(TableName = urltable,Item={'Links':{'S': link}})
```

Task 2: Build API gateway and add CRUD methods.

```
    ##########################   API ###################################
    myapi=apigateway.LambdaRestApi(self,"SHANAWAR_ALI_API",handler=apilambda)
    apilambda.add_environment(key = 'table_name', value = urls_table.table_name)

    ########################## creating API gateway ###################
    apilambda.grant_invoke( aws_iam.ServicePrincipal("apigateway.amazonaws.com"))
    urls_table.grant_read_write_data(apilambda)

    items = myapi.root.add_resource("items")
#     Allowed methods: ANY,OPTIONS,GET,PUT,POST,DELETE,PATCH,HEAD POST /items

  # CRUD OPERATIONS
    items.add_method("PUT") # CREATE: ADD URL TO TABLE
    items.add_method("GET") # READ: GET ALL URLS FROM TABLE
    items.add_method("POST") # UPDATE: UPDATE URL IN TABLE
    items.add_method("DELETE") # DELETE: DELETE URL FROM TABLE

    ###########################################################################################
```

Task 3: Create CRUD operations in backend API Lambda

- Retrieve method and body from API Gateway.
- Perform operations according to retrieved methods and urls.

```
method = events['httpMethod']
if method == 'GET':
    data = read.ReadFromTable(urltable)
    response = f"URLS = {data} "

elif method == 'PUT':
    newurl = events['body']
    client.put_item(
    TableName = urltable,
    Item={
    'Links':{'S' : newurl},
    })
    response = f"Url = {events['body']} is successfully added into the table"

elif method == 'DELETE':
    url = events['body']
    print(url)
    client.delete_item(
    TableName =urltable,
    Key={
    'Links':{'S' : url}
    })
    response = f"Url= {events['body']} is successfully deleted from the table"

elif method == 'POST':
    url = events['body']
    url_ex_new=url.split(",")
    ex=url_ex_new[0]
    new=url_ex_new[1]
    URLS_LIST=read.ReadFromTable(urltable)  #read table
    if ex in URLS_LIST:                  #if item is avaialble then
        client.delete_item(TableName= urltable,Key={'Links':{'S' : ex}})
        client.put_item(TableName= urltable,Item={'Links':{'S' : new}})
        response="Successfully updated in DynamoDB table."
    else:
        response="Failed to update"
```

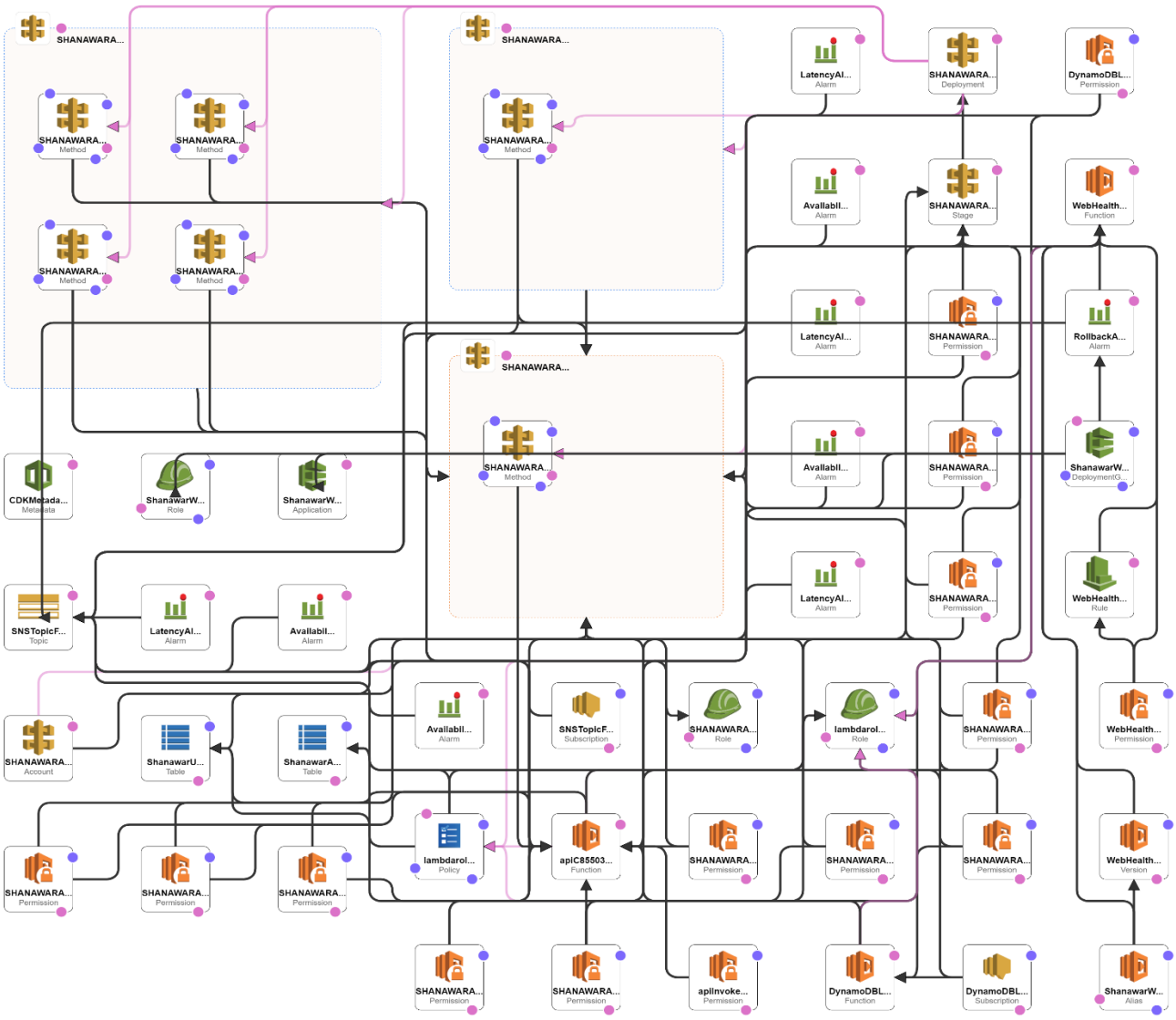
Task 4: Create automated testing in pipeline

```python
import pytest ,urllib3,requests,datetime

api='https://xx7b4z0m61.execute-api.us-east-2.amazonaws.com/prod'


AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def test_put():
    http=urllib3.PoolManager()
    response=http.request("GET",api)
    assert response.status == 200


AWS: Add Debug Configuration | AWS: Edit Debug Configuration
def test_latency():
    http=urllib3.PoolManager()
    start=datetime.datetime.now()
    response=http.request("GET",api)
    end=datetime.datetime.now()
    diff=end-start
    latency_sec=round(diff.microseconds * 0.000001,6)
    assert latency_sec<1
```

Designer Templates:

AWS CloudFormation Designer (Designer) is a graphic tool for creating, viewing, and modifying AWS CloudFormation templates. With Designer, you can diagram your template resources using a drag-and-drop interface, and then edit their details using the integrated JSON and YAML editor. Whether you are a new or an experienced AWS CloudFormation user, AWS CloudFormation Designer can help you quickly see the interrelationship between a template's resources and easily modify templates.

Production Stage Cloud Template:



Outputs:

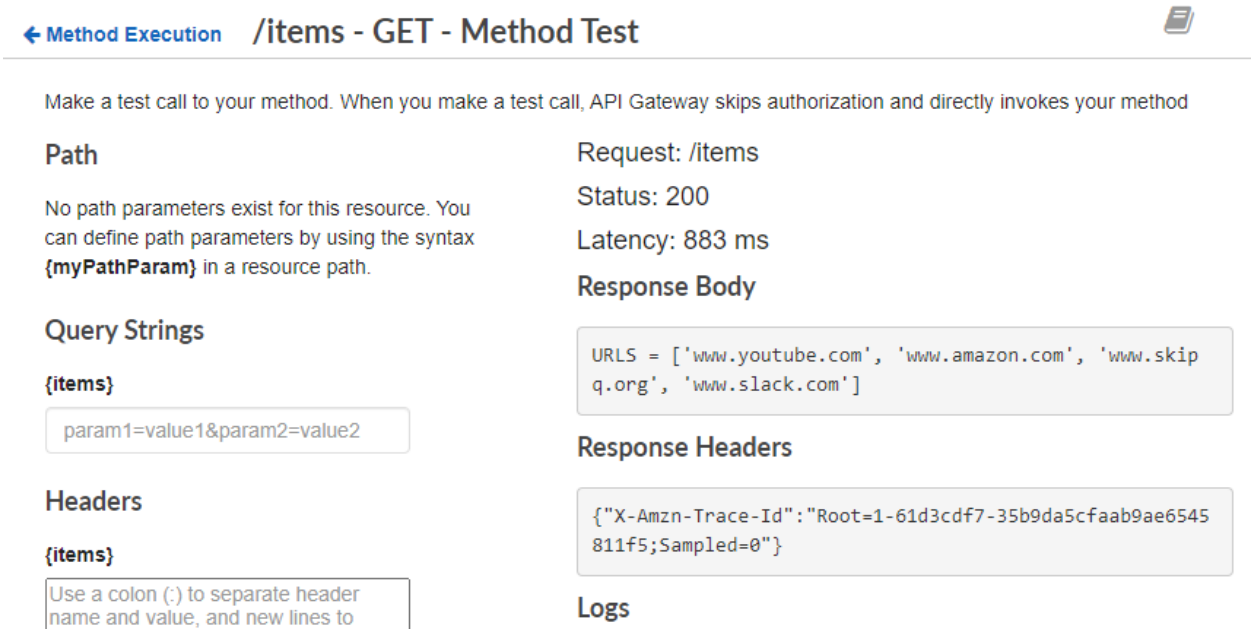
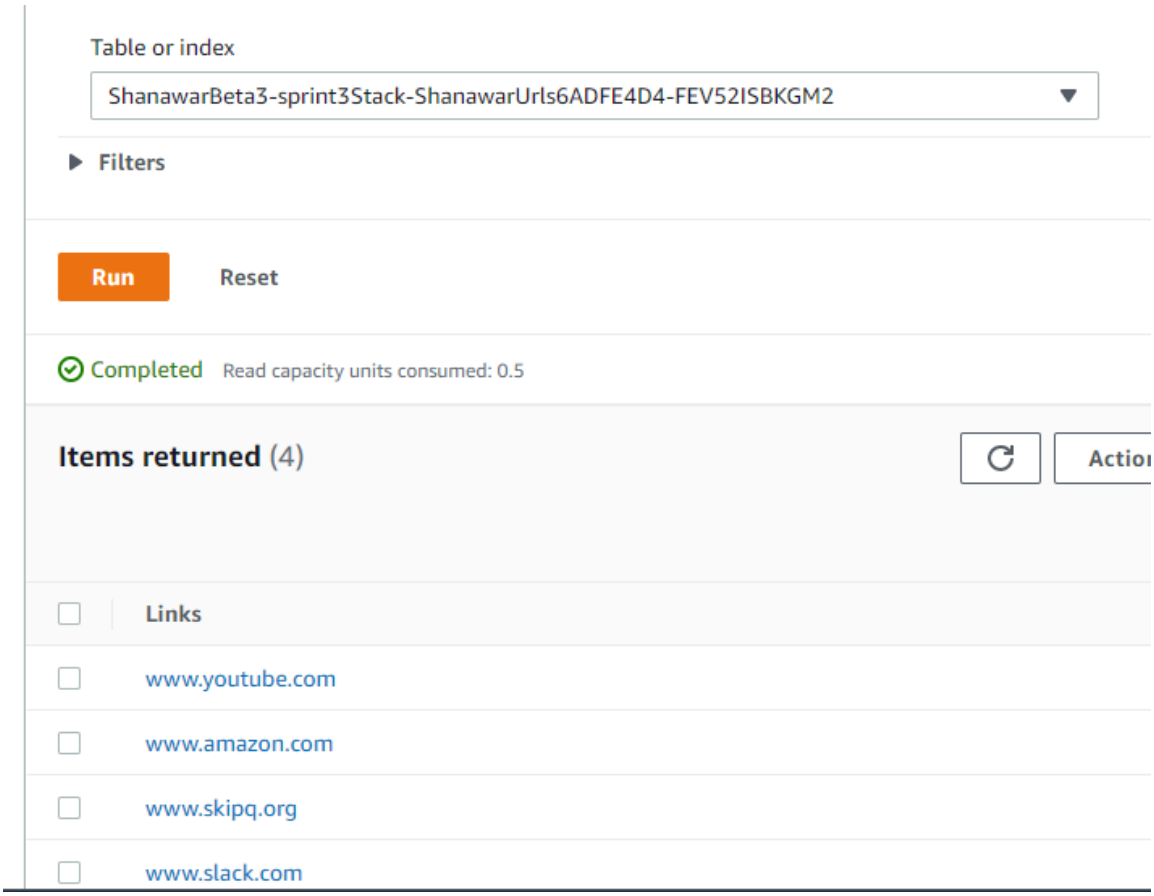


Table:



Issues Faced:

Insufficient data for alarms:

☐	ShanawarBeta3-sprint2Stack-AvailabilityAlarmwwwslackcomD10F51E5-1V8UKBU34L6O1	OK	2022-01-03 04:38:25	url_availabilitywww.slack.com < 1 for 1 datapoints within 1 minute
☐	ShanawarBeta3-sprint2Stack-AvailabilityAlarmwwwyoutubecom64A57ED7-O0PTSP9GJRO1	OK	2022-01-03 04:38:03	url_availabilitywww.youtube.com < 1 for 1 datapoints within 1 minute
☐	ShanawarBeta3-sprint2Stack-LatencyAlarmwwwfacebookcom128E5193-19559FESQ27N7	Insufficient data	2022-01-03 04:35:19	url_latencywww.facebook.com > 0.25 for 1 datapoints within 1 minute
☐	ShanawarBeta3-sprint2Stack-LatencyAlarmwwwyoutubecom87EF138	Insufficient data	2022-01-03 04:35:18	url_latencywww.youtube.com > 0.25 for 1 datapoints within 1 minute

- Timeout default for lambda is 3sec. Add timeout in lambda of 5 minutes.

S3 Bucket Lambda Key Error:

▼ 2022-01-02T13:01:18.159+05:00 's3': KeyError Traceback (most recent call last):

```
's3': KeyError
Traceback (most recent call last):
  File "/var/task/s3_dynamo_lambda.py", line 11, in lambda_handler
    BucketName = event['Records'][0]['s3']['bucket']['name']
KeyError: 's3'
```

- Lambda is not being triggered by S3 bucket creation. Adding event source for S3 bucket creation solved the issue.

Internal Server Error:

Request: /items

Status: 502

Latency: 510 ms

Response Body

```
{
  "message": "Internal server error"
}
```

- Internal server error was happening in API Gateway methods. It was because of wrong syntax for getting body requests in API. Fixing the syntax solves the issue.

GitHub push issue:

```
shanawaraliskipq:~/environment/ProximaCentauri-1 (main) $ git push
Username for 'https://github.com/shanawar2021skipq/ProximaCentauri-1.git': shanawar2021skipq
Password for 'https://shanawar2021skipq@github.com/shanawar2021skipq/ProximaCentauri-1.git':
To https://github.com/shanawar2021skipq/ProximaCentauri-1.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/shanawar2021skipq/ProximaCentauri-1.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
shanawaraliskipq:~/environment/ProximaCentauri-1 (main) $ git pull
```

- Git pull solved the issue.

References:

1. https://aws.amazon.com/premiumsupport/knowledge-center/lambda-sns-ses-dynamodb/
2. https://docs.aws.amazon.com/cdk/api/latest/docs/aws-construct-library.html
3. https://searchaws.techtarget.com/definition/CloudWatch
4. https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-troubleshooting.html
5. https://docs.aws.amazon.com/awssupport/latest/user/troubleshooting.html
6. https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html
7. https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_apigateway.html