

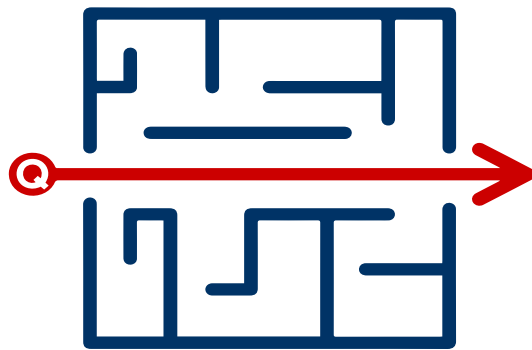
Production Grade Programming

AWS DevOps

Documentation

By

Shanawar Ali Chouhdry



SkipQ

In Partial Fulfillment

Of the Requirements for the graduation

from SKIPQ

Table of Contents

Sprint 1.....	5
Description.....	5
Concepts:	5
AWS:	5
AWS-CDK:	5
AWS-SDK:.....	5
Flow Diagram:.....	6
Implementation	6
Task 1: Create Hello Lambda Function	6
Task 2: Monitor Web Health Metrics on CloudWatch	7
Task 3: Create SNS Subscription	9
Task 4: Updating DynamoDB Table	9
Outputs:.....	10
Email Subscription:	10
CloudWatch Dashboard:.....	11
DynamoDB Table:	12
Issues and Troubleshooting.....	13
Updating python:.....	13
Update DynamoDB	13
SNS Error:.....	13
Sprint 2.....	14
Description:.....	14
Concepts:	14
Designer Template:.....	14
Pipeline Toolkit Stack:	14
Pipeline Stack:	15
Production Stage Stack:.....	16
Implementation:	17
Task 1: Building Pipeline	17

Task 2: Create Rollback Metric and Alarm	20
Task 3: Canary Deployment	20
Outputs:	23
Pipeline:	23
Rollback Alarm:	24
Issues and Troubleshooting:	25
CDK Bootstrap:	25
Rollback Complete Error:	25
Stage Deploying Failed:	26
Role Access Error	26
Sprint 3:	27
Description:	27
Concepts:	27
Amazon API Gateway:	27
Flow Diagram	28
Implementation	28
Task 1: Update DynamoDB with S3 urls.json file	28
Task 2: Build API gateway and add CRUD methods.	28
Task 3: Create CRUD operations in backend API Lambda	29
Task 4: Create automated testing in the pipeline	29
Outputs:	30
API Gateway:	30
DynamoDB Table:	30
Issues and Troubleshooting:	31
Insufficient data for alarms:	31
S3 Bucket Lambda Key Error:	31
Internal Server Error:	32
GitHub push issue:	32
Sprint 4	33
Description:	33
Concepts:	33

ReactJS:.....	33
Chakra UI:	33
Amazon Cognito:	33
Implementation:	34
Task 1: Build a UI for a web crawler	34
Task 2: Fetch Data of URLs from API	34
Task 3: Add AWS Cognito User Pools	35
Output:	36
Login	36
User Interface:	36
Issues and Troubleshooting:	37
CORS error:	37
Sprint 5.....	38
Description.....	38
Concepts	38
Docker:	38
ECS:	38
ECR:.....	39
Implementation:	39
Task 1: Build a Docker image for API Test Client	40
Task 2: Publish Images to ECR	40
Task 3: Deploy images from ECR to ECS	41
Outputs	42
API Test Results:	42
ECS:	43
Issues and Troubleshooting:	43
S3 Bucket Limit:	43
Pipeline Deployment Stuck:	43
References:	44

Sprint 1

Description

Create a web health monitoring application that triggers alarms when thresholds are breached and sends notification on email.

Concepts:

- Introduction to the DevOps Engineer Role and Infrastructure-as-Code (IaC)
- Foundational services (EC2, S3), Microservice architecture
- Introduction to the Art of Monitoring Web Applications
- Learn AWS Services: IAM, Lambda, CloudWatch, SNS, SQS
- Learn Tools: Shell and Scripting, Vim, GitHub

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.

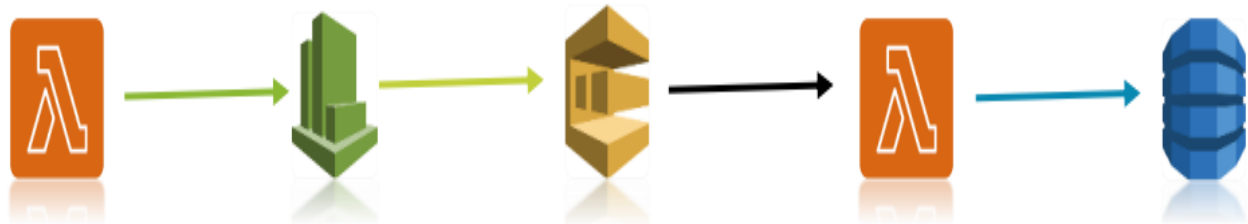
AWS-CDK:

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.

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.

Implementation

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.

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.

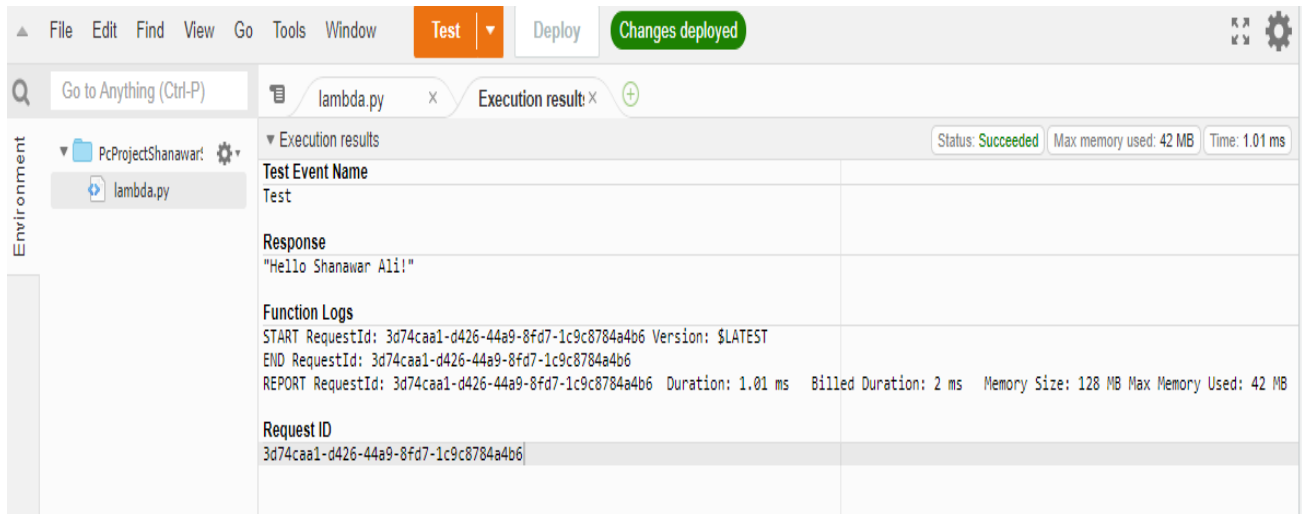
To invoke a function, create a lambda handler function

```

AWS: Add Debug Configuration | AWS: Edit Debug Configuration
1 def lambda_handler(event, context):
2     return 'Hello {} {}'.format(event['first_name'], event['last_name'])

```

Test Results:



The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test', and 'Deploy'. The 'Test' tab is active, and a green button labeled 'Changes deployed' is visible. Below the tabs, there is a search bar and a list of environments. The 'Environment' section shows a folder named 'PcProjectShanawar' with a sub-item 'lambda.py'. The 'Execution results' section shows a table with the following data:

Test Event Name	Response	Function Logs	Request ID
Test	"Hello Shanawar Ali!"	START RequestId: 3d74caa1-d426-44a9-8fd7-1c9c8784a4b6 Version: \$LATEST END RequestId: 3d74caa1-d426-44a9-8fd7-1c9c8784a4b6 REPORT RequestId: 3d74caa1-d426-44a9-8fd7-1c9c8784a4b6 Duration: 1.01 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 42 MB	3d74caa1-d426-44a9-8fd7-1c9c8784a4b6

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. Datetime library in python is used to calculate time difference.

```

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.

```

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
                }
            ]
        )

```


Task 3: Create SNS Subscription

Amazon Simple Notification Service (Amazon SNS) is a managed service that provides message delivery from publishers to subscribers. Publishers communicate asynchronously with subscribers by sending messages to a *topic*, which is a logical access point and communication channel. Clients can subscribe to the SNS topic and receive published messages using a supported endpoint type, such as Amazon Kinesis Data Firehose, Amazon SQS, AWS Lambda, HTTP, email, mobile push notifications, and mobile text messages (SMS).

```
topic =sns.Topic(self,"Web Health Emailing by Shanawar")
self.snssubscriptions(topic,constants.EMAIL,DBLambda)

#####
def snssubscriptions(self,topic,email,dblambd):
    topic.add_subscription(subscriptions_.EmailSubscription(email))
    topic.add_subscription(subscriptions_.LambdaSubscription(dblambda))
#####
```

Task 4: 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(Table_name=table_name,Item={
        'MessageID':{'S':Message},
        'TimeStamp':{'S':Timestamp}
    })
```

Outputs:

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](#).

After subscribing to SNS notifications, you will start receiving emails whenever alarms get triggered.



AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾

9:11 AM (6 minutes ago) ☆ ↶ ⋮

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

...

CloudWatch Dashboard:

Web Health Metrics Shanawar

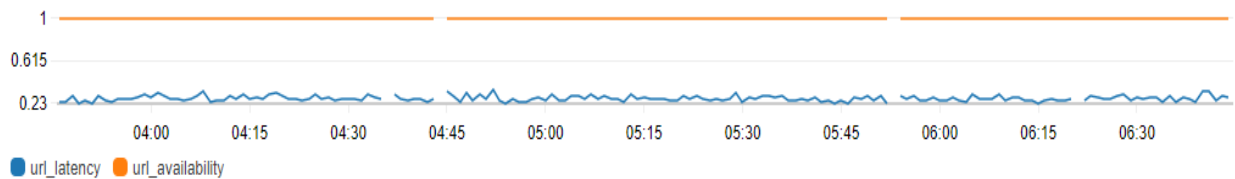
1h 3h 12h 1d 3d 1w custom ▾

Line ▾

Actions ▾



No unit



All metrics

Graphed metrics (2)

Graph options

Source

Math expression ▾ ?		Dynamic labels ▾ ?		Statistic: Average ▾		Period: 10 Seconds ▾	Remove all	
<input checked="" type="checkbox"/>	Label	Details		Statistic	Period	Y Axis	Actions	
<input checked="" type="checkbox"/>	url_latency	Shanawar_WebHealth • url_latency • Region: DU...		Average	10 Seconds			
<input checked="" type="checkbox"/>	url_availability	Shanawar_WebHealth • url_availability • Region: ...		Average	10 Seconds			

Alarms

PcProjectShanawarStack-LatencyAlarm5394FC57-1SC63GNWC478G



Actions ▼

url_latency

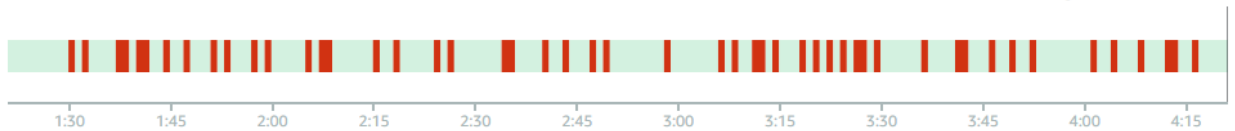
OK

url_latency > 0.28 for 1 datapoints within 1 minute

No unit



Click timeline to see the state change at the selected time.



DynamoDB Table:

ShanawarAlarmTable

☐ SkipqProjectStack-DynamoTableB2B22E15-TLRSLGXJENL2☐ TalhaAlarmTable☐ alarmtable☐ ddbTable☐ shadaab_try_table

Retrieve next page.

Retrieve next page

Items returned (50)



Actions ▼

Create item

< 1 ... > ⚙️

<input type="checkbox"/>	MessageID ▼	TimeStamp ▼
<input type="checkbox"/>	f1911f96-1...	2021-12-19T21:55:57.365Z
<input type="checkbox"/>	318a6676-...	2021-12-19T21:44:57.364Z
<input type="checkbox"/>	d026bea0-...	2021-12-19T21:08:57.415Z
<input type="checkbox"/>	7e19810b-...	2021-12-19T23:52:57.372Z
<input type="checkbox"/>	8612bba9-...	2021-12-20T00:10:57.339Z

Issues and Troubleshooting

Updating python: Python 2.7 to python3 update error. Solving using bashrc file but alias python = python3

Update DynamoDB: Unable to update DynamoDB table. Grant read write access solves the problem. Adding environment variable helped access table name in lambda handler function.

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

SNS Error: Unable to receive notifications. Adding policies to lambda role solved it.

```
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
```

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 CI/CD
- Learn AWS services: CodePipeline and CodeDeploy
- 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

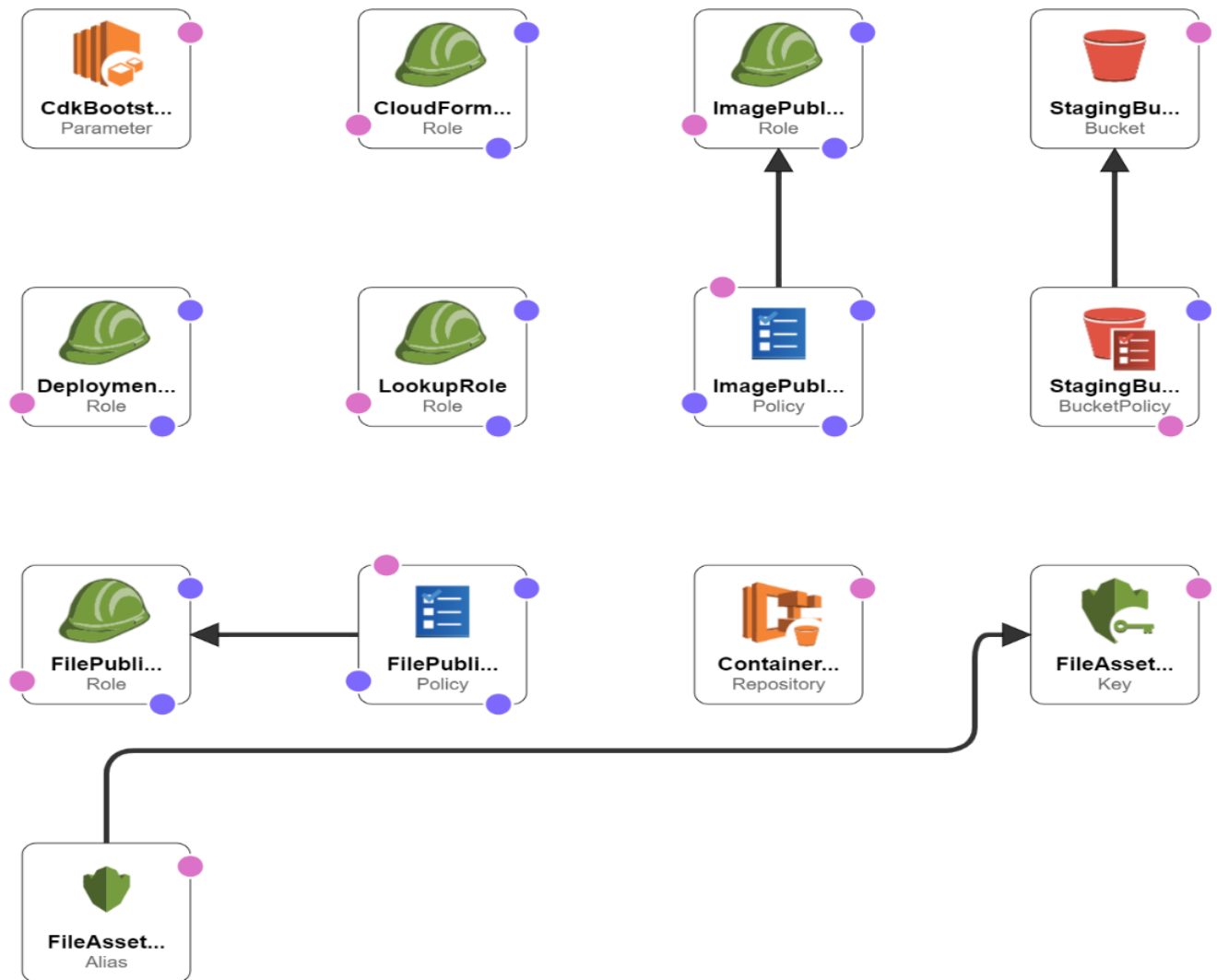
Designer Template:

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

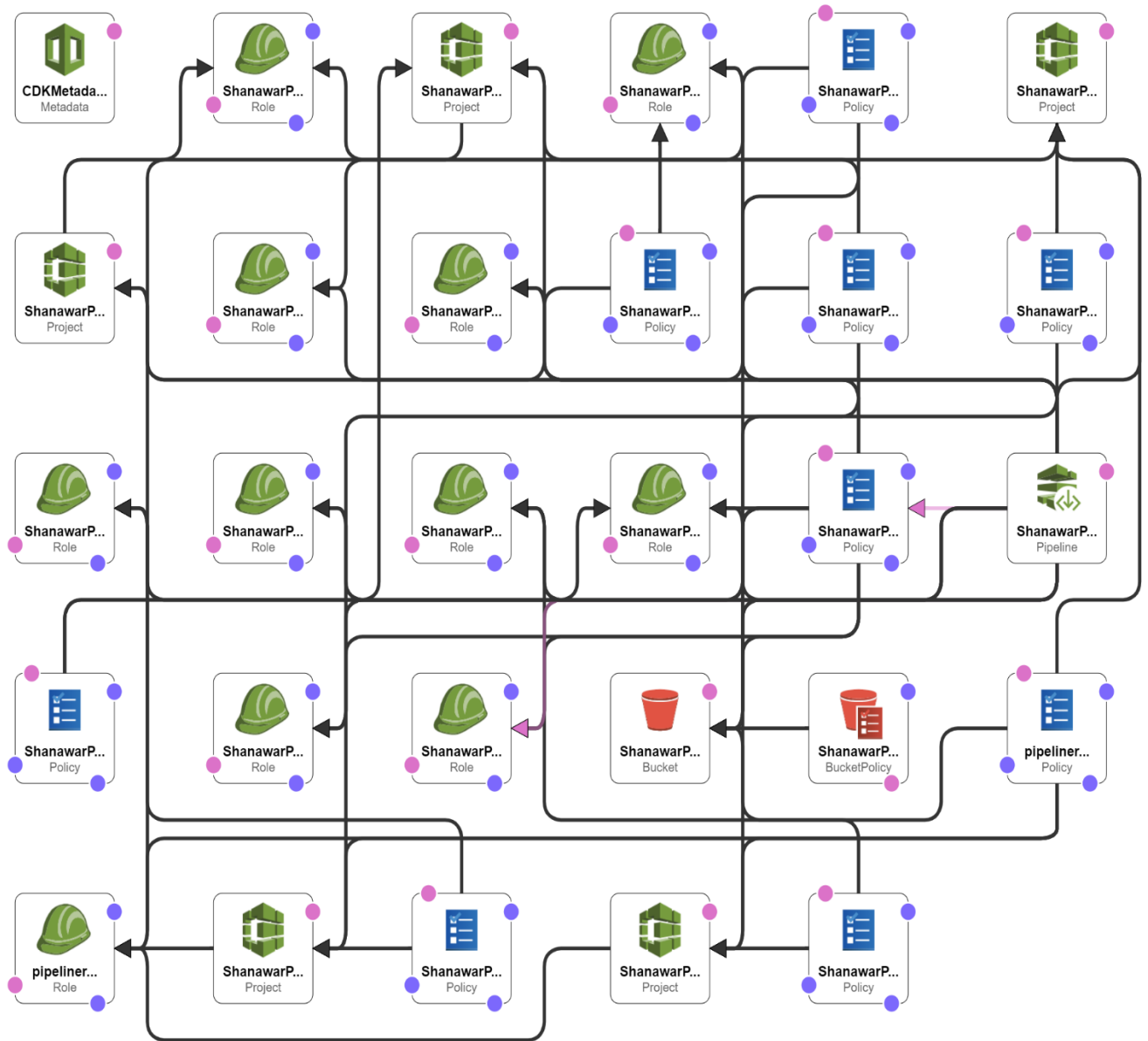
roles that grant permissions needed to perform deployments. The process of provisioning these initial resources is called bootstrapping.



Pipeline Stack:

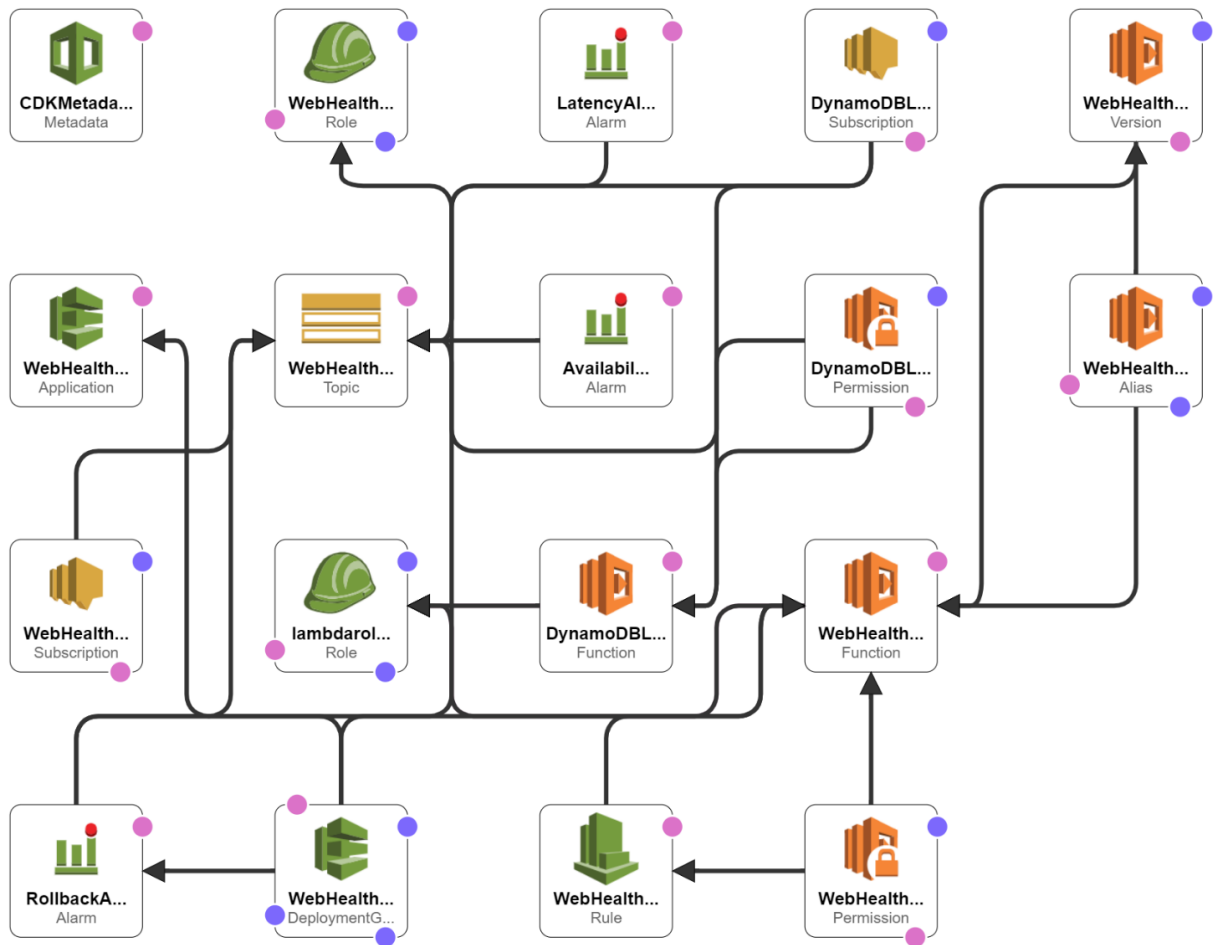
The pipeline stack determines the flow structure of the pipeline and self-updates actions whenever new code is pushed to the source repository. It contains resources like :

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



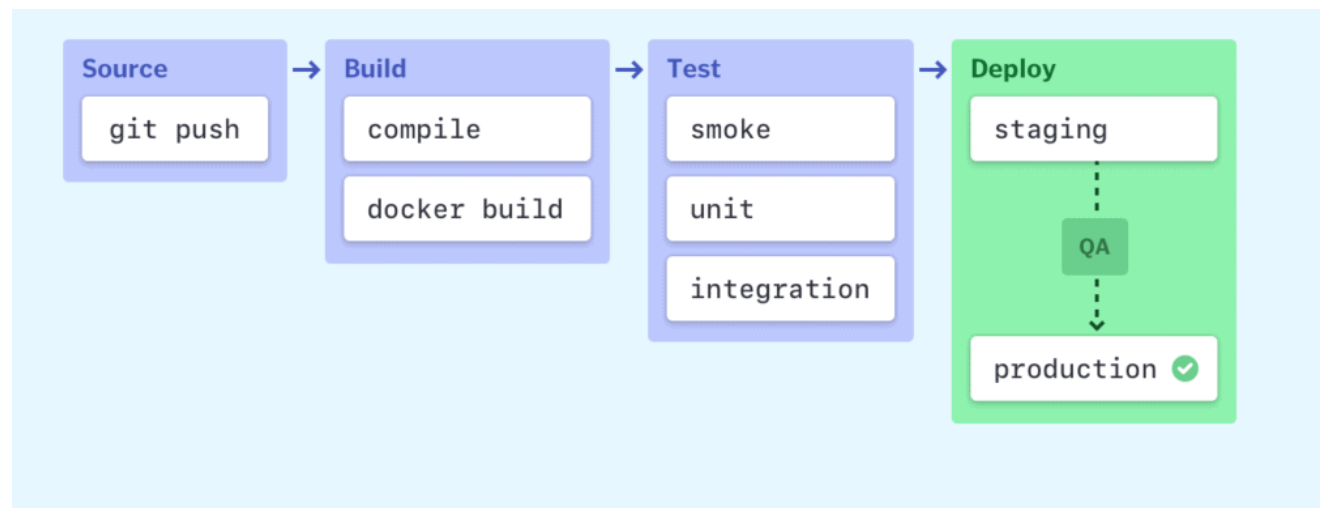
Production Stage Stack:

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



Implementation:

Task 1: 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.

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 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.

```
##### 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. 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 and in an AWS account.

```
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)
```

Task 2: Create Rollback Metric and Alarm

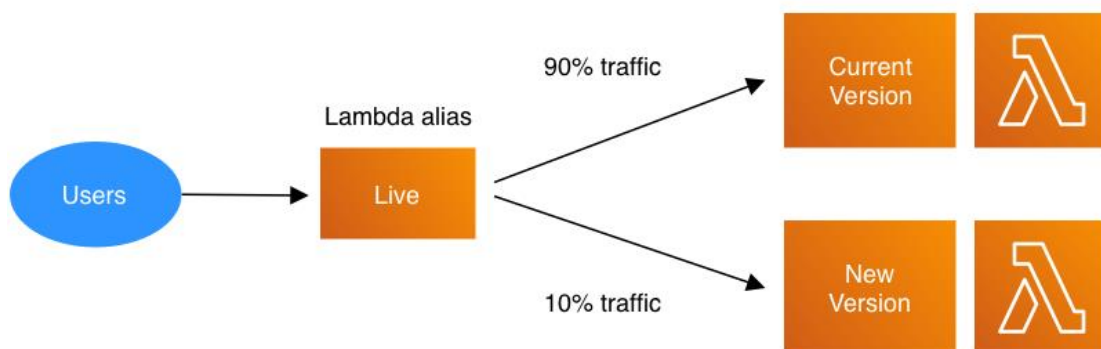
```
# DEFININING ROLLBACK METRIC
rollback_metric=cloudwatch_.Metric(
    namespace='AWS/Lambda',
    metric_name='Duration',
    dimensions_map={'FunctionName':WebHealthLambda.function_name},
    period= cdk.Duration.minutes(1))

# 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))
```

Task 3: Canary Deployment

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[IArm]]) - 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

"""

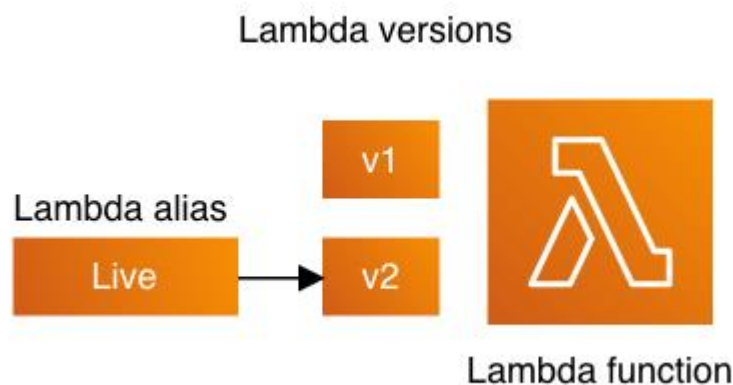
# Linear: Traffic is shifted in equal increments with an equal number of minutes between each increment.
# linear options specify the percentage of traffic that's shifted in each increment and the number of minutes between each increment.

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

```

Lambda versions and aliases

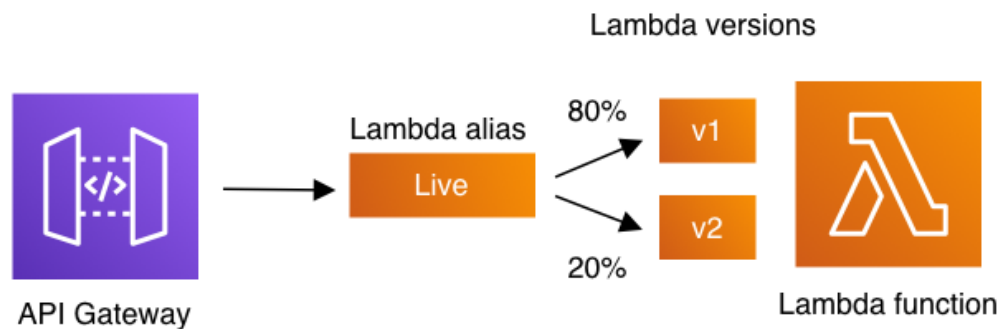
AWS Lambda allows you to publish multiple versions of the same function. Each version has its code and associated dependencies, as well as its function settings (like memory allocation, timeout, and environment variables). You can then refer to a given version by using Lambda Alias. An alias is nothing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



Traffic shifting with Lambda aliases

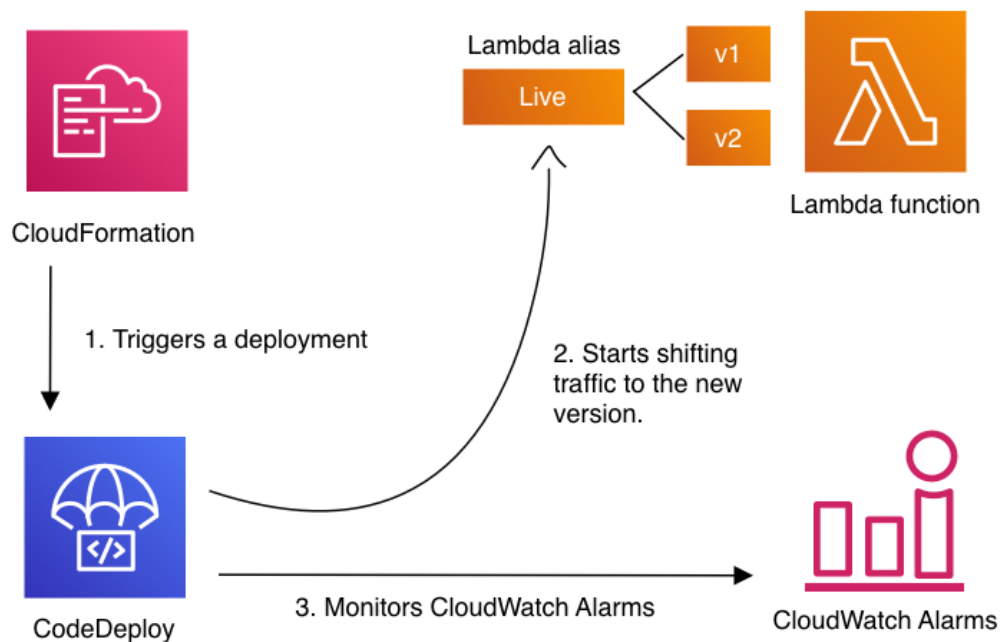
With the introduction of alias traffic shifting, it is now possible to trivially implement canary deployments of Lambda functions. By updating additional version weights on an alias, invocation traffic is routed to the new function versions based on the weight specified.

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 the 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.



Outputs:

Pipeline:

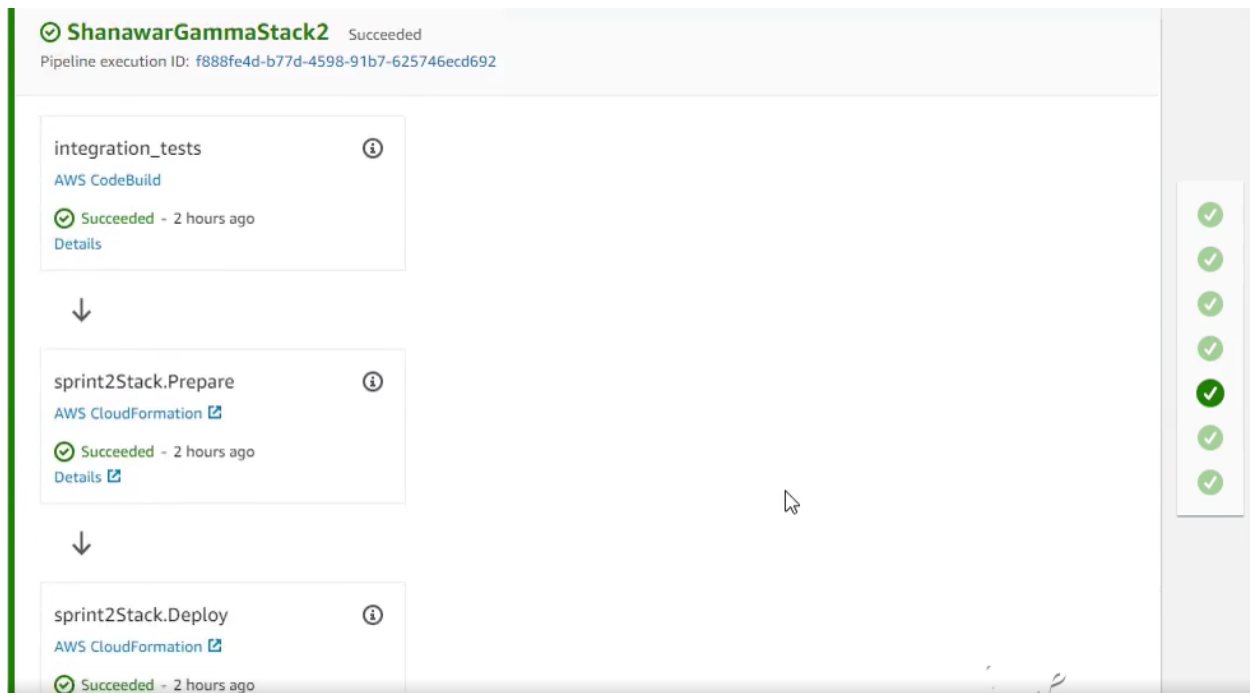
The screenshot displays the AWS CodePipeline console for a pipeline named "ShanawarBetaStack2". The pipeline execution ID is f888fe4d-b77d-4598-91b7-625746ecd692. The pipeline consists of three stages, all of which have succeeded.

Stage 1: Source (Succeeded)
Pipeline execution ID: f888fe4d-b77d-4598-91b7-625746ecd692
Action: shanawar2021skipq_ProximaC... (GitHub (Version 1))
Status: Succeeded - 2 hours ago
58cd19cd
58cd19cd shanawar2021skipq_ProximaCentauri: isabelle

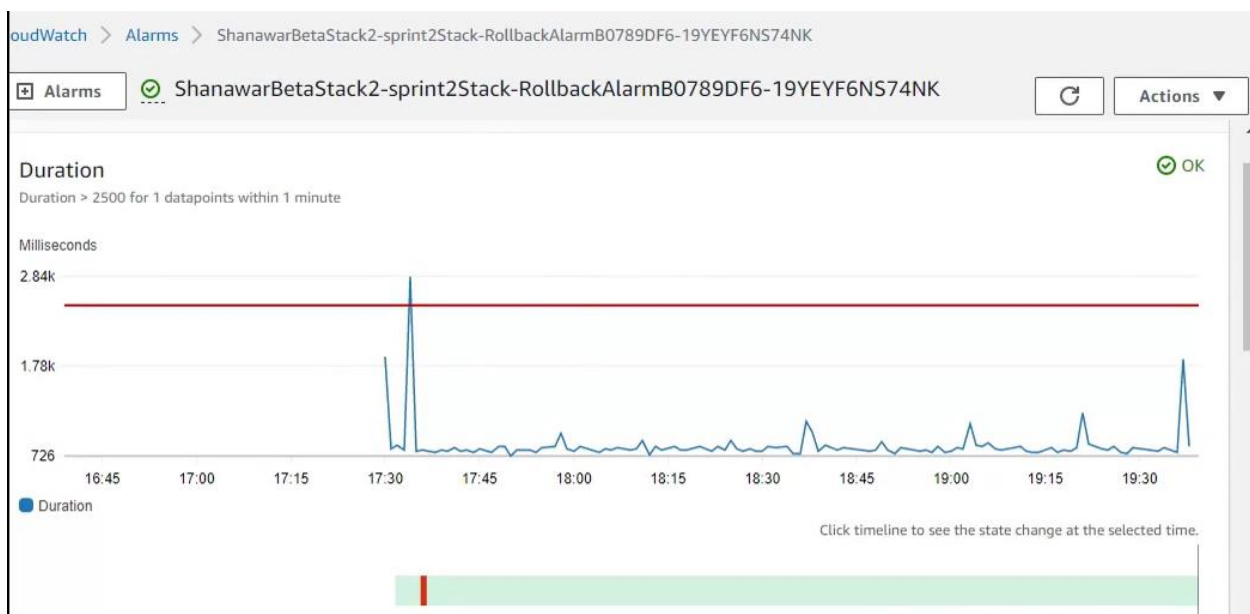
Stage 2: Build (Succeeded)
Pipeline execution ID: f888fe4d-b77d-4598-91b7-625746ecd692
Action: Shanawar_synth (AWS CodeBuild)
Status: Succeeded - 2 hours ago
Details

Stage 3: ShanawarBetaStack2 (Succeeded)
Pipeline execution ID: f888fe4d-b77d-4598-91b7-625746ecd692
Actions:
- unit_tests (AWS CodeBuild)
Status: Succeeded - 2 hours ago
Details
- sprint2Stack.Prepare (AWS CloudFormation)
Status: Succeeded - 2 hours ago
Details
- sprint2Stack.Deploy (AWS CloudFormation)
Status: Succeeded - 2 hours ago
Details

A "Disable transition" button is visible between the Source and Build stages. A vertical bar on the right side of the console shows a series of green checkmarks, indicating the success of each action in the pipeline.



Rollback Alarm:



Issues and Troubleshooting:

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.

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 redeploying solves the problems.

4 Answers

Active

Oldest

Votes



100

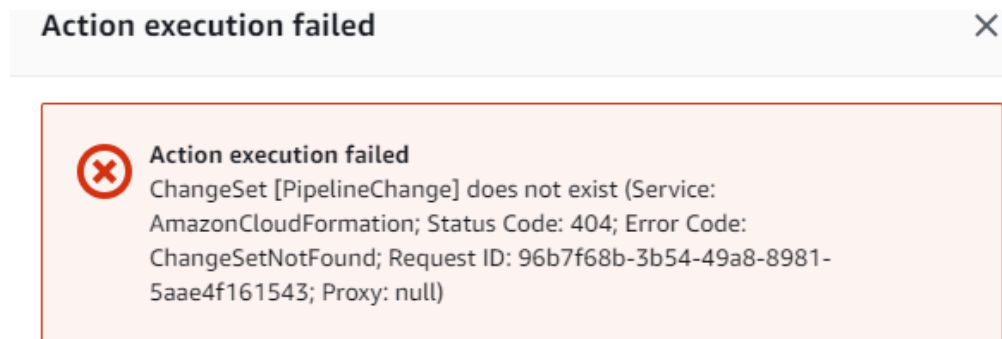


This happens when stack creation fails. By default the stack will remain in place with a status of `ROLLBACK_COMPLETE`. This means it's successfully rolled back (deleted) all the resources which the stack had created. The only thing remaining is the empty stack itself. You cannot update this stack; you must manually delete it, after which you can attempt to deploy it again.



If you set "Rollback on failure" to disabled in the console (or set `--on-failure` to `DO_NOTHING` in the CLI command, if using `create-stack`), stack creation failure will instead result in a status of `CREATE_FAILED`. Any resources created before the point of failure won't have been rolled back.

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)
at AcceptorStateMachine.runTo (/usr/local/lib/node_modules/aws-cdk/node_modules/aws-sdk/lib/state_machine.js:11:12)
```

solution: Create role function with code build service principal 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
```

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.

Concepts:

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

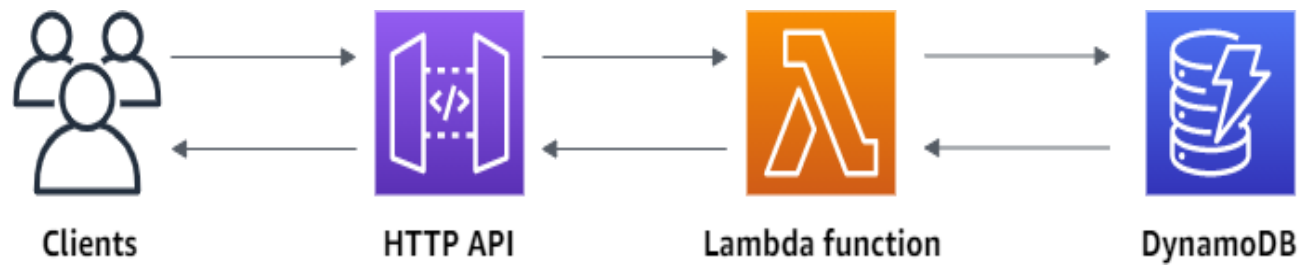
Amazon API Gateway:

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



Implementation

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

Create a table for URLs and then loop through links retrieved from bucket and put them in a 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(Table_name = 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(urldata)
    response = f"URLS = {data} "

elif method == 'PUT':
    newurl = events['body']
    client.put_item(
        TableName = urldata,
        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 =urldata,
        Key={
            'Links':{'S' : url}
        })
    response = f"Url= {events['body']} is successfully deleted from the table"
```

Task 4: Create automated testing in the pipeline

```
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
```

Outputs:

API Gateway:

← Method Execution /items - GET - Method Test



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

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Query Strings

{items}

param1=value1¶m2=value2

Headers

{items}

Use a colon (:) to separate header name and value, and new lines to

Request: /items

Status: 200

Latency: 883 ms

Response Body

```
URLS = ['www.youtube.com', 'www.amazon.com', 'www.skipq.org', 'www.slack.com']
```

Response Headers

```
{"X-Amzn-Trace-Id": "Root=1-61d3cdf7-35b9da5cfaab9ae6545811f5;Sampled=0"}
```

Logs

DynamoDB Table:

Table or index

ShanawarBeta3-sprint3Stack-ShanawarUrls6ADFE4D4-FEV52ISBKGM2

► Filters

Run

Reset

✔ Completed Read capacity units consumed: 0.5

Items returned (4)

↻

Action

<input type="checkbox"/>	Links
<input type="checkbox"/>	www.youtube.com
<input type="checkbox"/>	www.amazon.com
<input type="checkbox"/>	www.skipq.org
<input type="checkbox"/>	www.slack.com

Issues and Troubleshooting:

Insufficient data for alarms:

<input type="checkbox"/>	ShanawarBeta3-sprint2Stack-AvailabilityAlarmwwwslackcomD10F51E5-1V8UKBU34L6O1	🟢 OK	2022-01-03 04:38:25	url_availabilitywww.slack.com < 1 for 1 datapoints within 1 minute
<input type="checkbox"/>	ShanawarBeta3-sprint2Stack-AvailabilityAlarmwwwyoutube.com64A57ED7-O0PTSP9GJRO1	🟢 OK	2022-01-03 04:38:03	url_availabilitywww.youtube.com < 1 for 1 datapoints within 1 minute
<input type="checkbox"/>	ShanawarBeta3-sprint2Stack-LatencyAlarmwwwfacebookcom128E5193-19559FESQ27N7	🟡 Insufficient data	2022-01-03 04:35:19	url_latencywww.facebook.com > 0.25 for 1 datapoints within 1 minute
<input type="checkbox"/>	ShanawarBeta3-sprint2Stack-LatencyAlarmwwwyoutube.com87EF138	🟡 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 an 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 the 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
```

- Updating readme file on GitHub caused the issue that repository contains changes which I don't have locally on my machine. The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.

Sprint 4

Description:

Build a Front-End user interface for the CRUD API Gateway using ReactJS. The user interface should allow users to see and search the database (DynamoDB) and should load URLs with pagination. Login should be enabled through React with authentication using AWS Cognito or equivalent OAuth method. The React app can be rendered with an AWS Lambda Function. Use the library of foundational and advanced components and design systems in Chakra UI to develop your React application

Concepts:

- Learn how to create a Front-End app with ReactJS
- Learn how to enable authentication using the OAuth method
- Write accessible React apps using readily available UI libraries.

ReactJS:

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications

Chakra UI:

Create accessible React apps with speed. Chakra UI is a simple, modular and accessible component library that gives you the building blocks you need to build your React applications.

Amazon Cognito:

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Apple, Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0 and OpenID Connect.

.

Implementation:

Task 1: Build a UI for a web crawler

- 2 Buttons: Search and Get all URLs
- 1 Text Input for search bar
- Output Field

```
return (  
  <>  
  <center>  
    <Heading mb={7} color='royalblue'>Web Crawler</Heading>  
    <Button onClick={()=>setprint1(true)} colorScheme='blue'>Search</Button>  
    <Input placeholder='Search' onChange={(e)=>setSearchData(e.target.value)}/>  
    <Button onClick={()=>setprint(true)} colorScheme='blue'>Show URLs</Button>  
    {  
      print?  
      <> {<ul dangerouslySetInnerHTML={{__html: data}}></ul>} </>  
      :null  
    }  
  </center>  
</>  
)
```

Task 2: Fetch Data of URLs from API

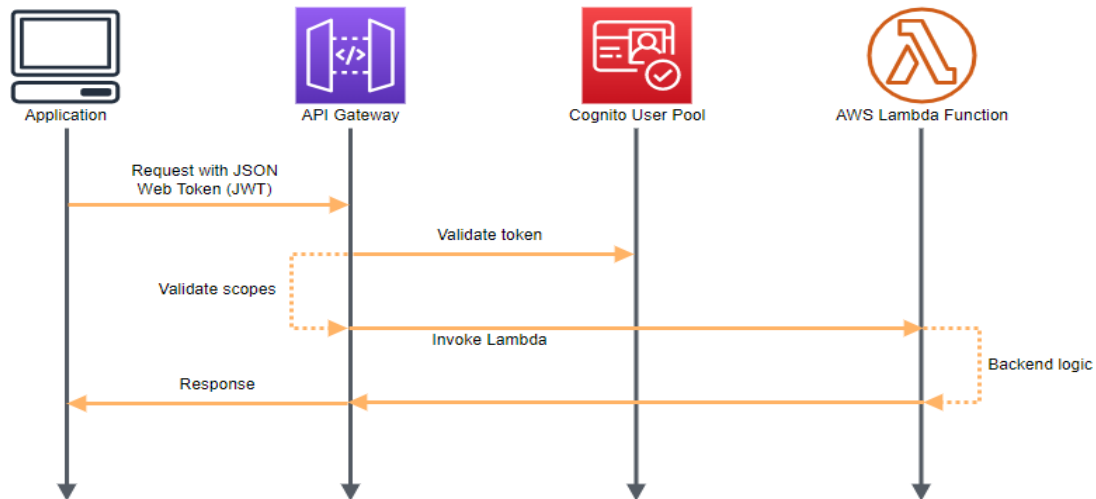
- Axios Get Function returns response of API get function.
- Looping through the output to display separate URLs

```
axios.get(API).then((response) =>  
  {  
    console.log(response.data)  
    var urlsdta = response.data.split('[');  
    var urlsdta = urlsdta[1].split(',');  
    var loopData = ''  
    var i = 0 ;  
    while (i < urlsdta.length-1){  
      loopData += `<li>${urlsdta[i]}</li>`  
      i++;  
    }  
    console.log(loopData);  
    setdata(loopData)  
  });
```

Task 3: Add AWS Cognito User Pools

- Create User pools
- Create pool Clients
- Configure Callback URLs

A diagram showing how an Amazon Cognito authorization workflow works



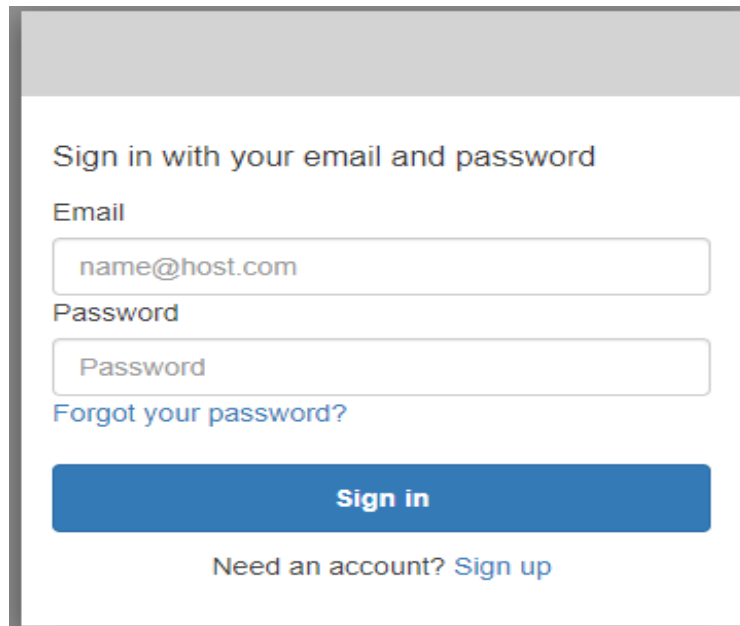
```
##### SPRINT 4 #####
##### Cognito #####
user_pool = aws_cognito.UserPool(self, 'shanawar_UserPool',
    removal_policy=cdk.RemovalPolicy.DESTROY,
    self_sign_up_enabled=True,
    sign_in_aliases={'email': True},
    auto_verify={'email': True},
    password_policy={
        'min_length': 8,
        'require_lowercase': False,
        'require_digits': False,
        'require_uppercase': False,
        'require_symbols': False,
    },
    account_recovery=aws_cognito.AccountRecovery.EMAIL_ONLY
)

user_pool_client = aws_cognito.UserPoolClient(self, 'UserPoolClient',
    user_pool=user_pool,
    auth_flows={
        'admin_user_password': True,
        'user_password': True,
        'custom': True,
        'user_srp': True
    },
    o_auth=aws_cognito.OAuthSettings(
        flows=aws_cognito.OAuthFlows(
            implicit_code_grant=True
        ),
        callback_urls=["http://localhost:3000/"]
    ),
    supported_identity_providers=[aws_cognito.UserPoolClientIdentityProvider.COGNITO]
)

auth = apigateway.CognitoUserPoolsAuthorizer(self, 'AuthorizerForApi', cognito_user_pools=[user_pool])
```

Output:

Login



Sign in with your email and password

Email

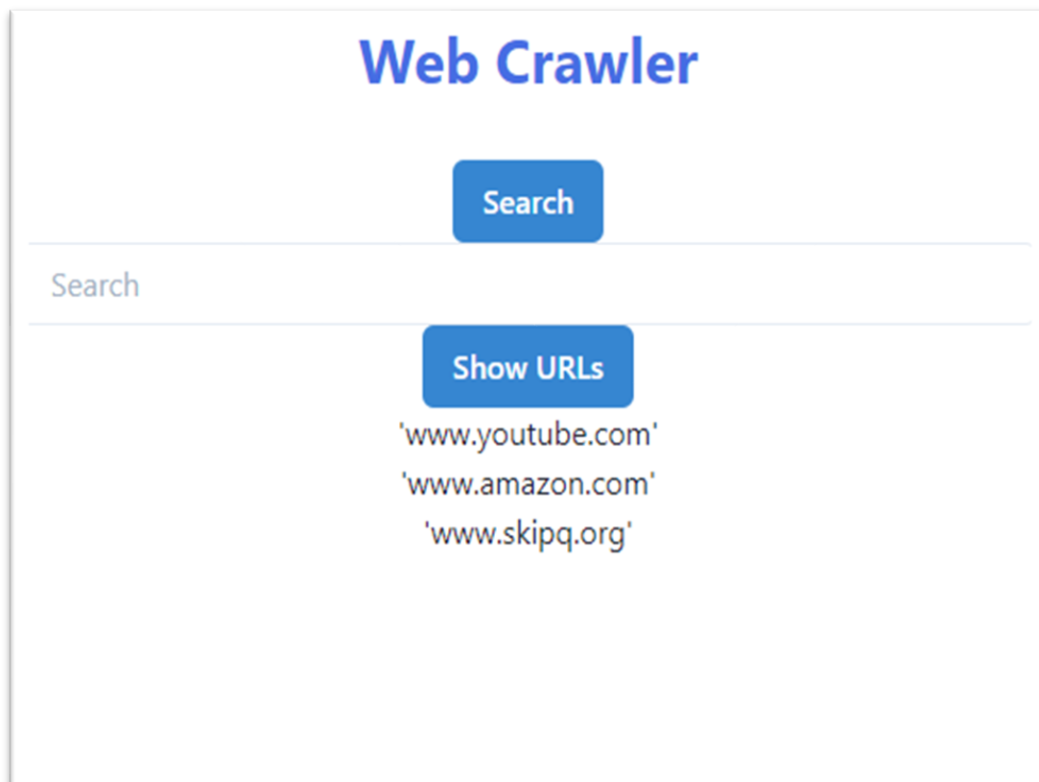
Password

[Forgot your password?](#)

Sign in

Need an account? [Sign up](#)

User Interface:



Web Crawler

Search

Search

Show URLs

'www.youtube.com'

'www.amazon.com'

'www.skipq.org'

Issues and Troubleshooting:

CORS error:

Cross-Origin Resource Sharing (CORS) is an HTTP-header-based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request. For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts. For example, XMLHttpRequest and the Fetch API follow the same-origin policy. This means that a web application using those APIs can only request resources from the same origin the application was loaded from unless the response from other origins includes the right CORS headers.



Solution: Browser Extension solved the issue or you can modify response headers.

```
##### API #####
myapi=apigateway.LambdaRestApi(self,"SHANAWAR_ALI_API",handler=apilambda,
default_cors_preflight_options=apigateway.CorsOptions(
    allow_origins=apigateway.Cors.ALL_ORIGINS,
    allow_methods=apigateway.Cors.ALL_METHODS)
)

return{
    'statusCode' : 200,
    'headers': {
        'Access-Control-Allow-Headers': '*',
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': '*'
    },
    'body' : json.dumps(response)
}
```

Sprint 5

Description

Use docker to build API test clients using pyresttest. These tests will exercise the web crawler's CRUD endpoint built in the previous sprint. Publish built images to Elastic Container Registry (ECR). Deploy API test clients from Sprint 4 on an EC2 instance/ AWS Fargate. Build and push API test dockers through CodePipeline. Push API test results into CloudWatch. Setup alarming and notification on API test metrics.

Concepts

- Learn AWS services: ECR
- Learn docker: DockerFile, Image, Containers. Docker commands [build, start/stop container, deleting container].
- Learn container registries: Working with images (Pull/Push), auto-updates to pull new images once published
- Learn API functional testing framework: pyresttest
- Learn AWS Services: EC2 and AWS Fargate
- Extend tests and prod/beta CI/CD pipelines in CodeDeploy / CodePipeline

Docker:

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run. Running Docker on AWS provides developers and admins a highly reliable, low-cost way to build, ship, and run distributed applications at any scale.

ECS:

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster. Your containers are defined in a task definition that you use to run individual tasks or tasks within a service. In this context, a service is a configuration that enables you to run

and maintain a specified number of tasks simultaneously in a cluster. You can run your tasks and services on a serverless infrastructure that is managed by AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.



Run containers at scale

Amazon ECS makes it easy to use containers as a foundational building block for your applications by eliminating the need for you to install, operate, and scale your own cluster management infrastructure.



Flexible container placement

Amazon ECS lets you schedule long-running applications, services, and batch processes. Amazon ECS maintains application availability and allows you to scale your containers up or down to meet your application's capacity requirements.



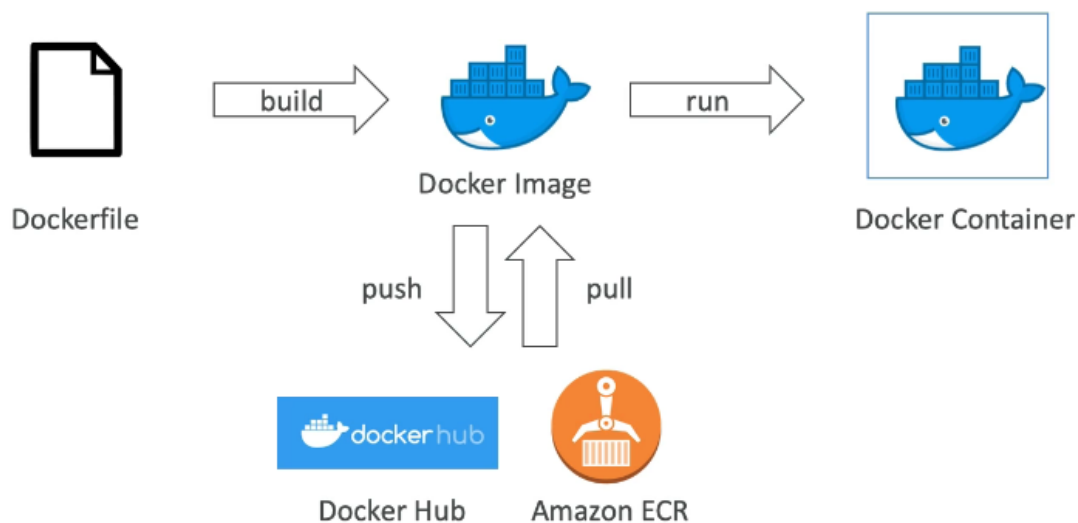
Integrated and extensible

Amazon ECS is integrated with familiar features like Elastic Load Balancing, EBS volumes, VPC, and IAM. Simple APIs let you integrate and use your own schedulers or connect Amazon ECS into your existing software delivery process.

ECR:

Amazon Elastic Container Registry (Amazon ECR) is an AWS-managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM. This is so that specified users or Amazon EC2 instances can access your container repositories and images. You can use your preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

Implementation:



Task 1: Build a Docker image for API Test Client

1. A Dockerfile is simply a text-based script of instructions that is used to create a container image.

```
1 FROM python:2-alpine
2 LABEL Author="Z.d. Peacock <zdp@thoomtech.com>"
3
4 # This needs to be set otherwise pycurl won't link correctly
5 ENV PYCURL_SSL_LIBRARY=openssl
6
7 RUN apk add --no-cache --update openssl curl \
8     && apk add --no-cache --update --virtual .build-deps build-base python-dev curl-dev \
9     && pip install jmespath jsonschema pyresttest \
10    && apk del .build-deps
11
12 WORKDIR /tests
13 COPY . /tests
14
15 ENTRYPOINT ["pyresttest"]
```

2. Build the docker image:

```
docker build -t <docker-name> <directory containing Dockerfile>
```

```
docker run <docker-name>
```

Task 2: Publish Images to ECR

1. Create a Repository in the ECR.
2. Use the following commands to push your image to ECR.

Push commands for shanawardemp



```
word --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/m0u6n8i0
```

Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:

```
docker build -t shanawardemp .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag shanawardemp:latest public.ecr.aws/m0u6n8i0/shanawardemp:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push public.ecr.aws/m0u6n8i0/shanawardemp:latest
```

Close

Task 3: Deploy images from ECR to ECS

1. Create a VPC for ECS Cluster
2. Create an ECS Cluster and select type: EC2/Fargate
3. Create a task definition and add your container
4. Start ECS Service

```
#####
##### SPRINT 5 #####
# vpc = ec2.Vpc(self, "Shanawar_VPC")

vpc = ec2.Vpc.from_lookup(self, "Shanawar_VPC", is_default=True)

# Create an ECS cluster
cluster = ecs.Cluster(self, "ShanawarCluster",
    vpc=vpc
)

cluster.add_capacity("ShanawarClustercapacity", instance_type=ec2.InstanceType("t2.xlarge"))
# Create Task Definition
task_definition = ecs.Ec2TaskDefinition(self, "ShanawarTaskDef")

task_definition.add_container("shanawarContainer",
    image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample"),
    memory_limit_mib=2048,
)

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

Outputs

API Test Results:

Run Command:

```
docker run <image-name> <api> <test.yaml file>
```

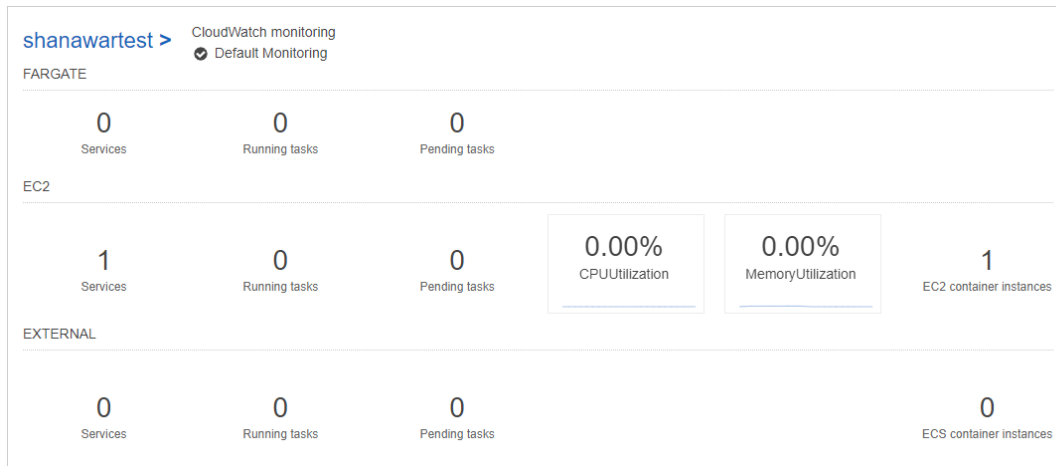
```
(.venv) shanawaraliskipq:~/environment/ProximaCentauri-1/shanawar/sprint5 (main) $ docker run --rm testimage http
s://oob9333t07.execute-api.us-east-2.amazonaws.com/prod api_test.yaml
Test Group Default SUCCEEDED: : 3/3 Tests Passed!
```

If results are successful, it means your docker container is running successfully and your API is functional.

You can also add --log debug in the tail of the command to see logs for testing.

```
docker run <image-name> <api> <test.yaml file> --log debug
```

ECS:



Issues and Troubleshooting:

S3 Bucket Limit:

S3 bucket limit exceeded and the pipeline couldn't be deployed as it has to create pipeline artifacts bucket. Deleting unnecessary buckets solved the issue.

```
10:40:25 AM | CREATE_FAILED | AWS::S3::Bucket | ShanawarPipeline/P...ne/ArtifactsBucket
You have attempted to create more buckets than allowed (Service: Amazon S3; Status Code: 400; Error Code: TooManyBuckets; Request ID: MYA96HFCANN3CVH
P; S3 Extended Request ID: LuW4a/ni+zaPuxFG7H1a1MjhkFiZkJR+cMLVP1L191d4Phn4EnBVLUXQV9j1TOstp17CWFR9M2U=; Proxy: null)
10:40:26 AM | ROLLBACK_IN_PROGRESS | AWS::CloudFormation::Stack | shanawaralipeline
The following resource(s) failed to create: [ShanawarPipelineUpdatePipelineSelfMutateCodePipelineActionRole23F0A419, ShanawarPipelineAssetsFileRole71
594601, ShanawarPipelineShanawarBetaunittestsCodePipelineActionRole6887875B, ShanawarPipelineAssetsFileAsset1CodePipelineActionRole337B4E01, Shanawar
PipelineArtifactsBucket96F8F801, ShanawarPipelineUpdatePipelineSelfMutationRoleEB3C9867, ShanawarPipelineShanawarProdGotoProductionCodePipelineAction
RoleAD525EC9, ShanawarPipelineBuildShanawarsynthesizingCodePipelineActionRoleC11F21E3, ShanawarPipelineRole94177629, CDKMetadata, pipelinerole8D2EF64
[REDACTED] .....] (11/26)

10:40:25 AM | CREATE_FAILED | AWS::S3::Bucket | ShanawarPipeline/P...ne/ArtifactsBucket
You have attempted to create more buckets than allowed (Service: Amazon S3; Status Code: 400; Error Code: TooManyBuckets; Request ID: MYA96HFCANN3CVH
P; S3 Extended Request ID: LuW4a/ni+zaPuxFG7H1a1MjhkFiZkJR+cMLVP1L191d4Phn4EnBVLUXQV9j1TOstp17CWFR9M2U=; Proxy: null)
10:40:26 AM | ROLLBACK_IN_PROGRESS | AWS::CloudFormation::Stack | shanawaralipeline
The following resource(s) failed to create: [ShanawarPipelineUpdatePipelineSelfMutateCodePipelineActionRole23F0A419, ShanawarPipelineAssetsFileRole71
```

Pipeline Deployment Stuck:

The stack couldn't be deployed and it started taking more than 3 hours to deploy and then give update rollback complete error. ECS Service can take a lot of time if the environment is loaded and its capacity is large. Keeping capacity smaller 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://aws.amazon.com/premiumsupport/knowledge-center/iam-assume-role-cli/>
6. <https://cdkworkshop.com/30-python/70-advanced-topics/200-pipelines/1000-setting-up.html>
7. <https://docs.aws.amazon.com/awssupport/latest/user/troubleshooting.html>
8. <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamodb.html>
9. https://docs.aws.amazon.com/cdk/api/v1/python/aws_cdk.aws_apigateway.html
10. <https://www.freecodecamp.org/news/fetch-data-react/>
11. <https://developer.mozilla.org>
12. <https://medium.com/how-to-react/different-ways-to-loop-through-arrays-and-objects-in-react-39bcd870ccf>
13. https://www.youtube.com/watch?v=NEYrSUM4Umw&ab_channel=Codevolution
14. <https://github.com/thoom/pyresttest-docker>
15. https://www.youtube.com/watch?v=mbzxTgpiBL4&ab_channel=FooBarServerless