AWS Cloud Development Kit (CDK) v2

Absolute Beginner to Advanced

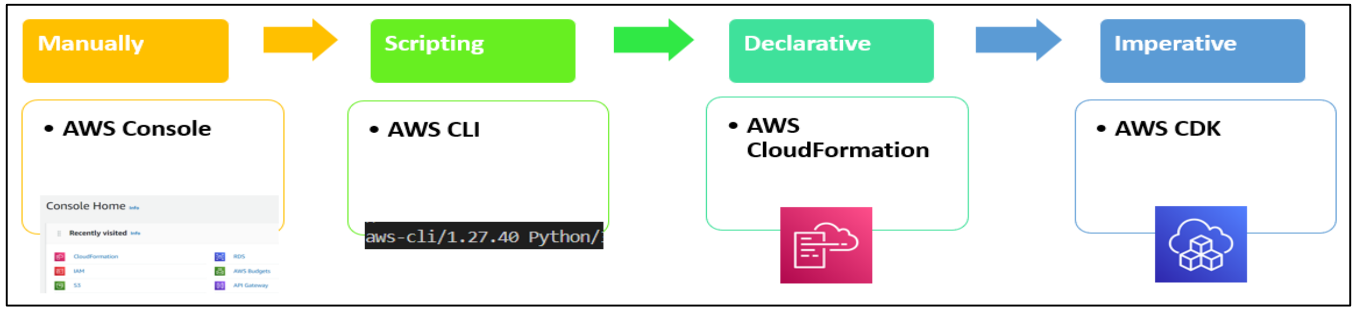
Section 1 and 2 :

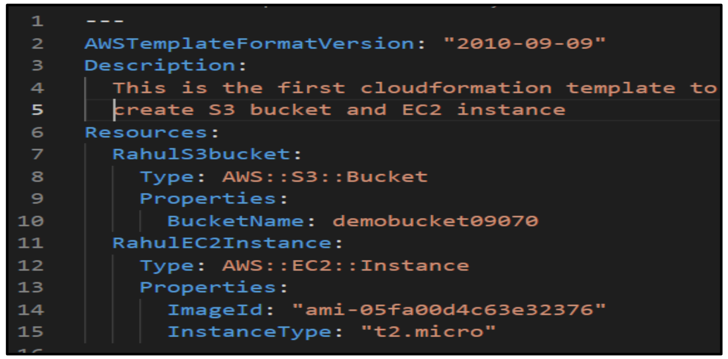
Course Introduction and Setup

AWS CDK v2 – Beginner to Advanced

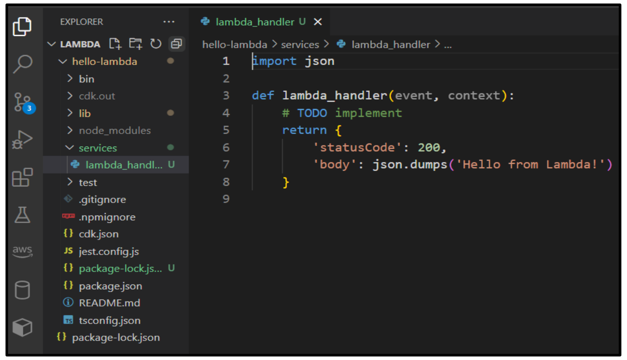
Section 3 : AWS CDK – Basic Concepts

- **Evolution of AWS Infrastructure as Code**
- **AWS CloudFormation – Overview**
- **What is AWS CDK and Benefits**
- **AWS CDK - Basic Concepts**
- **AWS CDK - Project Structure**



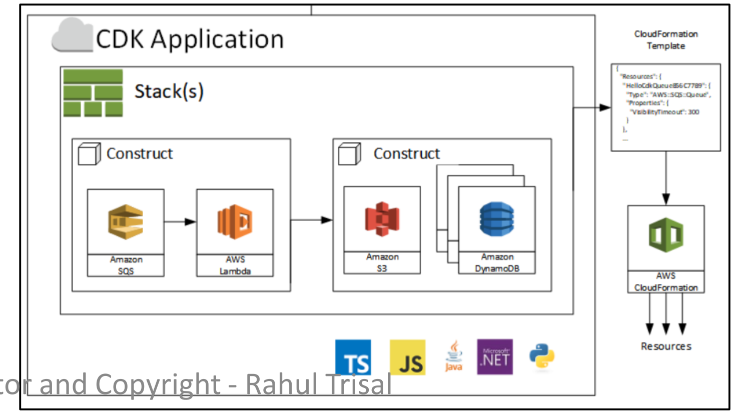


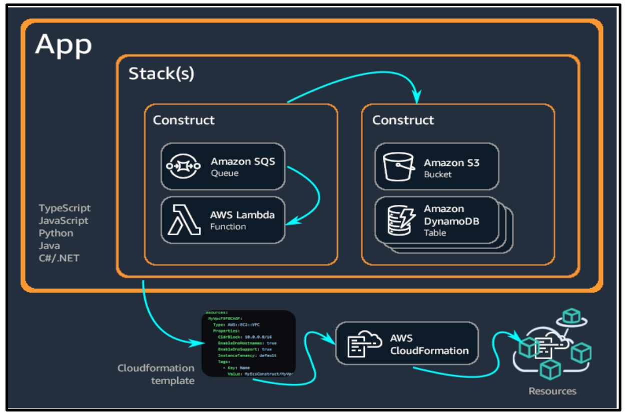
```
---
AWSTemplateFormatVersion: "2010-09-09"
Description:
  This is the first cloudformation template to
  create S3 bucket and EC2 instance
Resources:
  RahulS3bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: demobucket09070
  RahulEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: "ami-05fa00d4c63e32376"
      InstanceType: "t2.micro"
```



```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```





AWS CDK v2 – Beginner to Advanced

Section 4 : Create S3, DynamoDB, Lambda, IAM Role and CloudWatch using AWS CDK v2

Learn to create Individual AWS Services using AWS CDK v2

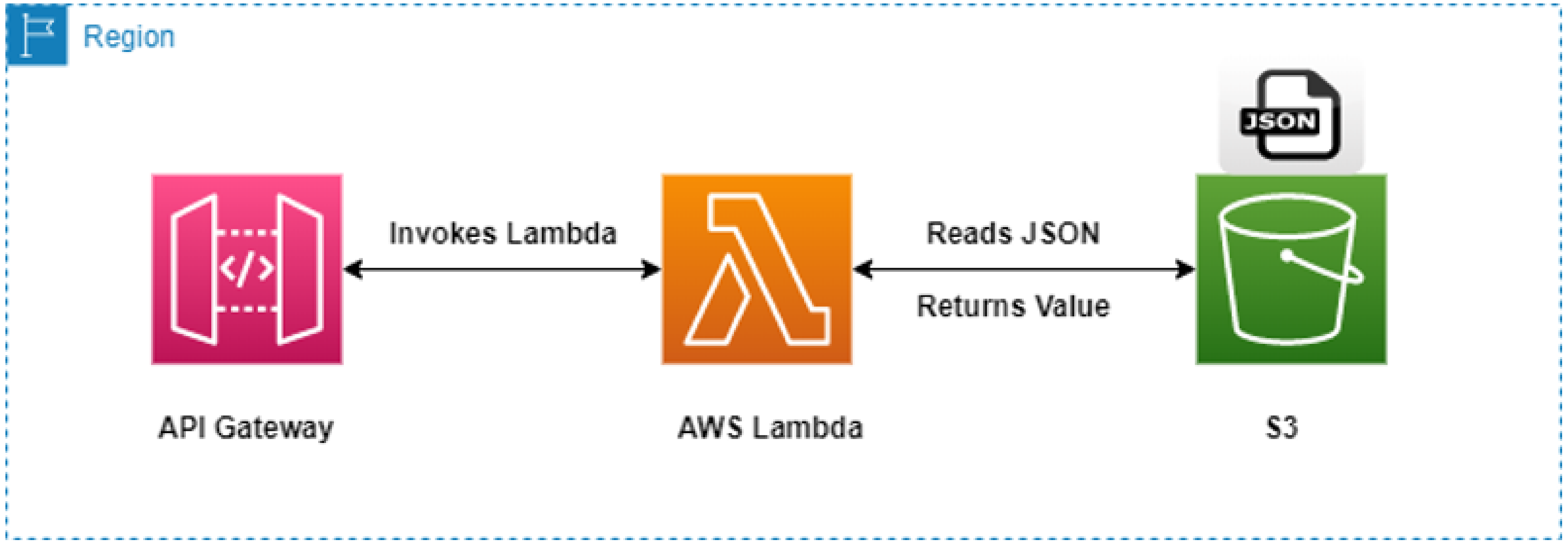
- **S3**
- **DynamoDB**
- **Lambda**
- **IAM**
- **CloudWatch**

AWS CDK

CloudWatch **IAM** **Lambda** **DynamoDB** **S3**

AWS Cloud Development Kit (CDK) v2 – Serverless Use Case

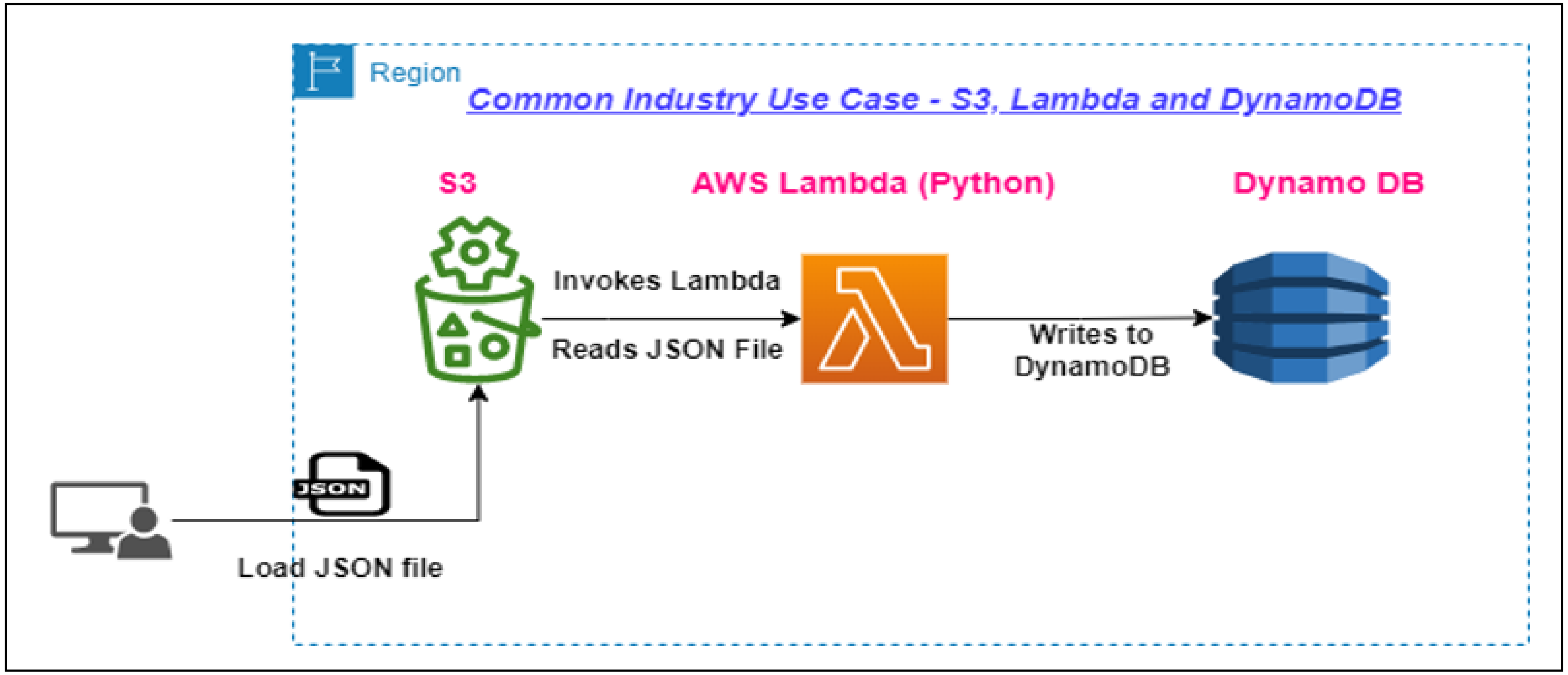
Section 5 : Serverless Use Case 1 - using API Gateway, AWS Lambda and S3



- *S3*
- *IAM Role*
- *AWS Lambda*
- *API Gateway*

AWS CDK v2 – Serverless Use Case

Section 6 : Serverless Use Case 2 - using S3, AWS Lambda and DynamoDB

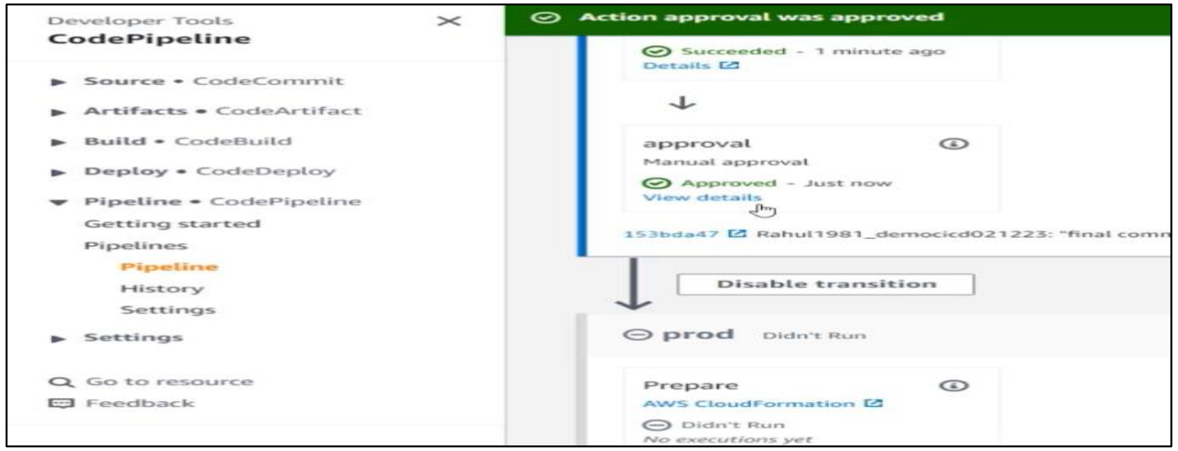


- *IAM Role*
- *S3*
- *S3 Event Notification*
- *AWS Lambda*
- *DynamoDB*

AWS CDK v2 – Beginner to Advanced

Section 7 : CI-CD Pipeline : Creating and Deploying AWS CDK Apps using CI-CD Pipeline

- **Building and Deploying a CI-CD Pipeline using AWS CDK v2**
- **Deploying AWS Services using the CI-CD Pipeline**



Developer Tools
CodePipeline

- Source • CodeCommit
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
  - Getting started
  - Pipelines
    - Pipeline
    - History
    - Settings
- Settings

Go to resource
Feedback

Action approval was approved

Succeeded - 1 minute ago
Details

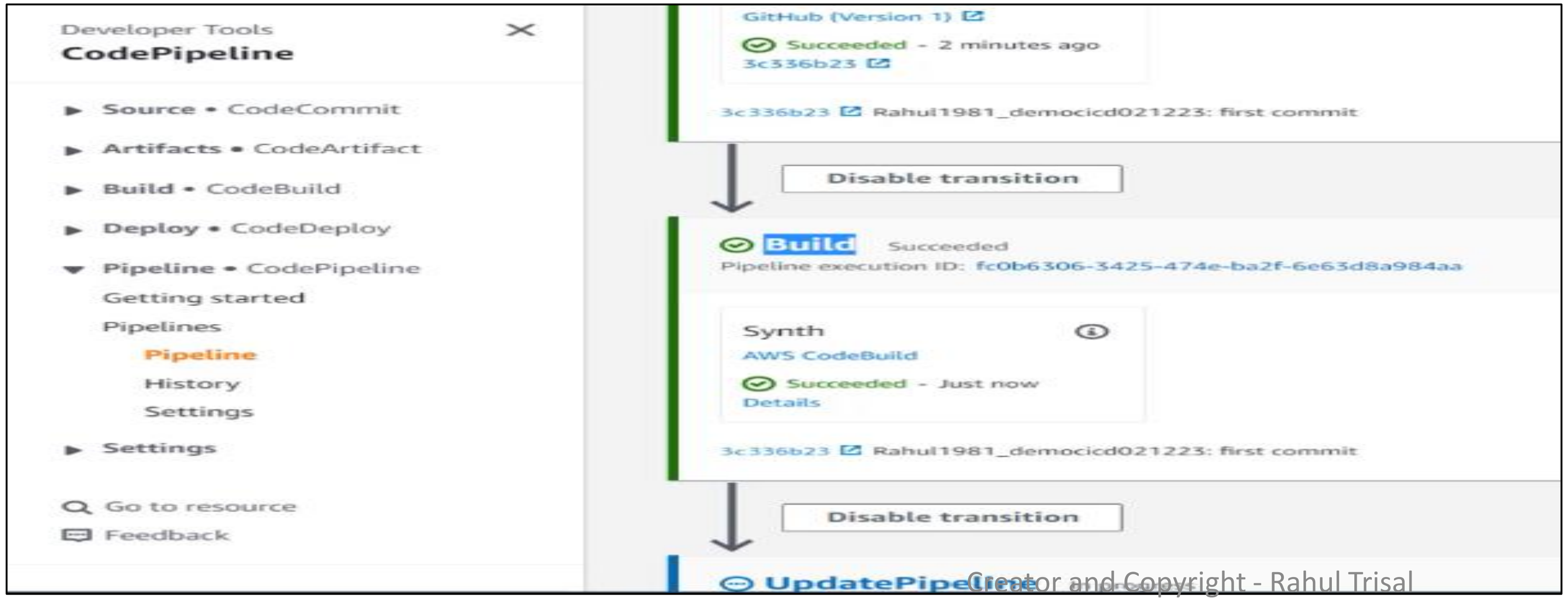
approval
Manual approval
Approved - Just now
View details

153bda47 Rahul1981_democicd021223: "final comm

Disable transition

prod Didn't Run

Prepare
AWS CloudFormation
Didn't Run
No executions yet



Developer Tools
CodePipeline

- Source • CodeCommit
- Artifacts • CodeArtifact
- Build • CodeBuild
- Deploy • CodeDeploy
- Pipeline • CodePipeline
  - Getting started
  - Pipelines
    - Pipeline
    - History
    - Settings
- Settings

Go to resource
Feedback

GitHub (Version 1)
Succeeded - 2 minutes ago
3c336b23

3c336b23 Rahul1981_democicd021223: first commit

Disable transition

Build Succeeded
Pipeline execution ID: fc0b6306-3425-474e-ba2f-6e63d8a984aa

Synth
AWS CodeBuild
Succeeded - Just now
Details

3c336b23 Rahul1981_democicd021223: first commit

Disable transition

UpdatePipeline

AWS CDK v2 – Beginner to Advanced

Section 8 : Additional AWS CDK v2 Concepts

- **Outputs in CDK**

- **Summary of AWS CDK Commands**

About Me

- I am **Rahul Trisal** working as an **AWS Solution Architect** in a Fortune 500 Organization
- 6X Cloud Certified
  - **AWS Solution Architect – Professional**
  - AWS Solution Architect – Associate
  - AWS Certified SysOps
  - AWS Cloud Practitioner
  - Azure Fundamental
  - IBM Bluemix Developer
- 200+ applications migrated working with Fortune 100 customers on large AWS Cloud Migration Programs
- Post content on AWS Careers, Architecture and Certification on my AWS YouTube channel and LinkedIn

Connect with me :

- Youtube - https://www.youtube.com/@trisalrahul/videos
- Linkedin - https://www.linkedin.com/in/rahul-trisal-7709628/
- Email – techcloudbyte@gmail.com
- Follow our LinkedIn page for latest on AWS - https://www.linkedin.com/company/tech-cloud-byte-techclobyte

Section 2:

Course Setup and Pre-requisites

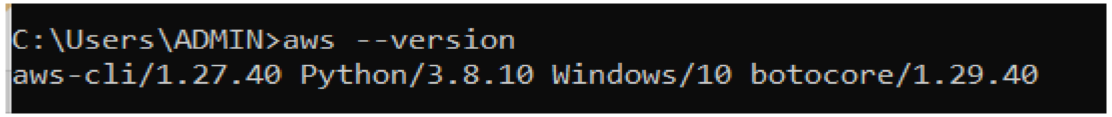
AWS CDK v2 – Pre-Requisites

1. Signup for AWS Account

2. IDE – Visual Code Studio - https://code.visualstudio.com/

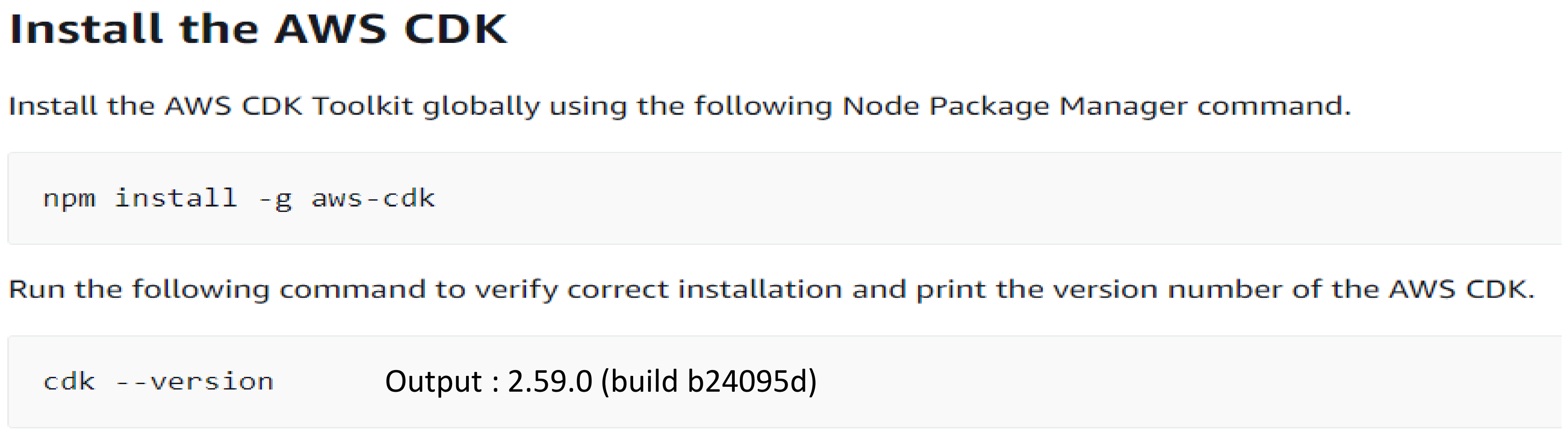
3. Install AWS CLI - https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

   Check using open Start -- > cmd -- > Run following command > *aws* ***- - version***; *output : aws-cli/2.7.24…*



```
C:\Users\ADMIN>aws --version
aws-cli/1.27.40 Python/3.8.10 Windows/10 botocore/1.29.40
```

4. Install the AWS CDK Toolkit for VS Code : https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/setup.html#install



Install the AWS CDK

Install the AWS CDK Toolkit globally using the following Node Package Manager command.

```
npm install -g aws-cdk
```

Run the following command to verify correct installation and print the version number of the AWS CDK.

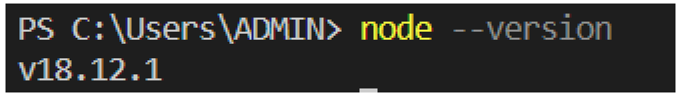
```
cdk --version
```

Output : 2.59.0 (build b24095d)

In case of error : Run following command >> "Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass" and then execute other commands

AWS CDK v2 – Pre-Requisites

5. Install Node.js - https://nodejs.org/en/ (Check by running following command >> **node - - version**, output should be > v10.3.0)

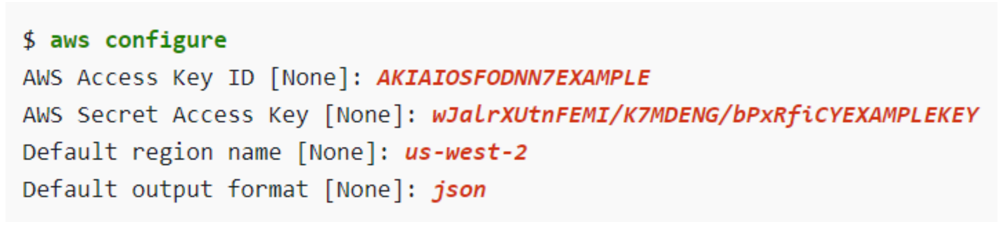


```
PS C:\Users\ADMIN> node --version
v18.12.1
```

To download for other Programming Languages – Use this link : https://cdkworkshop.com/15-prerequisites.html

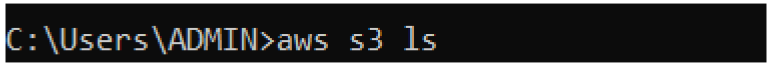
6. AWS Account User - Configure Credentials to access AWS services from Visual Studio - https://docs.aws.amazon.com/toolkit-for-vscode/latest/userguide/establish-credentials.html

- Create an IAM User
- Configure Credentials



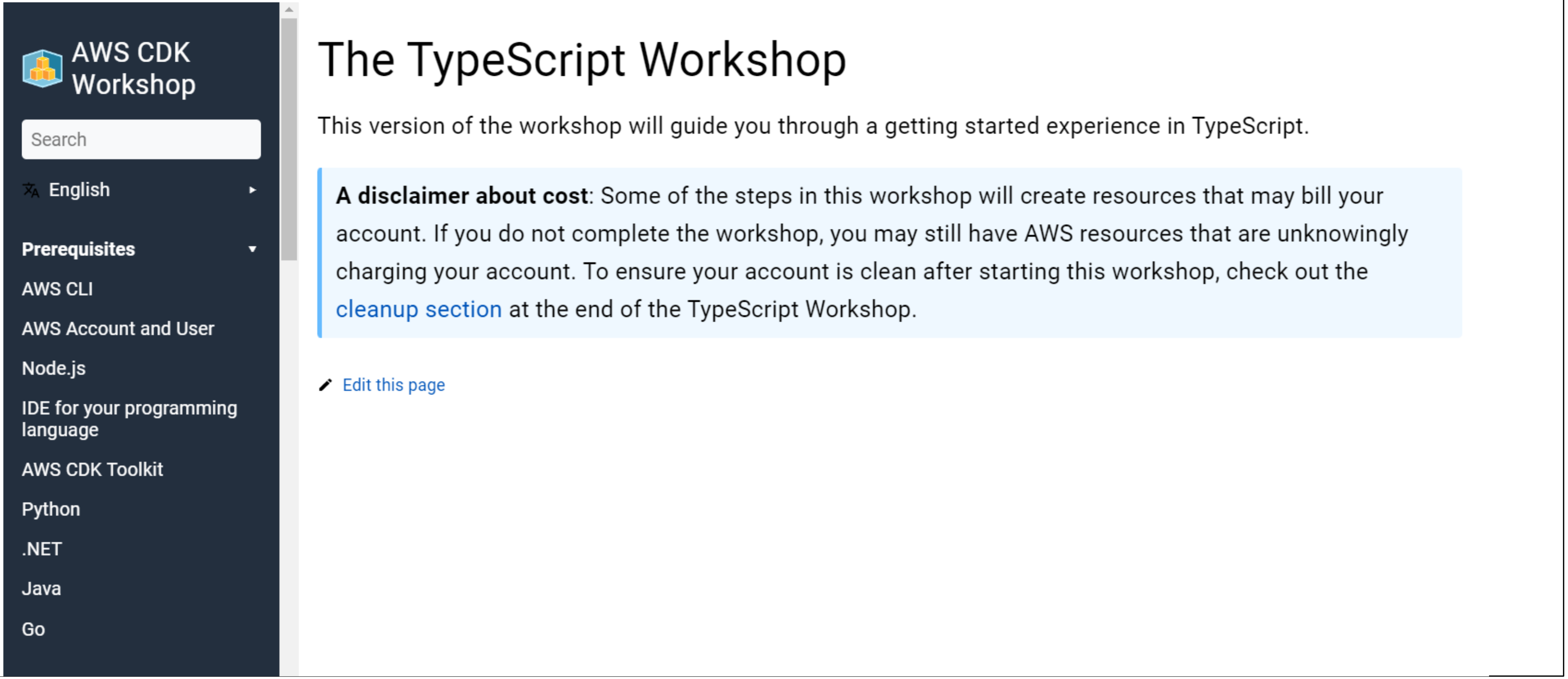
```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

- Test using following command If configured properly – > aws s3 ls (should return all s3 buckets)



```
C:\Users\ADMIN>aws s3 ls
```

AWS CDK – Pre-Requisites



AWS CDK Workshop

Search

English

Prerequisites

- AWS CLI
- AWS Account and User
- Node.js
- IDE for your programming language
- AWS CDK Toolkit
- Python
- .NET
- Java
- Go

The TypeScript Workshop

This version of the workshop will guide you through a getting started experience in TypeScript.

A disclaimer about cost: Some of the steps in this workshop will create resources that may bill your account. If you do not complete the workshop, you may still have AWS resources that are unknowingly charging your account. To ensure your account is clean after starting this workshop, check out the cleanup section at the end of the TypeScript Workshop.

Edit this page

https://cdkworkshop.com/15-prerequisites/100-awscli.html

Section 3:

AWS CDK - Basic Concepts

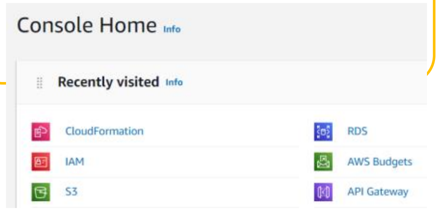
AWS CDK– Introduction

1. Evolution of AWS Infrastructure as Code
2. AWS CloudFormation Overview
3. What is AWS CDK
4. Benefits of AWS CDK
5. Relationship between AWS CDK and CloudFormation?
6. AWS CDK– Key Concepts
7. AWS CDK - How does AWS CDK work ?
8. AWS CDK - Project Structure

AWS CDK– Evolution of AWS Infrastructure as Code

Manually → **Scripting** → **Declarative** → **Imperative**

- **AWS Console**



Console Home Info

Recently visited Info

CloudFormation | RDS
IAM | AWS Budgets
S3 | API Gateway

- **AWS CLI**

```
aws-cli/1.27.40 Python/
```

- **AWS CloudFormation**

- **AWS CDK**

Pros
- Easy/No learning curve

Cons
- Not scalable
- Error prone

Pros
- Less error prone

Cons
- Not scalable
- No rollback/CI-CD Pipeline

Pros
- Scalable
- Rollback/CI-CD

Cons
- More code
- Learning Curve

Pros
- Less Code/Reusability
- Shorter learning curve

Cons
- Evolving

Evolution of Infrastructure as Code for AWS

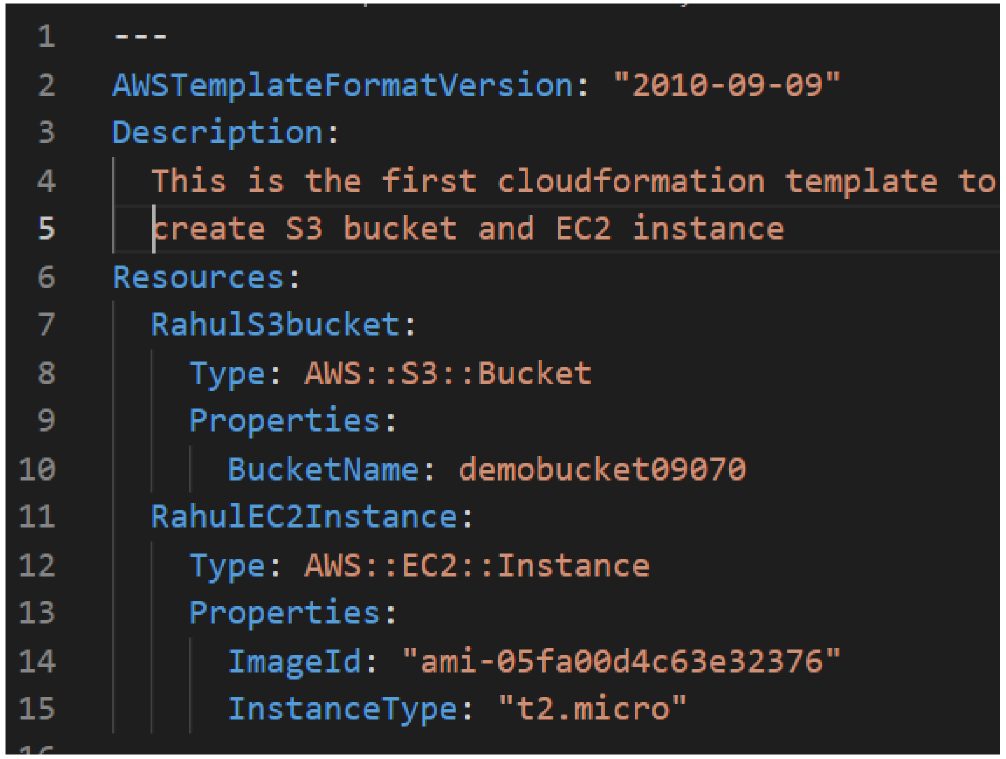
AWS CloudFormation Overview

What is AWS CloudFormation ?

AWS CloudFormation is an AWS service that **uses JSON or YAML** template files to **automate the setup of AWS resources.**

- ***Template*** - A CloudFormation template is a JSON or YAML formatted text file
- ***Stacks*** – In CloudFormation, all the resources are created as a single unit called a stack.
  - ✓ Such as 3 Tier Web App – ALB, EC2, ASG, DB etc. are created as a single unit through one template
  - ✓ Or create separate stacks such as Network Stack, Backend Stack

Sample CloudFormation Template for EC2 and S3



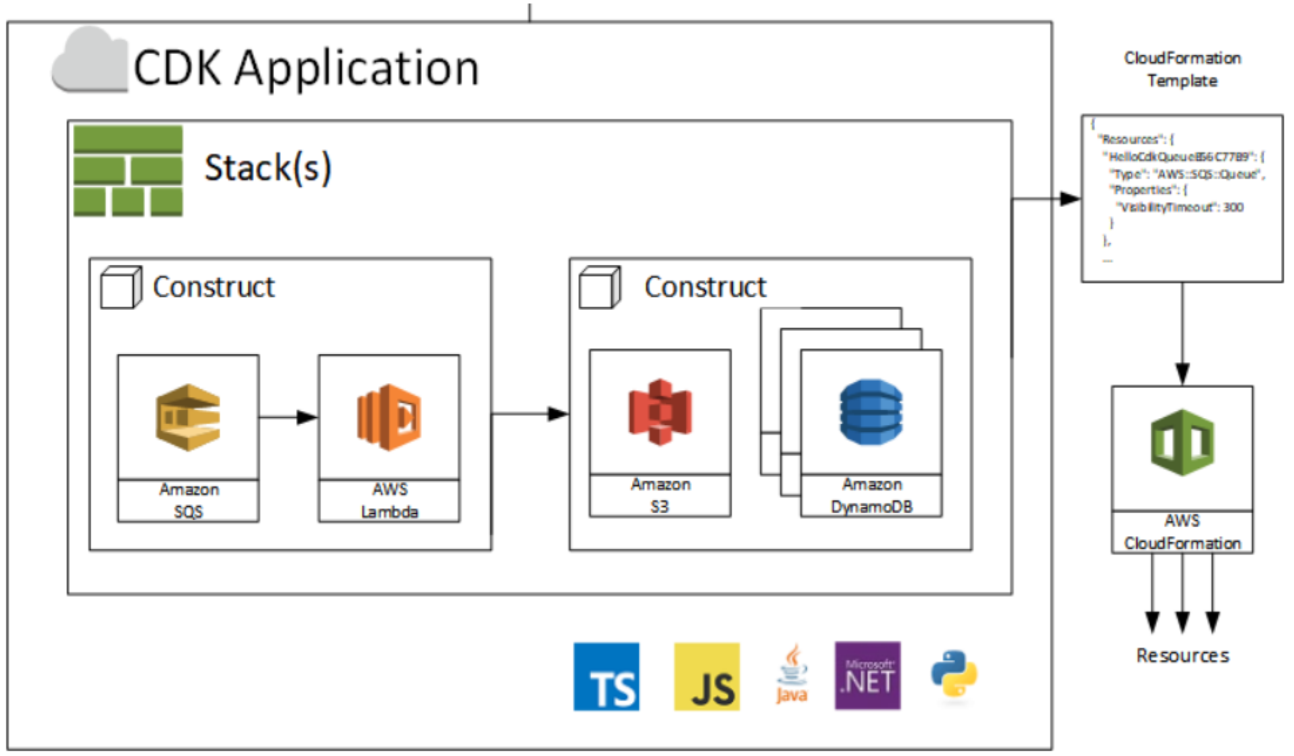
```yaml
---
AWSTemplateFormatVersion: "2010-09-09"
Description:
  This is the first cloudformation template to
  create S3 bucket and EC2 instance
Resources:
  RahulS3bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: demobucket09070
  RahulEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: "ami-05fa00d4c63e32376"
      InstanceType: "t2.micro"
```

AWS CDK– What is AWS CDK ?

What is AWS CDK ?

The AWS Cloud Development Kit (AWS CDK) is an :

- Open-source software development framework
- Used for defining **cloud infrastructure as code**
- Using **modern programming languages**
- Such as **TypeScript, JS, Python, Java, C#/.Net, and Go.**
- Deploying through **AWS CloudFormation.**



```
{
  "Resources": {
   "HelloCdkQueueB56C7789": {
    "Type": "AWS::SQS::Queue",
    "Properties": {
     "VisibilityTimeout": 300
    }
   },
   ...
```

source : aws

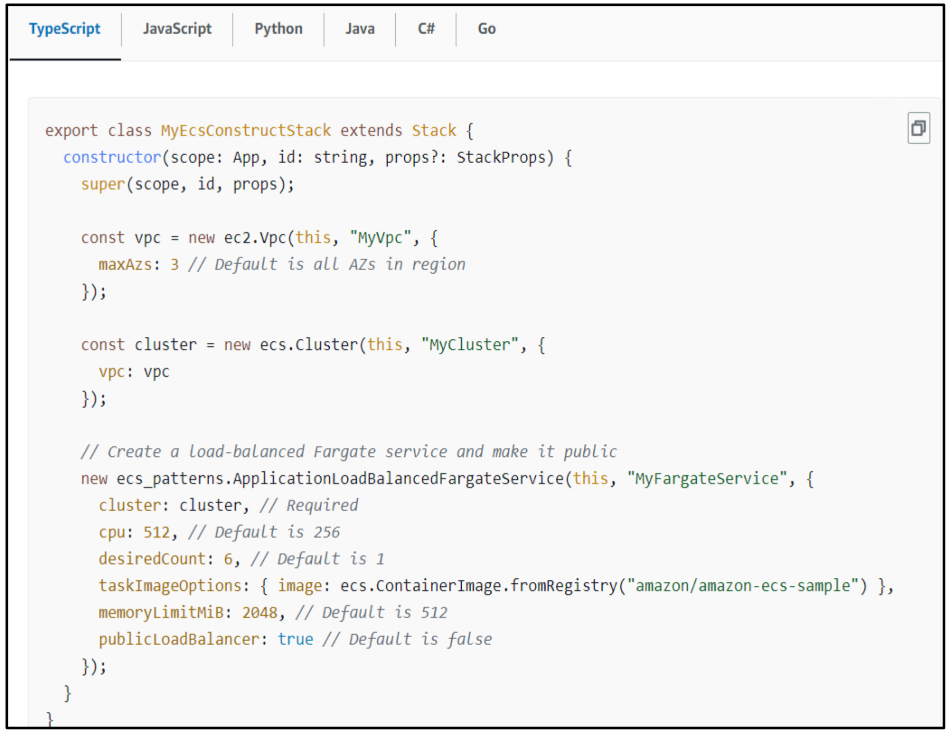
AWS CDK– Benefits

1. Write much less code

Amazon ECS service-Fargate launch type (19 AWS Services)

- AWS::EC2::EIP
- AWS::EC2::InternetGateway
- AWS::EC2::NatGateway
- AWS::EC2::Route
- AWS::EC2::RouteTable
- AWS::EC2::SecurityGroup
- AWS::EC2::Subnet
- AWS::EC2::SubnetRouteTableAssociation
- AWS::EC2::VPCGatewayAttachment
- AWS::EC2::VPC
- AWS::ECS::Cluster
- AWS::ECS::Service
- AWS::ECS::TaskDefinition
- AWS::ElasticLoadBalancingV2::Listener
- AWS::ElasticLoadBalancingV2::LoadBalancer
- AWS::ElasticLoadBalancingV2::TargetGroup
- AWS::IAM::Policy
- AWS::IAM::Role
- AWS::Logs::LogGroup

AWS CDK -15 lines of code



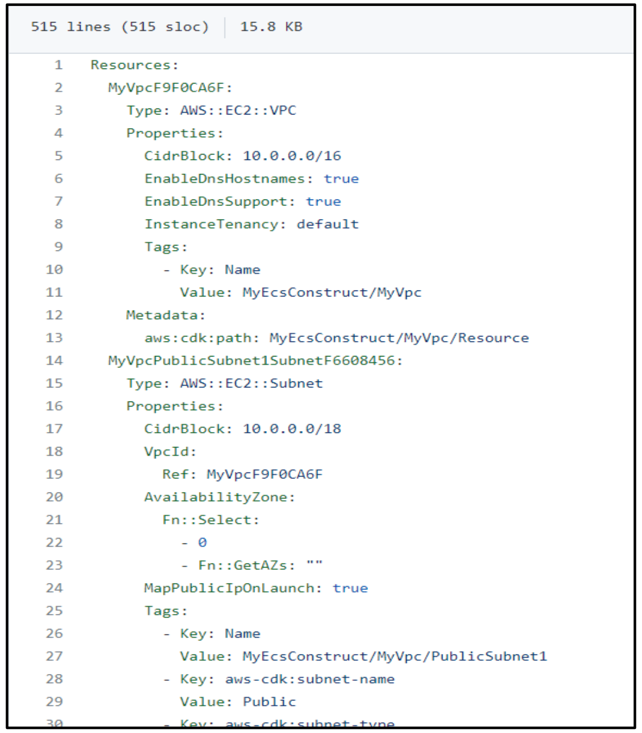
```
export class MyEcsConstructStack extends Stack {
  constructor(scope: App, id: string, props?: StackProps) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService", {
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample") },
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is false
    });
  }
}
```

CloudFormation – 500+ Lines of code



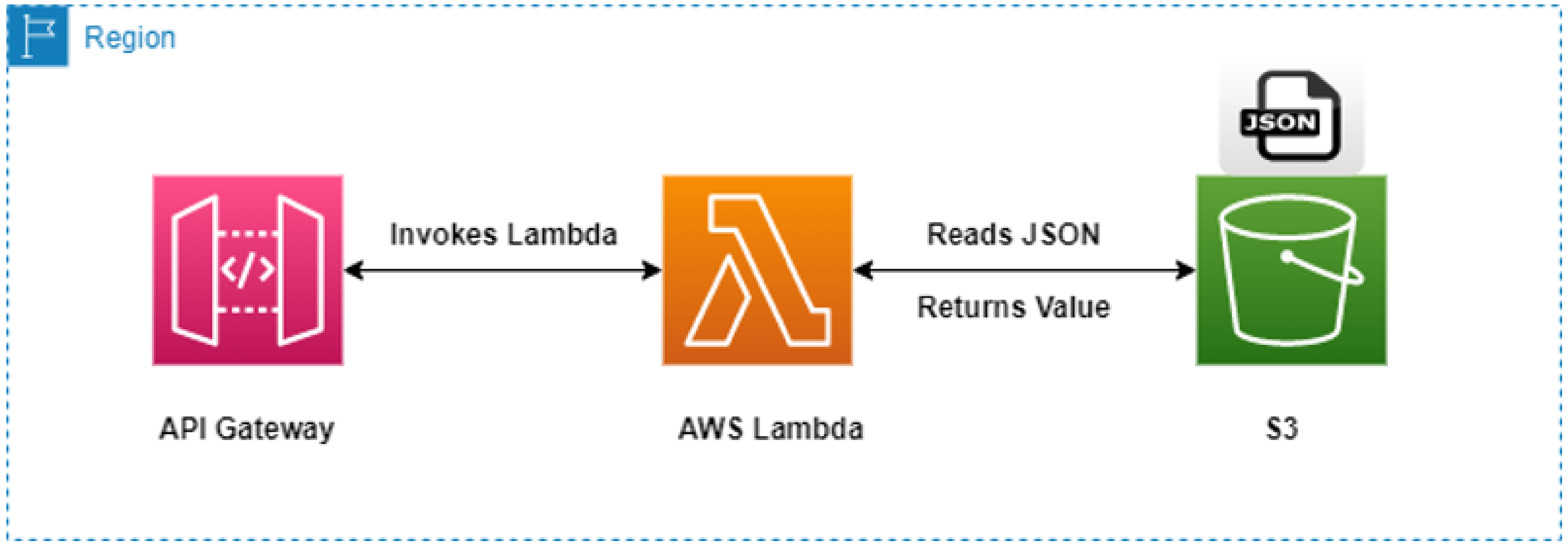
515 lines (515 sloc) | 15.8 KB

```
Resources:
  MyVpcF9F0CA6F:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      InstanceTenancy: default
      Tags:
        - Key: Name
          Value: MyEcsConstruct/MyVpc
    Metadata:
      aws:cdk:path: MyEcsConstruct/MyVpc/Resource
  MyVpcPublicSubnet1SubnetF6608456:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock: 10.0.0.0/18
      VpcId:
        Ref: MyVpcF9F0CA6F
      AvailabilityZone:
        Fn::Select:
          - 0
          - Fn::GetAZs: ""
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: MyEcsConstruct/MyVpc/PublicSubnet1
        - Key: aws-cdk:subnet-name
          Value: Public
        - Key: aws-cdk:subnet-type
```

Source : AWS

AWS Cloud Development Kit (CDK) v2 – Serverless Use Case

Serverless Use Case - using API Gateway, AWS Lambda and S3



- *S3*
- *IAM Role*
- *AWS Lambda*
- *API Gateway*

AWS CDK– Benefits

2. Developer-centric compared to AWS CloudFormation

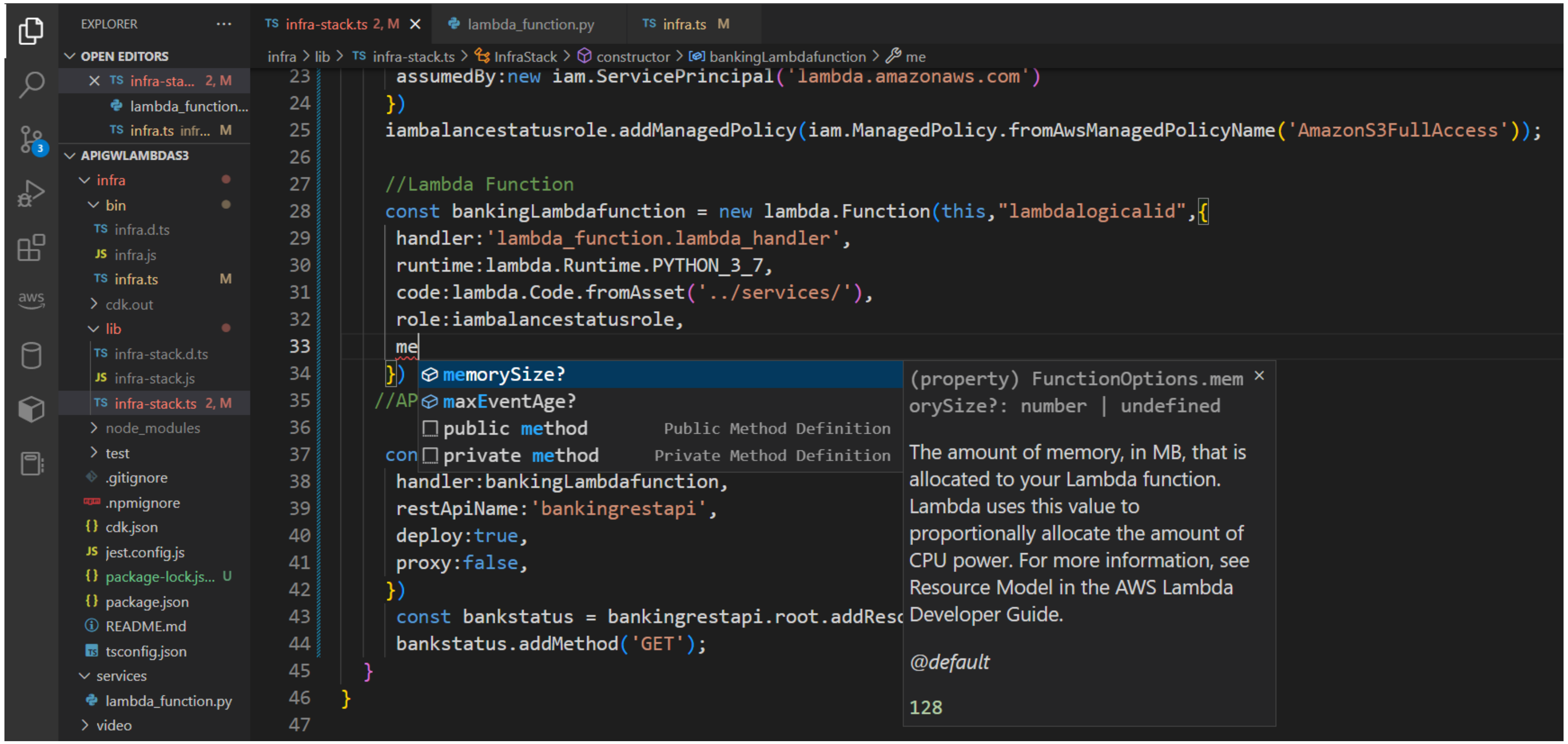
- CDK lets you **leverage your existing skills** and tools to build a cloud infrastructure.

- No Need to learn JSON or YAML

- **Multiple programming language support** and shorter learning curve.

AWS CDK– Benefits

3. Code completion within your IDE or editor. Less reference to documentation



```
    assumedBy:new iam.ServicePrincipal('lambda.amazonaws.com')
  })
  iambalancestatusrole.addManagedPolicy(iam.ManagedPolicy.fromAwsManagedPolicyName('AmazonS3FullAccess'));

  //Lambda Function
  const bankingLambdafunction = new lambda.Function(this,"lambdalogicalid",{
    handler:'lambda_function.lambda_handler',
    runtime:lambda.Runtime.PYTHON_3_7,
    code:lambda.Code.fromAsset('../services/'),
    role:iambalancestatusrole,
    me
  }) memorySize?
 //AP maxEventAge?
       public method          Public Method Definition
  con  private method         Private Method Definition
    handler:bankingLambdafunction,
    restApiName:'bankingrestapi',
    deploy:true,
    proxy:false,
  })
    const bankstatus = bankingrestapi.root.addResc
    bankstatus.addMethod('GET');
 }
}
```

(property) FunctionOptions.memorySize?: number | undefined

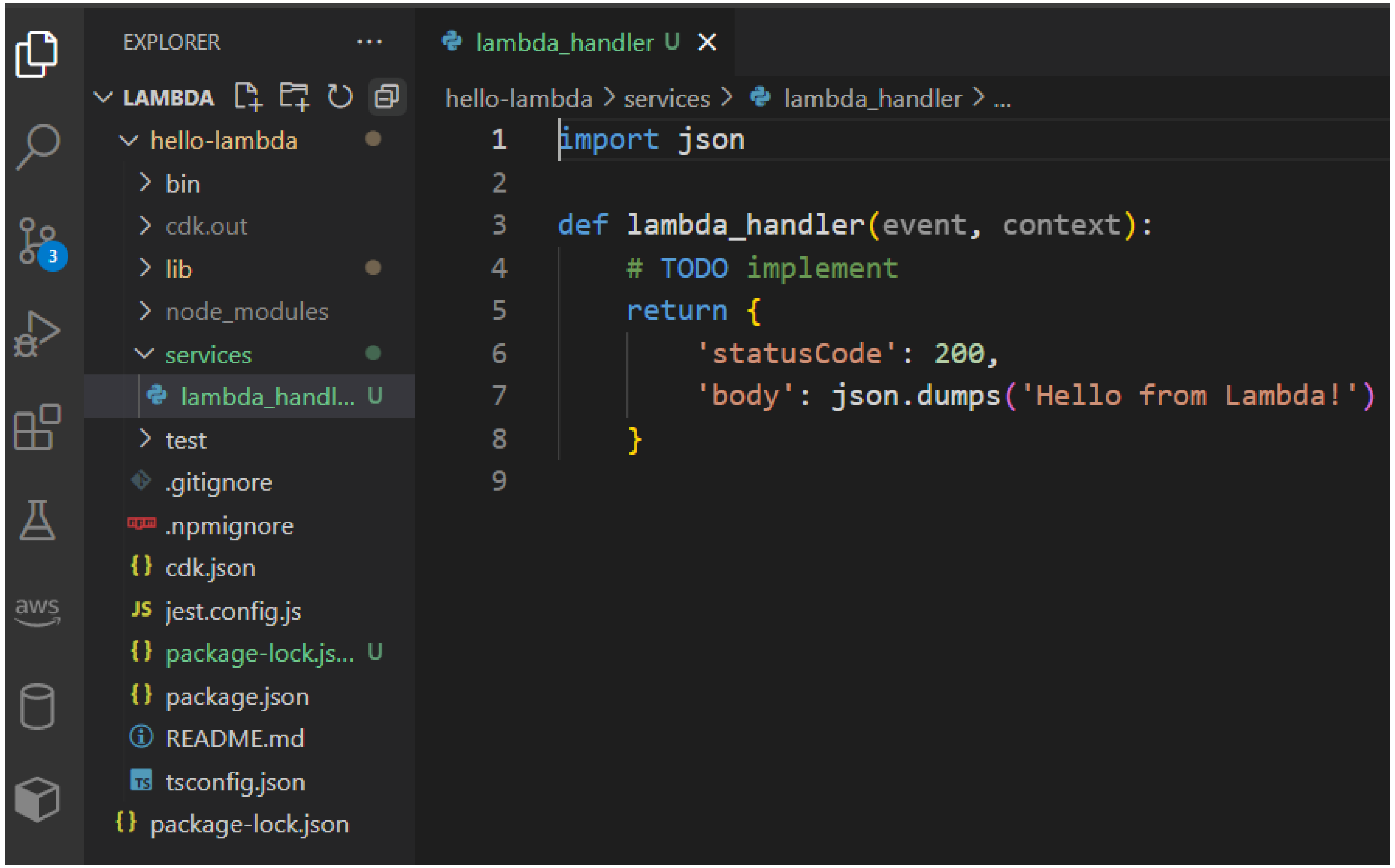
The amount of memory, in MB, that is allocated to your Lambda function. Lambda uses this value to proportionally allocate the amount of CPU power. For more information, see Resource Model in the AWS Lambda Developer Guide.

@default

128

AWS CDK– Benefits

4. Ability to execute application code with the infrastructure code



```python
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

AWS CDK– Benefits

5. Use high-level constructs to speed up development and re-usability

- The AWS CDK is shipped with an extensive library of constructs called the **AWS Construct Library**.
- The **construct library** is divided into **modules**, one for **each AWS service**.

L1 construct

- Provide all the **required CloudFormation attributes** for a particular cloud resource.
- These constructs are identified with name beginning with "Cfn," so they are also referred to as "Cfn constructs.

L2 construct

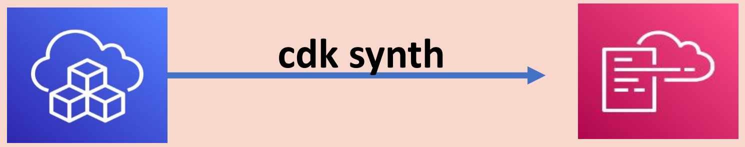
- **Don't need** to **configure every** attribute, Instead provided **"sensible defaults"** to easily spin up a resource.

L3 construct

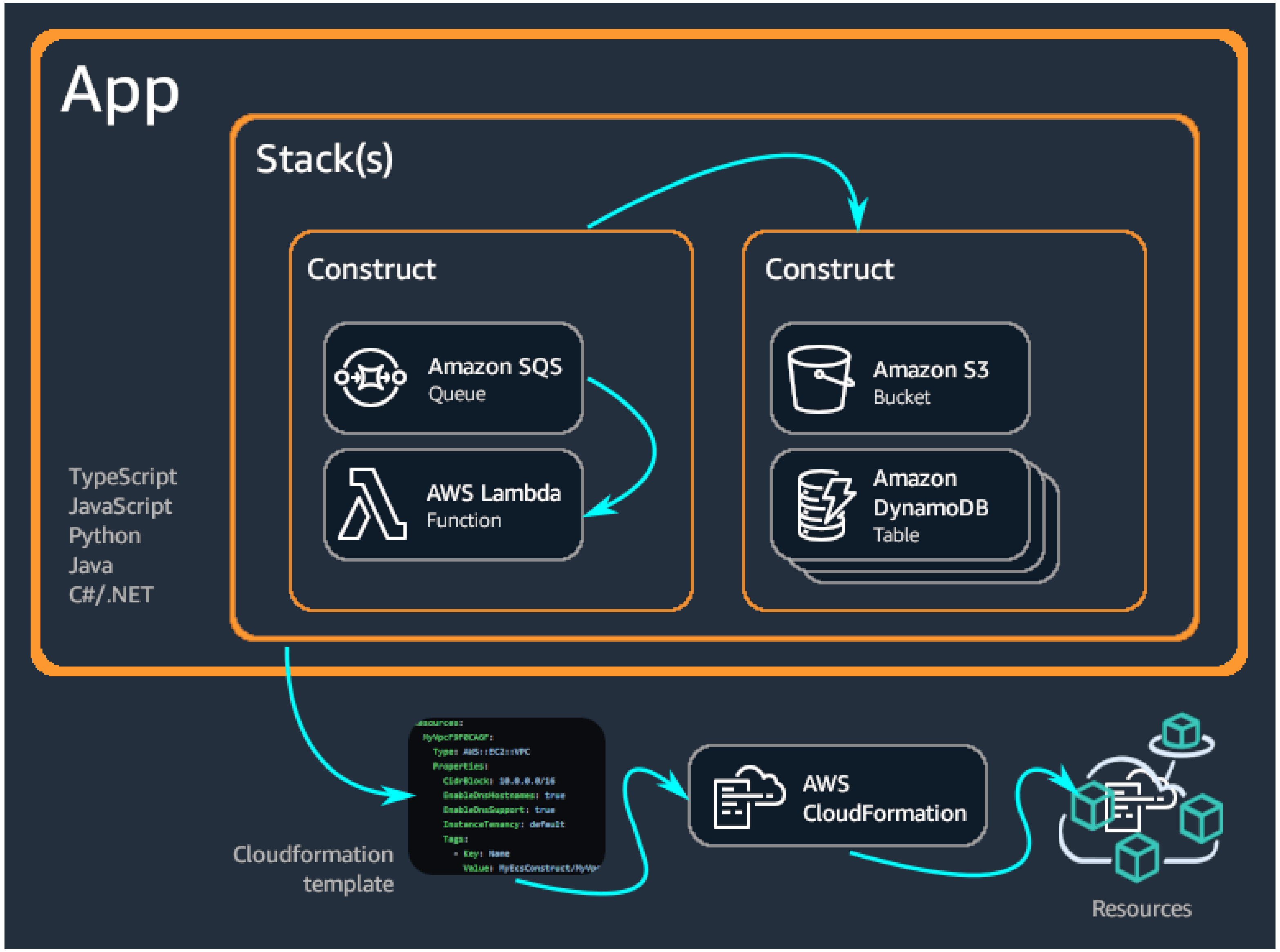
- A Level 3 construct represents various cloud resources that work together to accomplish a particular task called **"patterns".**
- **ApplicationLoadBalancedFarageteService construct** will create an ECS cluster with Fargate, an ECR repository and ALB etc.

AWS CDK and AWS CloudFormation

What is the relationship between AWS CDK and CloudFormation?

AWS CDK	CloudFormation
Used to write Infrastructure as Code	Used to write Infrastructure as Code
Developer-centric toolkit leveraging modern programming languages	Uses YAML/JSON
AWS CDK applications run, they compile down to fully formed CloudFormation JSON/YAML templates 	
CDK leverages CloudFormation providing all the benefits CloudFormation provides such as safe deployment, automatic rollback, and drift detection.	

AWS CDK– Key Concepts



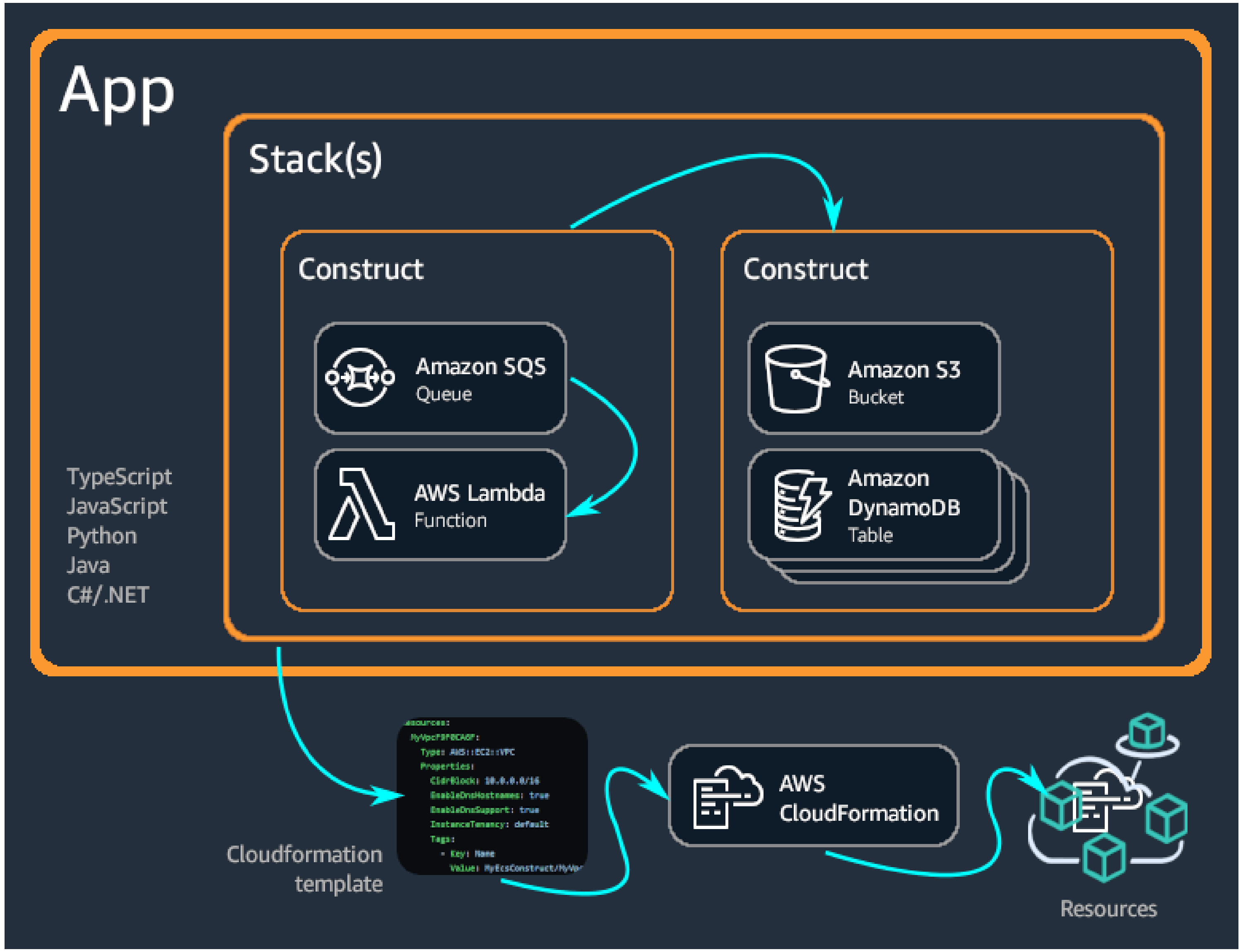
```
Resources:
  MyVpcF9F0CA6F:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      InstanceTenancy: default
      Tags:
        - Key: Name
          Value: MyEcsConstruct/MyVp
```

AWS CDK

1. ***App***
2. ***Stack***
3. ***Construct***
4. ***Resources***

Source : https://docs.aws.amazon.com/cdk/v2/guide/home.html

AWS CDK– Key Concepts



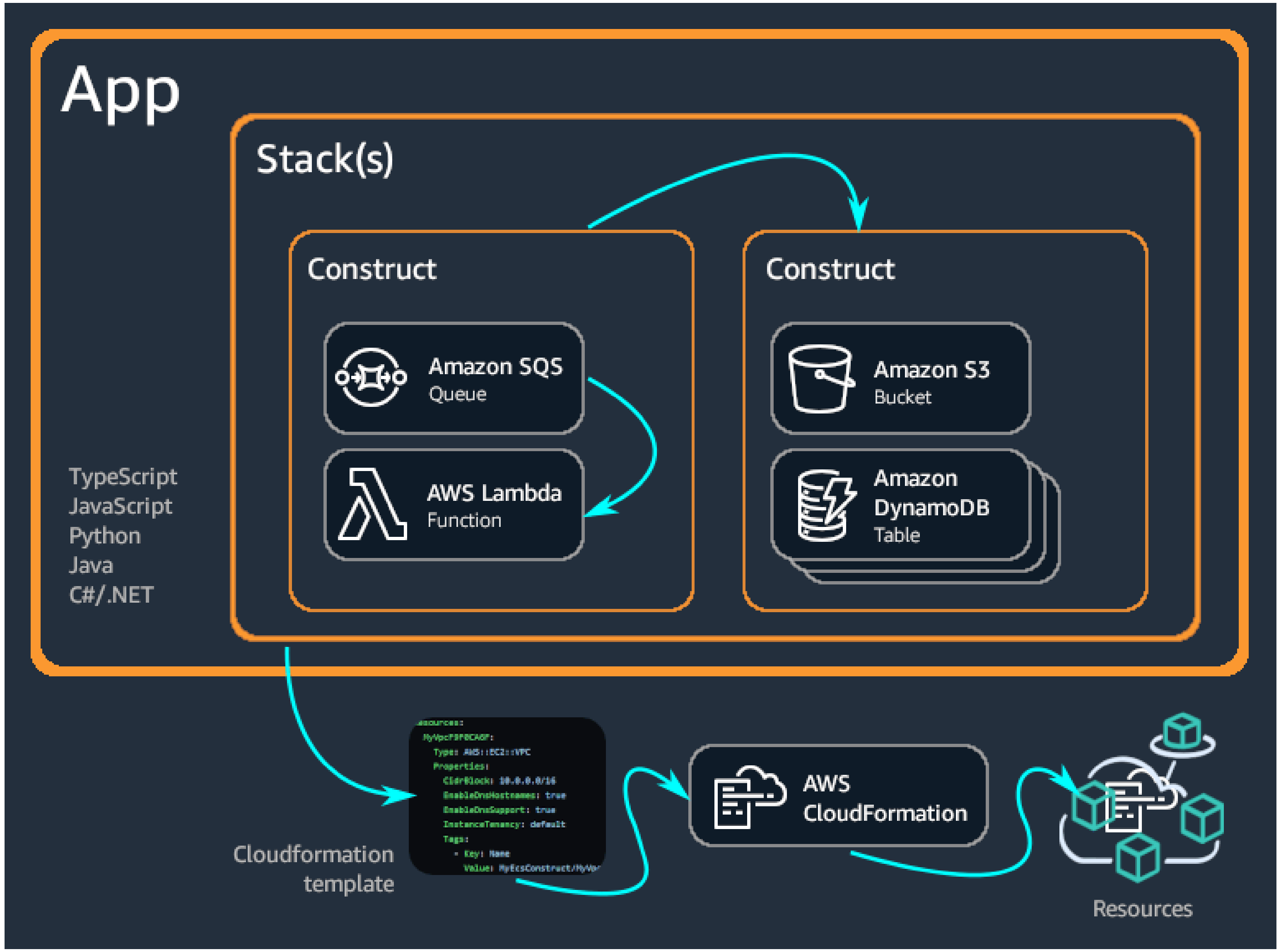
```
Resources:
  MyVpcF9F0CA6F:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      InstanceTenancy: default
      Tags:
        - Key: Name
          Value: MyEcsConstruct/MyVp
```

1. App

- App serves as the project **deliverable scope**
- An App is a **container** for **one or more stacks**
- Stacks within a **single App** can easily refer to each others' resources

Source : https://docs.aws.amazon.com/cdk/v2/guide/home.html

AWS CDK– Key Concepts



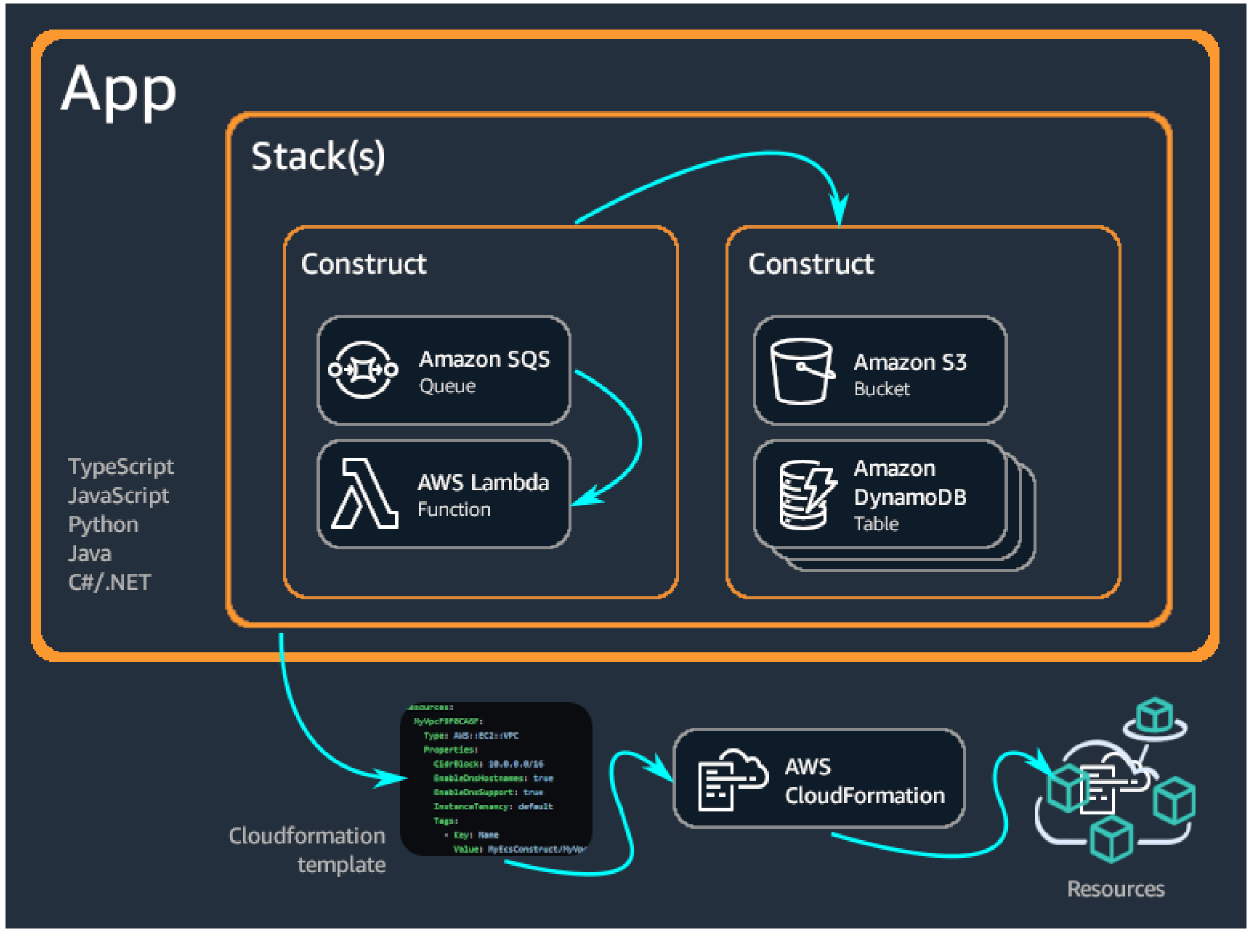
```
Resources:
  MyVpcF9F0CA6F:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      InstanceTenancy: default
      Tags:
        - Key: Name
          Value: MyEcsConstruct/MyVpc
```

Source : https://docs.aws.amazon.com/cdk/v2/guide/home.html

2. Stacks

- **Unit of deployment** in CDK is called a *stack*.
- All AWS resources defined within the scope of a stack are provisioned as a single unit.
- Similar to CloudFormation Stack
  - IAM Stack/Security Stack
  - Networking Stack
  - Function Stack
  - DB Stack

AWS CDK– Key Concepts



```
esources:
  MyVpcF9F0CA6F:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      InstanceTenancy: default
      Tags:
        - Key: Name
          Value: MyEcsConstruct/MyVp
```

3. Constructs

- Constructs are **basic building blocks of AWS CDK app**s.
- CDK includes a collection of constructs called the **AWS Construct Library,** containing constructs for every AWS service.
- A construct can represent a **single AWS resource**, such as S3 bucket or **multiple related AWS resources**
- It represents a "cloud component" and encapsulates everything CloudFormation needs to create the component.

Source : https://docs.aws.amazon.com/cdk/v2/guide/home.html

AWS CDK– Key Concepts

L1 construct

Provide all the **required CloudFormation attributes** for a particular cloud resource.

L2 construct

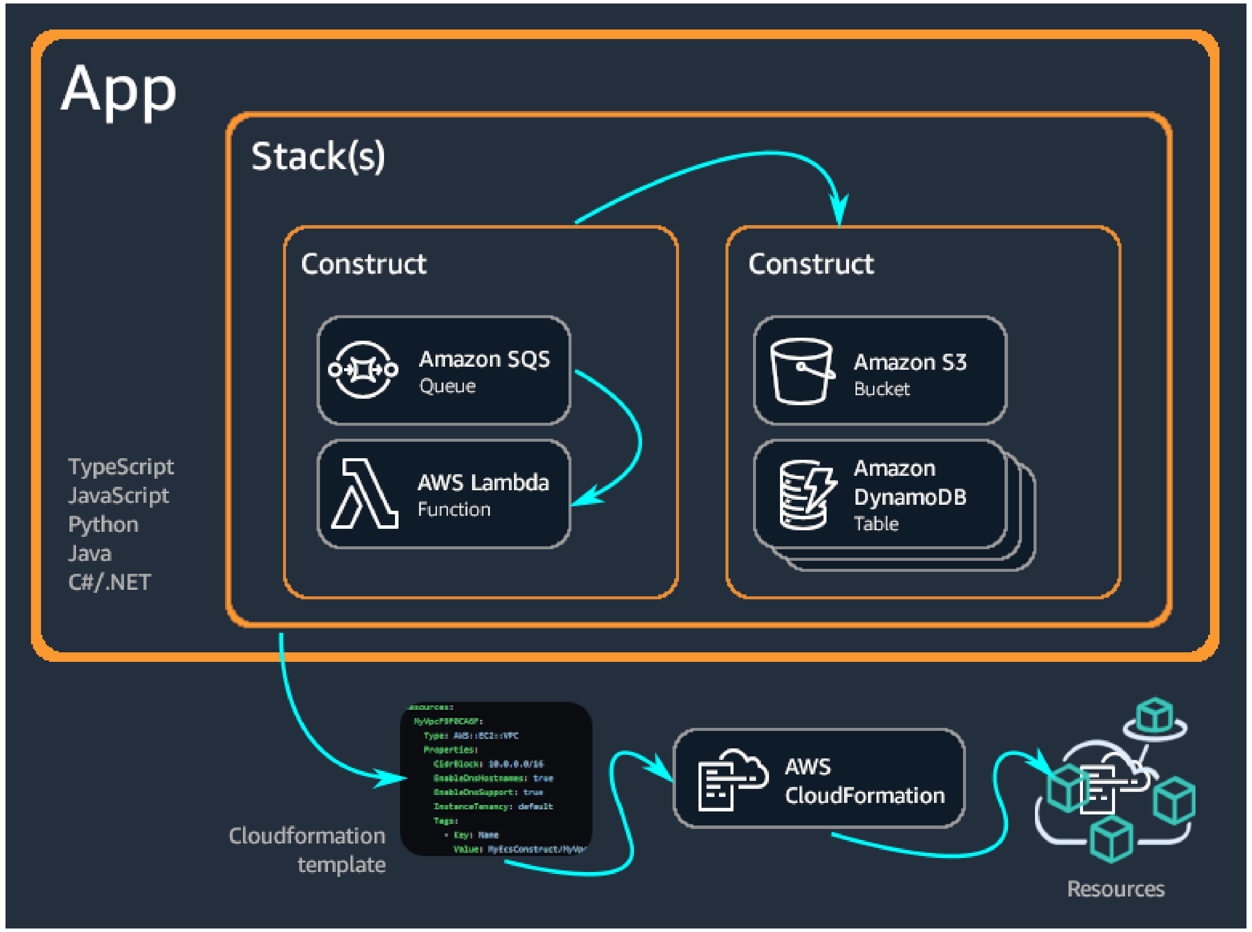
Don't need to **configure every** attribute, Instead provided **"sensible defaults"** to easily spin up a resource.

L3 construct

A Level 3 construct represents various cloud resources that work together to accomplish a particular task called **"patterns".**

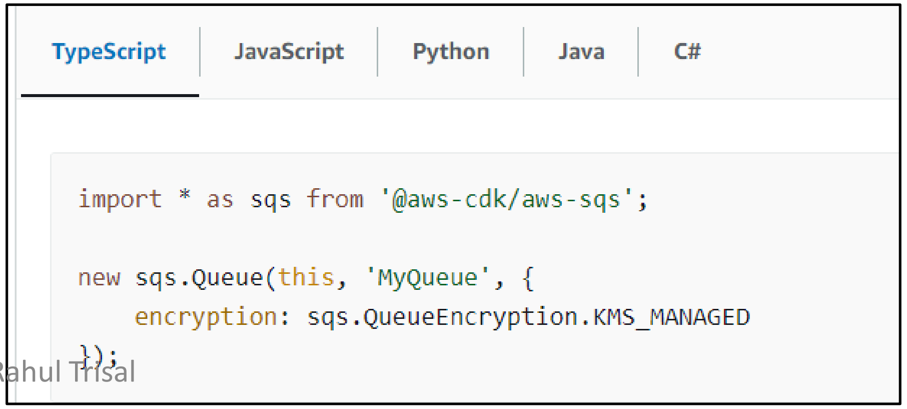
Example - **ApplicationLoadBalancedFarageteService construct** will create an ECS cluster powered by Fargate, an ECR repository to host your Docker images, Application Load Balancer to access your containers

AWS CDK– Key Concepts



4. Resources

- Create an **instance of a resource** using its corresponding construct
- Pass **scope** as the first argument
- **Logical ID** of the construct
- Set of **configuration properties** (props).
- For example, create Amazon SQS queue with AWS KMS encryption using the sqs.Queue construct from the AWS Construct Library.

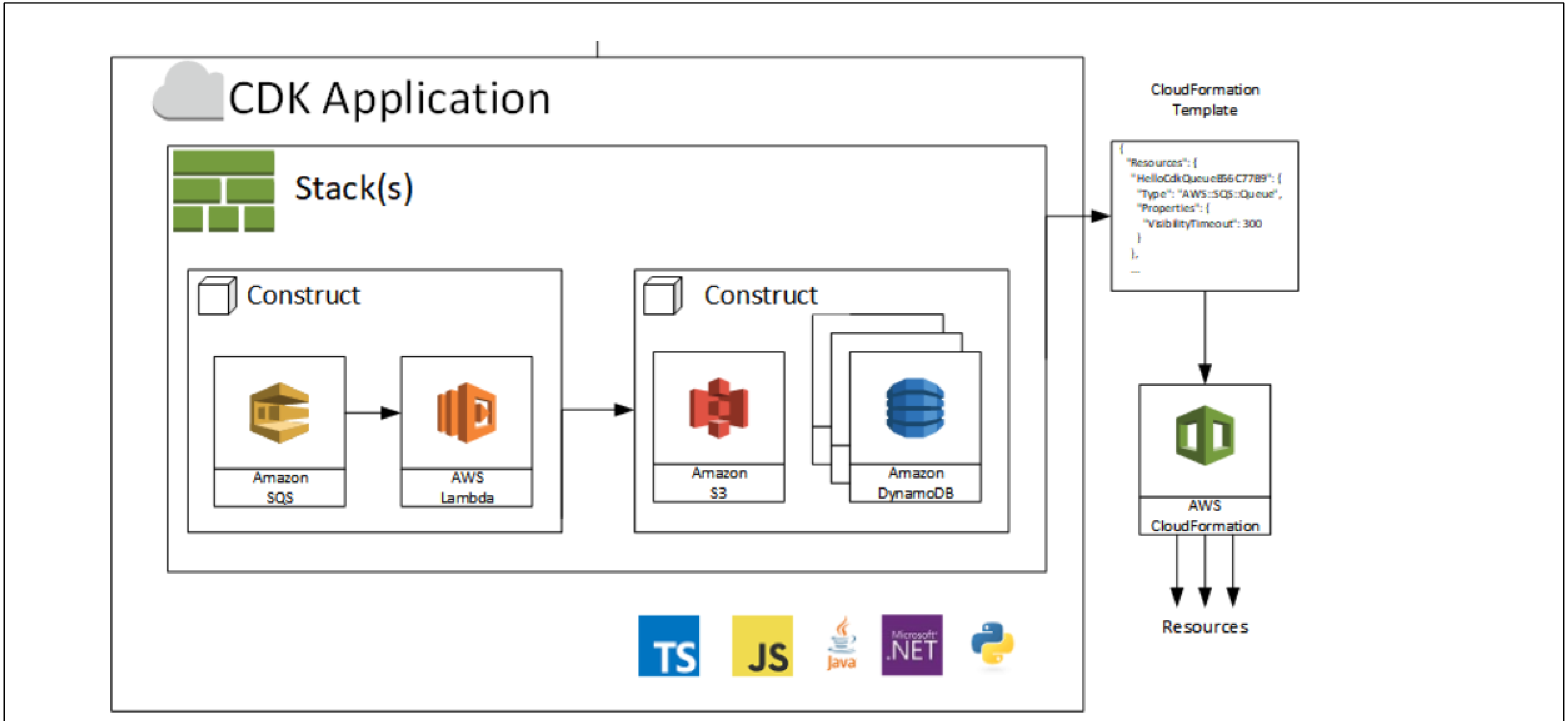


```
import * as sqs from '@aws-cdk/aws-sqs';

new sqs.Queue(this, 'MyQueue', {
    encryption: sqs.QueueEncryption.KMS_MANAGED
});
```

Source : https://docs.aws.amazon.com/cdk/v2/guide/home.html

AWS CDK– Key Concepts

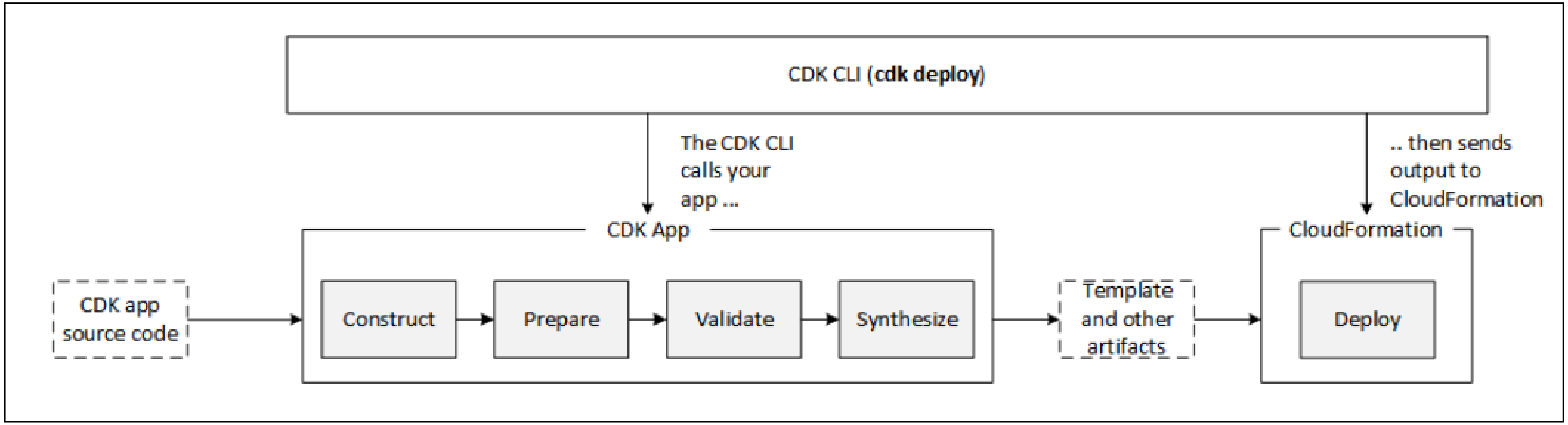


```
{
  "Resources": {
    "HelloCdkQueueE56C77B9": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
        "VisibilityTimeout": 300
      }
    },
    ...
```

https://aws.amazon.com/blogs/aws/aws-cloud-development-kit-cdk-typescript-and-python-are-now-generally-available/

AWS CDK– How does AWS CDK work ?

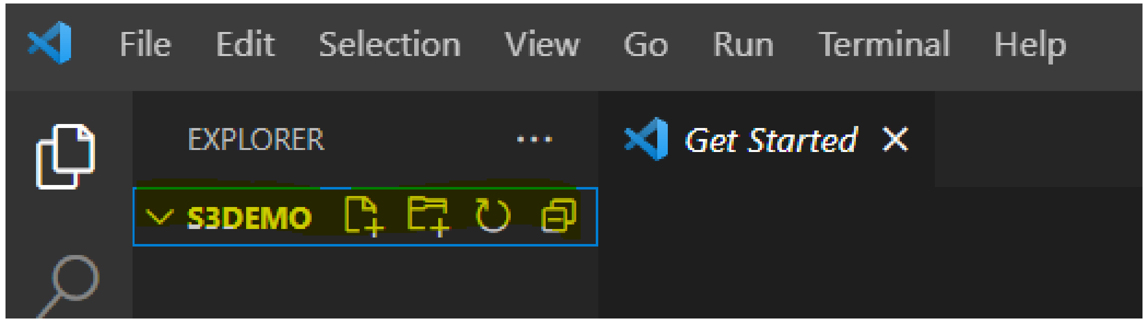
How does AWS CDK work – Execution Steps

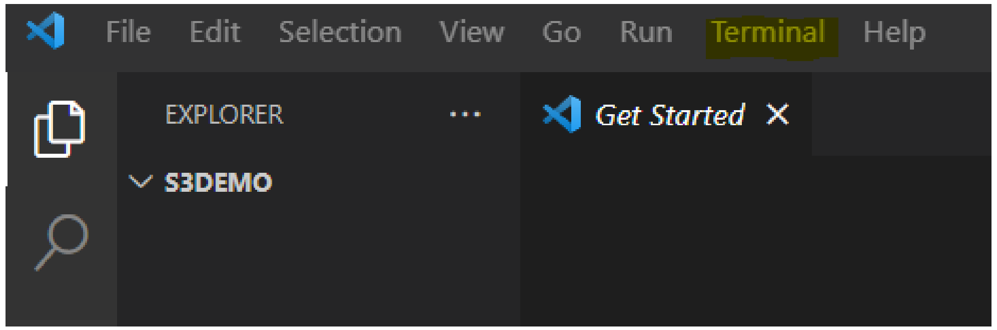


Source : https://docs.aws.amazon.com/cdk/v2/guide/apps.html

AWS CDK – Project Structure

1. **Create a folder on the Local Drive, Open the VSCode Editor with that Folder. Then Open 'Terminal in the VSCode'.**

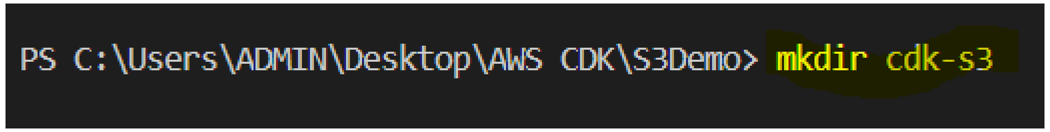




2. **Create the app (**Each AWS CDK app needs its own directory, with its own local module dependencies.)

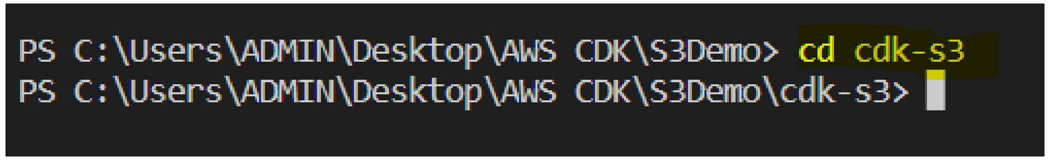
   Make a directory **cdk-s3** (or any other based on your choice) and then **change directory with cd**

- ***mkdir cdk-s3***



```
PS C:\Users\ADMIN\Desktop\AWS CDK\S3Demo> mkdir cdk-s3
```

- ***cd cdk-s3***



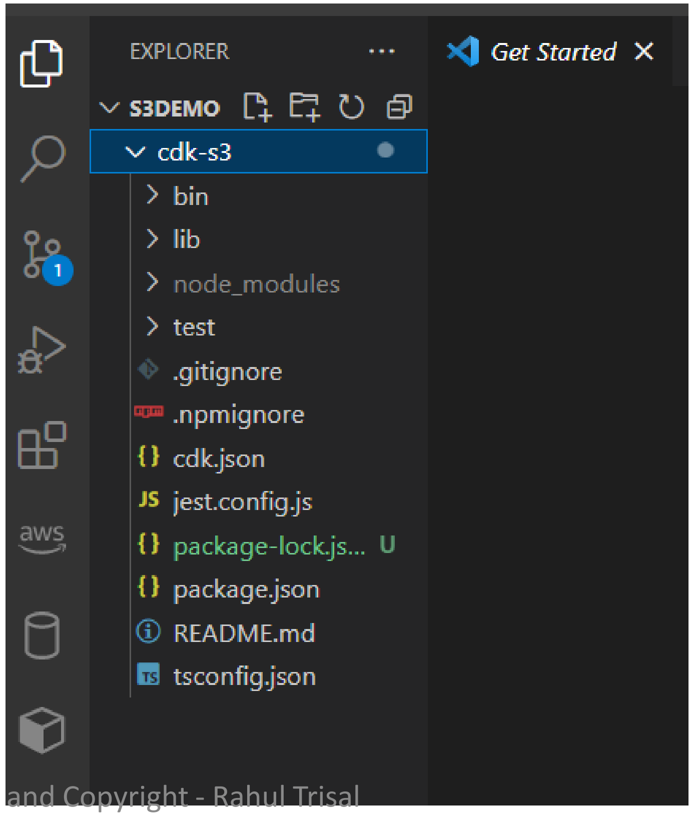
```
PS C:\Users\ADMIN\Desktop\AWS CDK\S3Demo> cd cdk-s3
PS C:\Users\ADMIN\Desktop\AWS CDK\S3Demo\cdk-s3>
```

AWS CDK – Project Structure

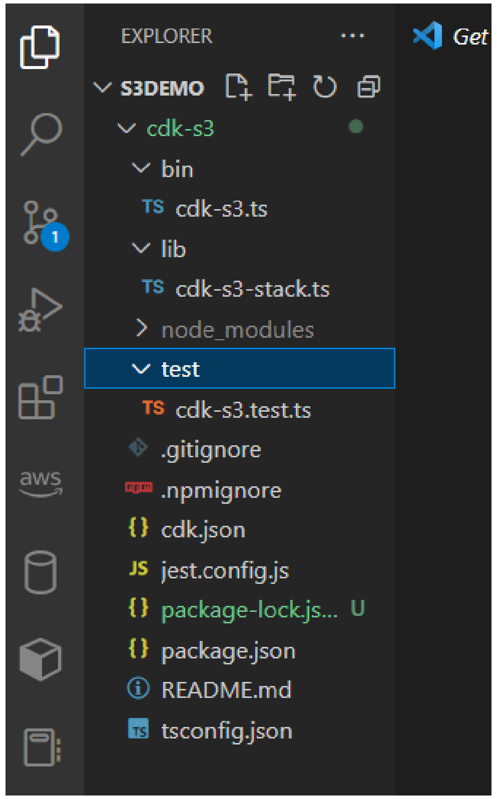
3. Initialize the app by using the **cdk init** command, specify the desired template ("**app**") and **programming language**

- *cdk init app --language typescript*

Below folder structure will be generated :

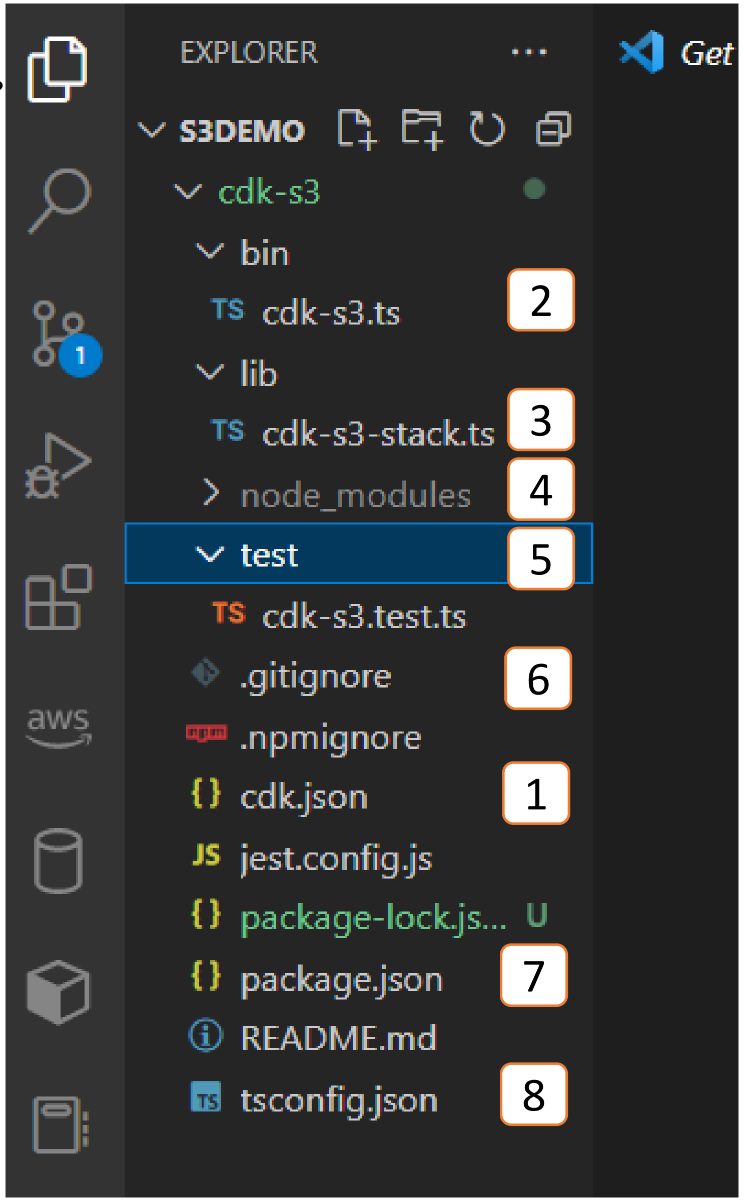


```
EXPLORER ...
v S3DEMO
  v cdk-s3
    > bin
    > lib
    > node_modules
    > test
    .gitignore
    .npmignore
    cdk.json
    jest.config.js
    package-lock.js...  U
    package.json
    README.md
    tsconfig.json
```



```
EXPLORER ...
v S3DEMO
  v cdk-s3
    v bin
      cdk-s3.ts
    v lib
      cdk-s3-stack.ts
    > node_modules
    v test
      cdk-s3.test.ts
    .gitignore
    .npmignore
    cdk.json
    jest.config.js
    package-lock.js...  U
    package.json
    README.md
    tsconfig.json
```

AWS CDK – Project Structure



1. cdk.json

- Tells the AWS Toolkit(CLI) how to run your app.
- "npx ts-node --prefer-ts-exts bin/hello-lambda.ts"

2. bin/cdk-workshop.ts

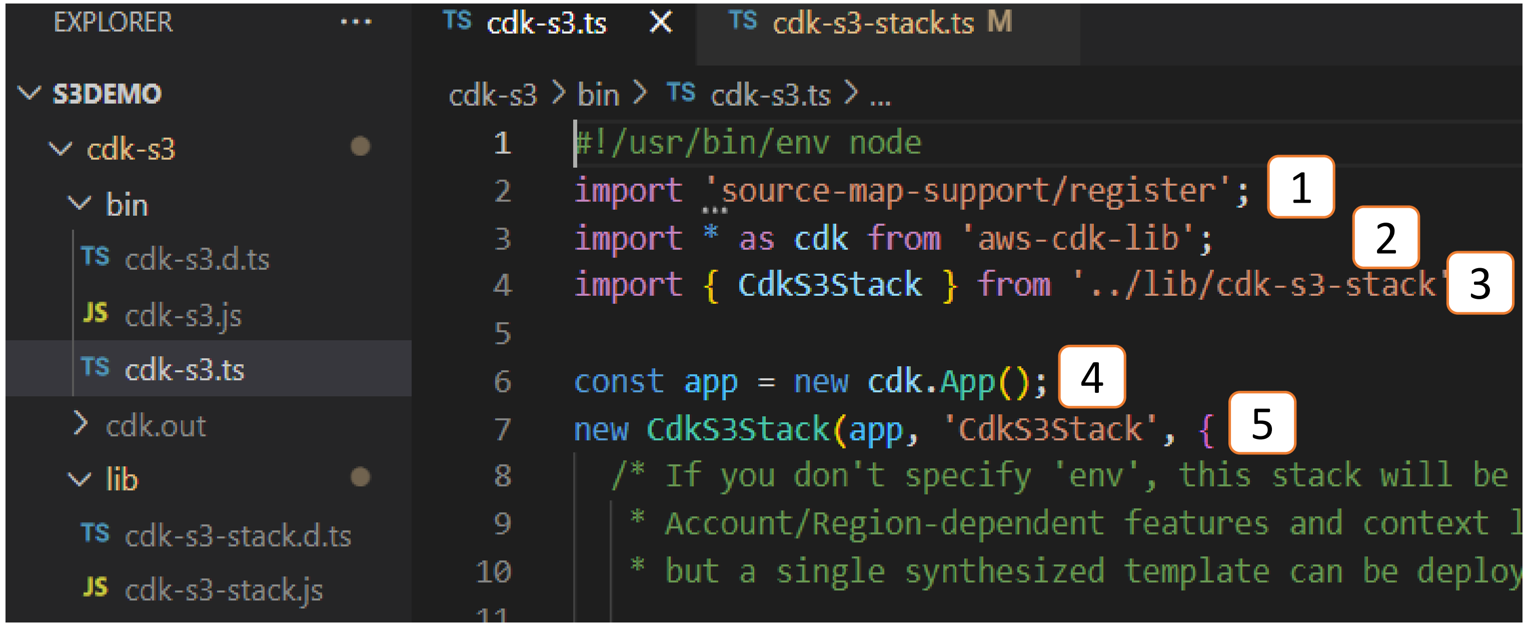
- Entrypoint of the CDK application.
- Will load the stack defined in lib/cdk-workshop-stack.ts.

3. lib/cdk-workshop-stack.ts (Most Important File)

- Is where your CDK application's main stack is defined.

AWS CDK – Project Structure

bin/cdk-workshop.ts



```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { CdkS3Stack } from '../lib/cdk-s3-stack'

const app = new cdk.App();
new CdkS3Stack(app, 'CdkS3Stack', {
  /* If you don't specify 'env', this stack will be
   * Account/Region-dependent features and context l
   * but a single synthesized template can be deploy
```

1. import **'source-map-support/register'**

2. **aws-cdk-lib** – main cdk package contains majority of AWS construct library.

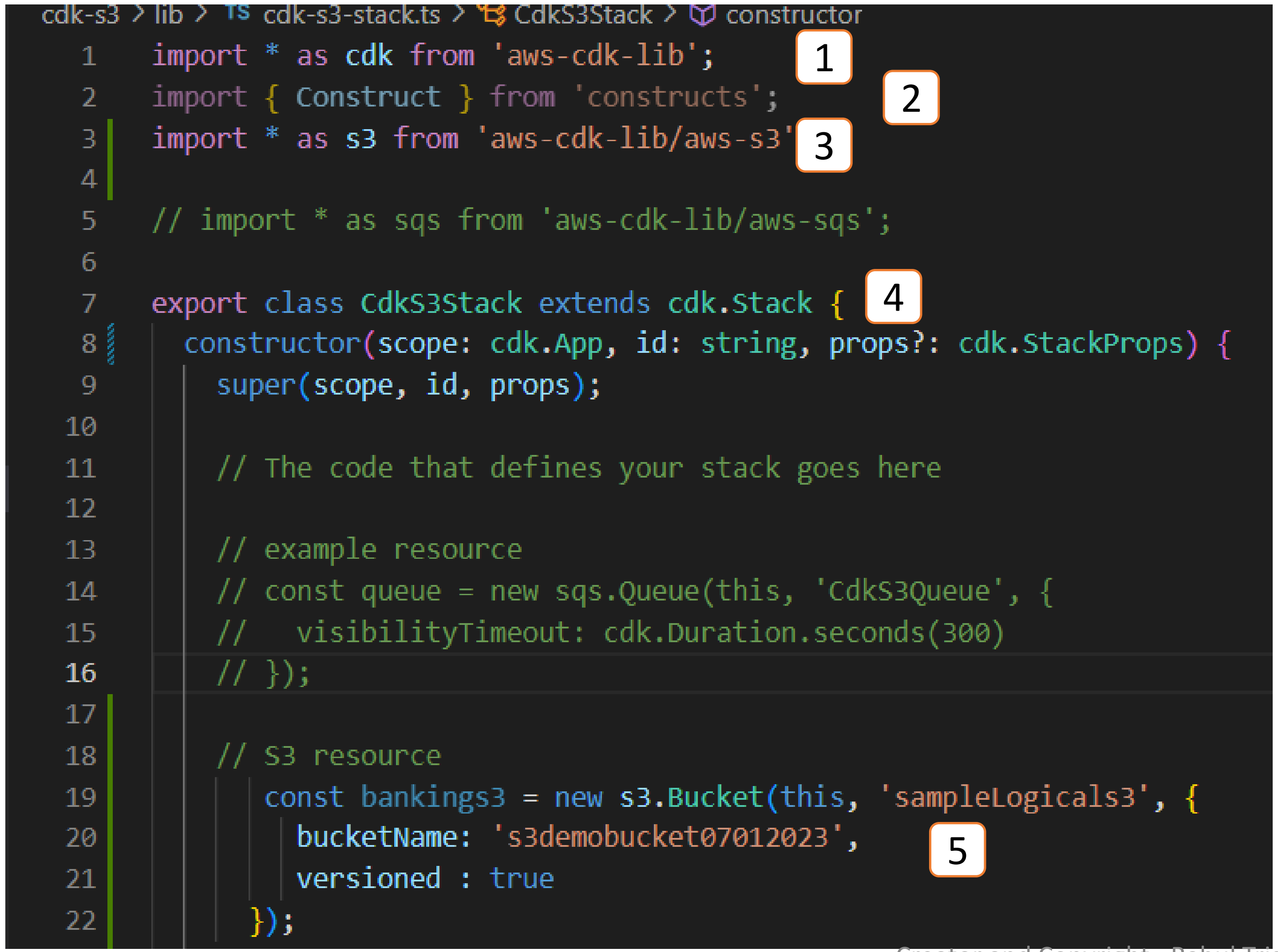
3. **Imports** the stack from **lib** folder

4. **New CDK application**
   - Entry point for app

5. **New Stack** (scope-app, logical id – stackname, properties)

AWS CDK – Your first AWS CDK app – S3 Bucket

3. Add the AWS Service Modules that will be created along with the Constructs


cdk-s3 > lib > TS cdk-s3-stack.ts > CdkS3Stack > constructor
1
2
3
4
5

```typescript
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as s3 from 'aws-cdk-lib/aws-s3'

// import * as sqs from 'aws-cdk-lib/aws-sqs';

export class CdkS3Stack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The cdk that defines your stack goes here

    // example resource
    // const queue = new sqs.Queue(this, 'CdkS3Queue', {
    //   visibilityTimeout: cdk.Duration.seconds(300)
    // });

    // S3 resource
      const bankings3 = new s3.Bucket(this, 'sampleLogicals3', {
        bucketName: 's3demobucket07012023',
        versioned : true
      });
```

1. import from 'aws-cdk-lib' (created by default)

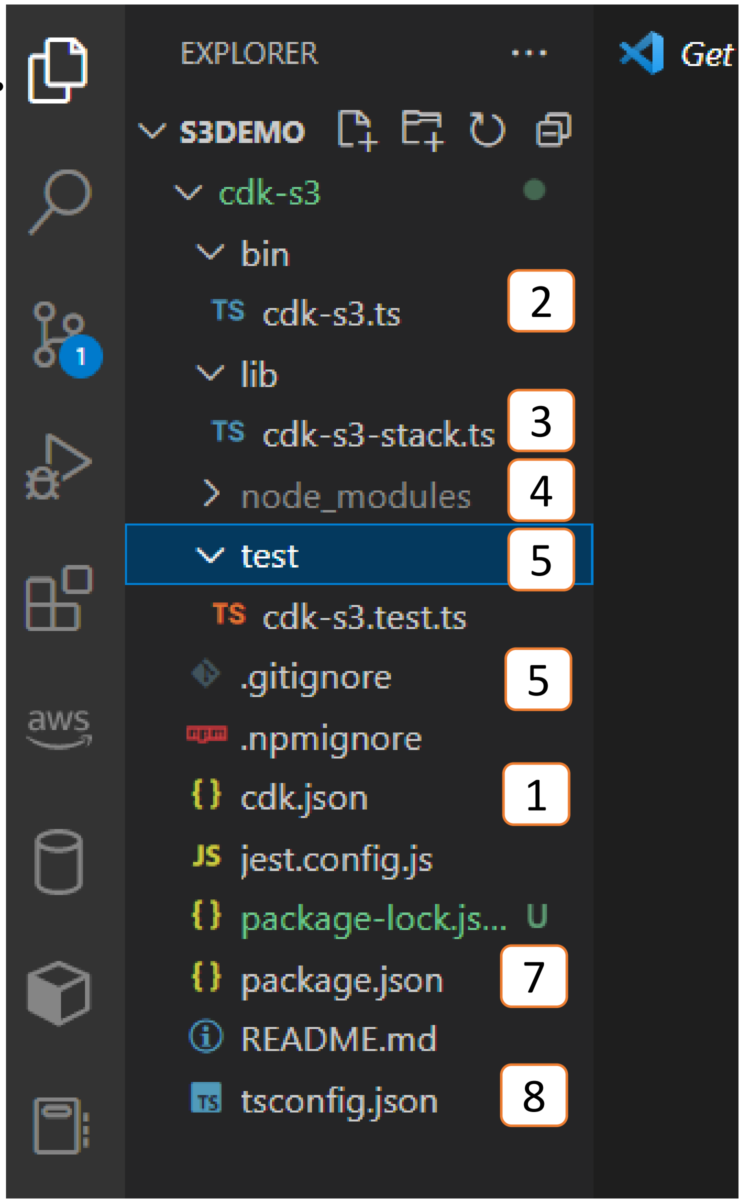
2. import 'from constructs' (created by default)

3. import the modules for AWS Services
   - **Refer to this link for documentation – Link**
   - *import * as s3 from 'aws-cdk-lib/aws- s3'*

4. CDK Stack

5. Add the parameters
All constructs take three parameters when they are initialized
   - **Scope** (will always be 'this')
   - **Logical ID** (Logical ID of the resource)
   - **Props** (Attributes)

AWS CDK – Your first AWS CDK app – S3 Bucket



```
EXPLORER ...
∨ S3DEMO
  ∨ cdk-s3
    ∨ bin
      TS cdk-s3.ts            2
    ∨ lib
      TS cdk-s3-stack.ts      3
    > node_modules            4
    ∨ test                    5
      TS cdk-s3.test.ts
    .gitignore                5
    .npmignore
    {} cdk.json               1
    JS jest.config.js
    {} package-lock.js...  U
    {} package.json           7
    README.md
    tsconfig.json             8
```

4. node_modules

- Is maintained by npm and includes all your project's dependencies.

5. test/cdk-s3.test.ts

- All the test files are included here

6. .gitignore and .npmignore

- Tells git and npm which files to include/exclude

7. package.json is your npm module manifest.

- It includes information like the name of your app, version, dependencies
- build scripts like "watch" and "build" (package-lock.json is maintained by npm)

8. tsconfig.json your project's typescript configuration

9. cdk.out CloudFormation template equivalent to our CDK stack

Section 4:

AWS Service Creation using CDK v2 –

S3, DynamoDB, IAM Role, Lambda & CloudWatch

AWS CDK – AWS Service Creation using AWS CDK v2

AWS Services to be created in this Section :

- AWS S3 Bucket
- AWS DynamoDB
- AWS IAM Role/ Lambda Execution Role
- AWS Lambda
- AWS Environment Variables – Account and Region
- AWS CloudWatch

Steps to create any AWS Resource using CDK v2

Step 1.

- Create a folder on the Local Drive
- Open the Folder in VSCode Editor
- Open ‘Terminal in the VSCode’

Steps to create any AWS Resource using CDK v2 1/4

Step 2.

Create the app

- Make a directory ***infra*** (or any other based on your choice)
    - *mkdir infra*
- Change directory with **c*d***
    - *cd infra*
- Initialize the app by using the ***cdk init*** command**.** Specify the programming language.
  - *cdk init app --language typescript*

Tip

Create separate Directories for Infrastructure Code and Application Code

Steps to create any AWS Resource using CDK v2

Step 3.

In the **lib/infra-stack.ts file,**

- **Import the modules/package (All or specific Construct from the Module)** for AWS Services being created
    - **(**Refer to this link for documentation **– Link)**
- ***import * as lambda from 'aws-cdk-lib/aws-lambda' (example)***

Step 4.

- **Define Scope, Logical ID and Props ((**Refer to this link for documentation **– Link)**
    - Scope = this
    - Logical ID – Logical ID Name (Different from Physical ID)
    - Props - Add the attributes to create the Resources - AWS documentation or Editor Code Complete

AWS CDK – Steps to create any AWS Resource using CDK 3/4

Step 5

Build the app (<u>Optional</u>)

Build(compile) after changing your code.

AWS CDK—the Toolkit implements it but a good practice to build manually to catch syntax and type errors.

- ***npm run build***

Step 6

Bootstrap (One Time) - Deploys the CDK Toolkit staging stack in S3 bucket

- ***cdk bootstrap***

AWS CDK – Steps to create any AWS Resource using CDK 4/4

Step 7

Synthesize an AWS CloudFormation template for the app

- ***cdk synth***

Step 8

Deploying the stack (Deploy the stack using AWS CloudFormation)

- ***cdk deploy***

AWS CDK – Modify and Destroy

Step 9

Modifying the app

The AWS CDK can update deployed resources after you modify your app

To see these changes, use the cdk diff command.

- ***cdk diff***

Step 10

Destroying the app's resources

- ***cdk destroy***

AWS CDK – AWS Service Creation Steps using AWS CDK v2

AWS S3 Bucket :

- Props(attributes) :
  - bucketName?
  - versioned?
  - publicReadAccess?
- Logical ID –
- Physical ID -

AWS CDK – AWS Service Creation steps using AWS CDK v2

AWS DynamoDB :

- Props :
  - readCapacity?
  - writeCapacity?
  - partitionKey
  - **tableName ? – New attribute**

AWS CDK – AWS Service Creation steps using AWS CDK v2

AWS IAM Role for Lambda(Lambda Execution Role):

- Props :
  - roleName?
  - description?
  - assumedBy
  - managedPolicies?

Summary of Steps to create any AWS Resource using CDK v2

Step 1. – Open the new folder in Visual Studio Code Editor and open Terminal

Step 2. – Create the app: Create Infra & Services Folder - ***mkdir infra, mkdir services and cd infra***

Step 3. – Initialize the CDK with ***cdk init app --language typescript***

Step 4. – *Import the module* for aws service being created - **Link**

Step 5. – *Define Scope, Logical ID and Props* – ***(this, 'logical id', {props})***

Step 6. – *Build the app* (Optional) with ***npm run build***

Step 7. – *Bootstrap* (One Time) with ***cdk bootstrap***

Step 8. – *Synthesize* an AWS CloudFormation template for the app with ***cdk synth***

Step 9. – *Deploying* the stack with ***cdk deploy***

AWS CDK – AWS Service Creation steps using AWS CDK v2

AWS Lambda:

- Props :
  - roleName?
  - Handler
  - Code
  - Runtime

AWS CDK v2 – Environment Variables

AWS Account and Region

- ***Account and Region Deployment based on Environment Variables***
    - env: { account: '123456789012', region: 'us-east-1' }
- ***Account and Region Deployment based on CLI configured Region***
    - Region that are implied by the current CLI configuration
    - env: { account: process.env.CDK_DEFAULT_ACCOUNT, region: process.env.CDK_DEFAULT_REGION }

AWS CDK – AWS Service Creation steps using AWS CDK

CloudWatch Alarm for Lambda:

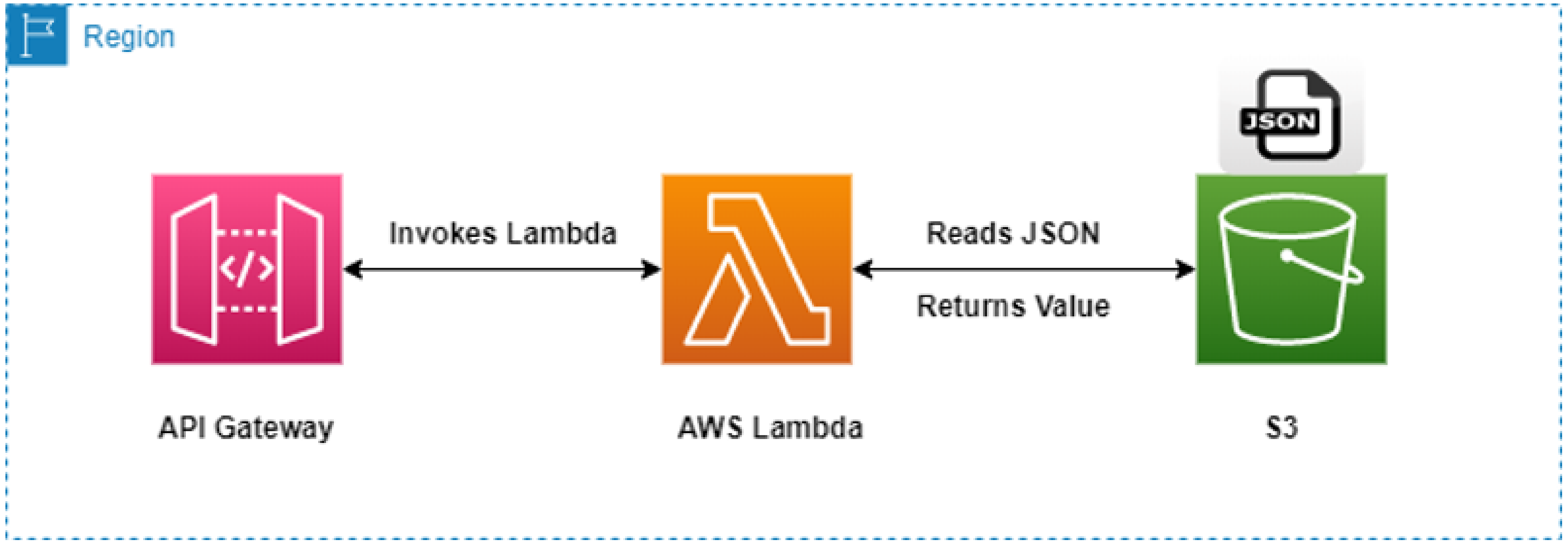
- Props :
  - evaluationPeriods
  - threshold
  - alarmName?
  - metric – metricErrors()

Section 5:

Use Case 1 - API Gateway, Lambda & S3

AWS Cloud Development Kit (CDK) v2 – Serverless Use Case

Serverless Use Case - using API Gateway, AWS Lambda and S3



- ***S3***
- ***IAM Role***
- ***AWS Lambda***
- ***API Gateway***

AWS CDK v2 – Serverless Use Case

Serverless Use Case-using API Gateway, AWS Lambda and S3 (Balance Status Application)

1. S3

- *bucketName – 'balanceStatus-0125'*

2. IAM Role

- *roleName*
- *assumedBy*
- *description*
- *IAM Policy attached to Role -* **AmazonS3FullAccess**

3. AWS Lambda

- *handler -* **lambda_function.lambda_handler**
- *role*
- *code*
- *runtime*
- *LambdaCode file –* ***lambda_function.py and Method – lambda_handler***

AWS CDK v2 – Serverless Use Case

Serverless Use Case - using API Gateway, AWS Lambda and S3

4. API Gateway

- *handler*
- *restApiName*
- *proxy*
- *deploy*

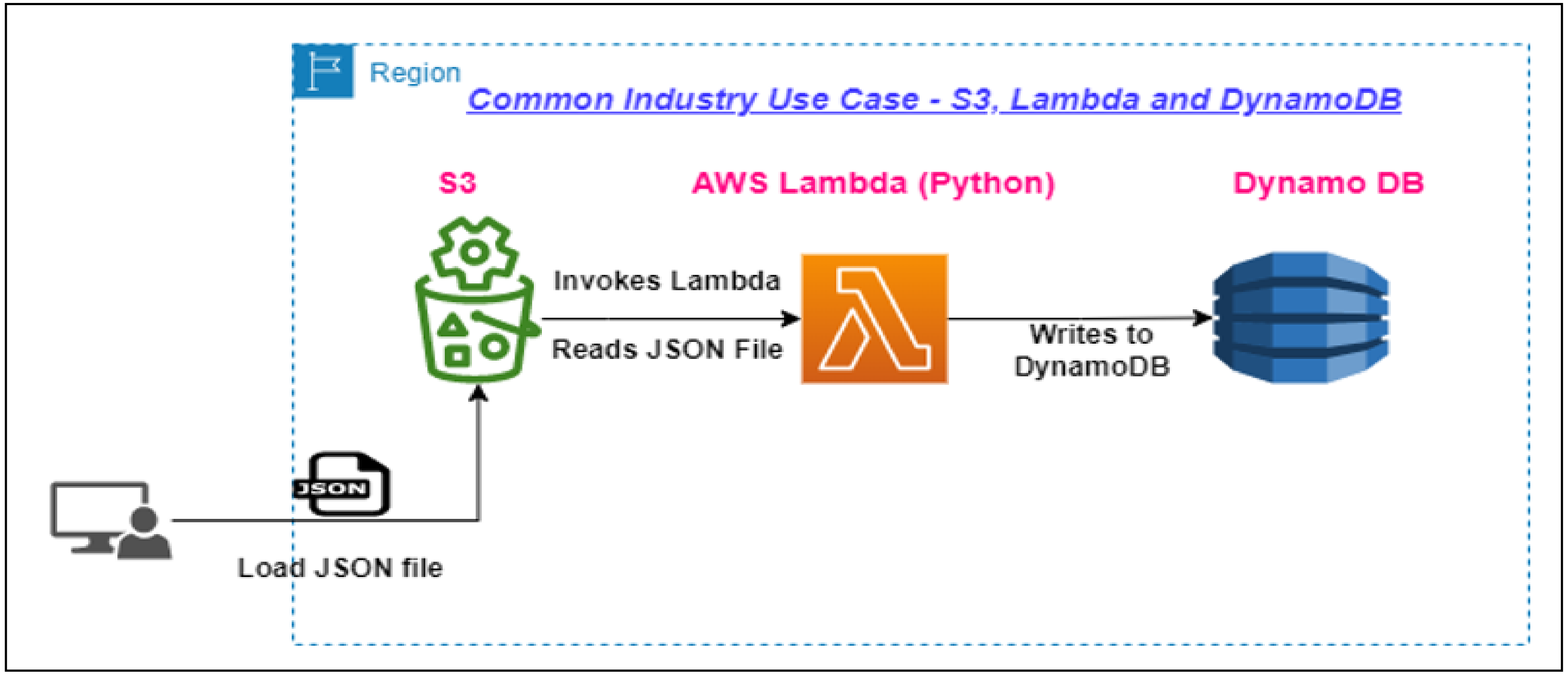
- *Resource - balanceStatus*
- *Method – GET*
- *Construct - LambdaRestAPI*

Section 6:

Use Case 2 - API Gateway, Lambda & S3

AWS CDK v2 – Serverless Use Case

Serverless Use Case - using S3, AWS Lambda and DynamoDB



- *IAM Role*
- *S3*
- *S3 Event Notification*
- *AWS Lambda*
- *DynamoDB*

Summary of Steps to create any AWS Resource using CDK v2

Step 1. – Open the new folder in Visual Studio Code Editor and open Terminal

Step 2. – Create the app: Create Infra & Services Folder - ***mkdir infra, mkdir services and cd infra***

Step 3. – Initialize the CDK with ***cdk init app --language typescript***

Step 4. – *Import the module* for aws service being created - **Link**

Step 5. – *Define Scope, Logical ID and Props* – ***(this, 'logical id', {props})***

Step 6. – *Build the app* (Optional) with ***npm run build***

Step 7. – *Bootstrap* (One Time) with ***cdk bootstrap***

Step 8. – *Synthesize* an AWS CloudFormation template for the app with ***cdk synth***

Step 9. – *Deploying* the stack with ***cdk deploy***

AWS CDK v2 – Serverless Use Case

Serverless Use Case - using S3, AWS Lambda and DynamoDB (Retail Inventory Feed)

1. IAM Role

- *roleName – inventoryfeed01role*
- *assumedBy*
- *description*
- *IAM Policy attached to Role -* **AmazonS3FullAccess, AmazonDynamoDBFullAccess and CloudWatchFullAccess**

2. AWS Lambda

- *handler -* **lambda_function.lambda_handler**
- *role*
- *code*
- *runtime*

- *Add dependency on IAM Role - stackA.node.addDependency(stackB) method*
- *LambdaCode file –* ***lambda_function.py and Method – lambda_handler***

AWS CDK v2 – Serverless Use Case

Serverless Use Case 1 - using S3, AWS Lambda and DynamoDB (Retail Inventory Feed)

3. S3 Bucket

- *bucketName – 'inventoryfeeds3bucket01'*

4. S3 Event Notification

- *Add following method* - bucket.addEventNotification(s3.EventType.OBJECT_CREATED, **new** s3n.LambdaDestination(fn));

AWS CDK v2 – Serverless Use Case

Serverless Use Case - using S3, AWS Lambda and DynamoDB (Retail Inventory Feed)

5. AWS DynamoDB

- *partitionKey – customername (String)*
- *tableName - inventoryfeedynamodb01*

Section 7:

CI-CD Pipeline : Creating and Deploying AWS CDK Apps using CI-CD Pipeline

AWS CDK v2 – Deploying AWS Services using CI-CD

Serverless Use Case - using S3, AWS Lambda and DynamoDB (Retail Inventory Feed)

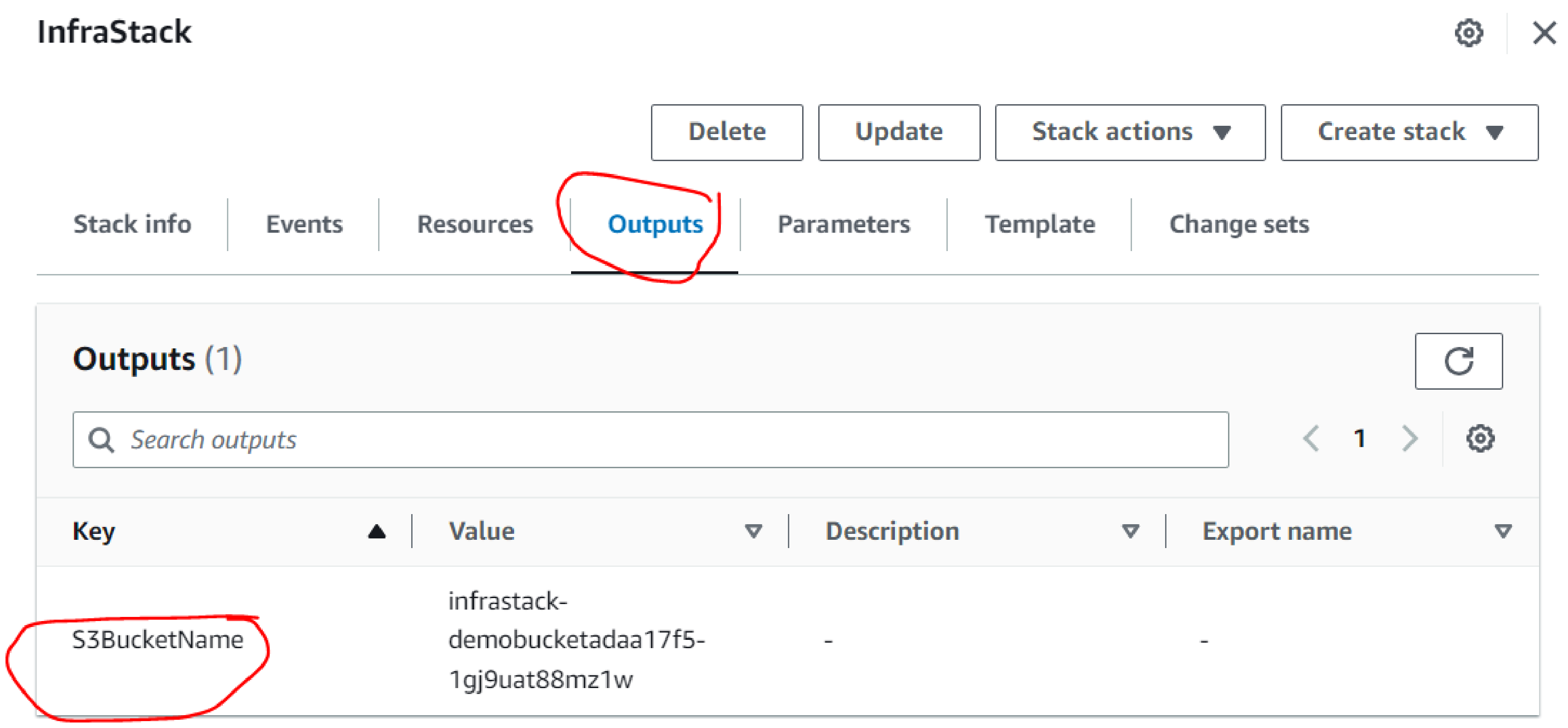
Refer to pdf uploaded in the relevant section for commands

Section 8:

Additional CDK Concepts

AWS CDK – CfnOutput

Creates an CfnOutput value for this stack.



Key	Value	Description	Export name
S3BucketName	infrastack-demobucketadaa17f5-1gj9uat88mz1w	-	-

AWS CDK – Main Commands

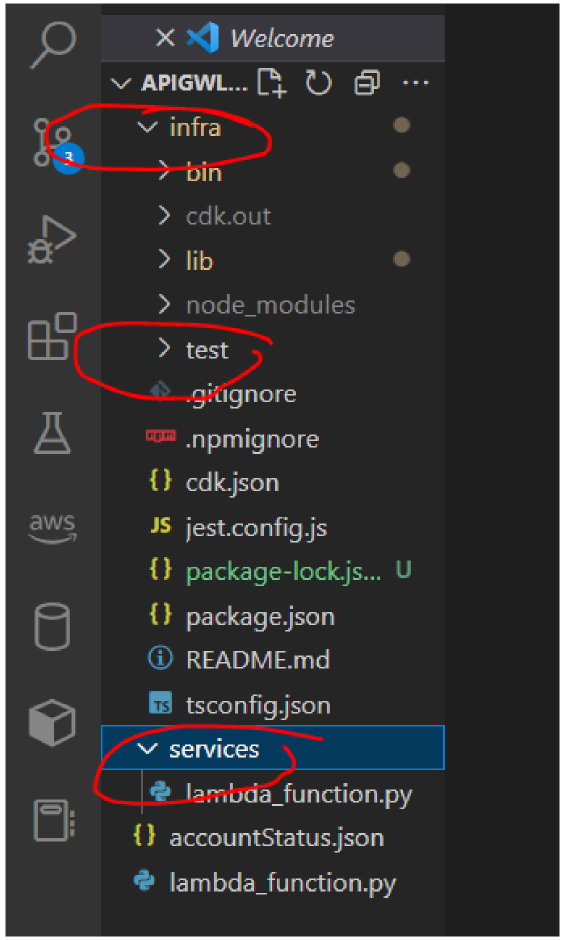
#	Command	Description
1	*cdk init app --language typescript*	Initialize the app
2	*npm run build*	Build the app (Optional)
3	*cdk bootstrap*	Bootstrap (One Time) - Deploys the CDK Toolkit staging stack
4	*cdk synth*	Synthesize an AWS CloudFormation template for the app
5	*dk deploy*	Deploying the stack - Deploys one or more specified stacks
6	*cdk diff*	Compares the specified stack and its dependencies with the deployed stacks or a local CloudFormation template
7	*cdk docs (doc)*	Opens the CDK API Reference in your browser
8	*cdk list (ls)*	Lists the stacks in the app
9	*cdk metadata [Stackname]*	Displays metadata about the specified stack
10	*cdk deploy - - hotswap*	--hotswap flag with cdk deploy to update AWS resources directly instead of generating an AWS CloudFormation changeset and deploying it (Lambda, ECS, Step Functions)
11	*cdk destroy*	Destroys one or more specified stacks

AWS CDK – Multi-Stack

- AWS CDK can help create apps containing any number of stacks.
- Each stack results in its own AWS CloudFormation template.
- Each stack in an app can be synthesized & deployed individually using the cdk deploy
  - cdk deploy --all
  - cdk deploy [stackname]

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

1. Separate the Infrastructure and Application Code into separate folders



Welcome
APIGWL...
infra
bin
cdk.out
lib
node_modules
test
.gitignore
.npmignore
cdk.json
jest.config.js
package-lock.js... U
package.json
README.md
tsconfig.json
services
lambda_function.py
accountStatus.json
lambda_function.py

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

2. Single or Multi Stacks for an end to end application

- Separate out the **sensitive AWS Services** such as IAM Role, Security Group and NACL in a separate Repo
- **Rest of the AWS Services** go into a separate repo
- Build a **separate stack** for sensitive services
- Rest of the services can be deployed as **single** or multiple **stacks**
- AWS recommends keeping **stateful resources (like databases) in a separate stack** from stateless resources.
    - Turn on **termination protection** on the stateful stack.
    - Can freely destroy or create multiple copies of the stateless stack without risk of data loss.

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

3. Resource Naming Convention – AWS generated or customized

- AWS usually recommends to **auto-generate physical names** such as S3 bucket, APIGW and other services
- However, sometimes it's a good practice to be able to **co-relate the AWS Service Name to business unit** and **application, stage etc.**

  Naming an API GW

- **'$ (business unit name)-$ (application name)-$(stage)- apigw**
- business unit name, app name, stage etc. **can be referenced from the configuration file as an environment variable**

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

4. Changing the logical ID of stateful resources can impact the service due to replacement

- Changing the logical ID of a resource results in the **resource being replaced** with a new one at the next deployment.
- For **stateful resources** like databases and S3 buckets, or **persistent infrastructure** like an Amazon VPC, this may cause **serious issues if resource is replaced.**
- Make sure **refactoring of your AWS CDK code** does not impact the logical ID.
- Write **unit tests** that assert that the logical IDs of your stateful resources remain static.

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

5. Resource Retention policies and Log Retention

- Define a retention policy for your Storage Services – S3, RDS , EFS etc. in each Environment
- S3 default retention policy is ‘Retain’
- CDK's default is to retain all logs forever

6. Application Deployment & CI-CD Pipeline is recommended to be in different AWS accounts

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

7. One repo across environments and deploy using the stage variable

- Create a single repository for your Infrastructure as Code and Application Code
- Deploy across the environments across the stages using the ‘stage’ variable in the configuration file

8. Use Secrets Manager and SSM for Storing Sensitive Values

- Use services like Secrets Manager and Systems Manager Parameter Store for sensitive values.
- Don’t check in to source control, using the names or ARNs of those resources.

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

9. Custom constructs based on architecture patterns aligning to business domains

- Large Organizations create their own pattern to encapsulate all the resources and their default values inside a single higher-level L3 construct that can be shared.
- It can range from a simple rapper around creation of encrypted bucket or an architecture pattern
- These patterns help provision multiple resources based on common patterns with a limited knowledge in a precise manner at speed.

AWS CDK - 10 Best Practices based on my Cloud Migration Experience

10. Measure everything

- Measure all aspects of your deployed resources, create metrics, alarms, and dashboards.
- Use CloudWatch, ELK/OpenSearch

Thank You