

AWS Cloud Development Kit (CDK) v2

Absolute Beginner to Advanced



AWS CDK v2 – Beginner to Advanced

6 Months on....

1236+ Learners

154+ Ratings



Sayyed A.

★★★★★ 3 weeks ago



The course is very helpful and the instructor has made sure that he has explained every topic in detail. Worth buying this course, Loved it!!



Brajesh K.

★★★★★ 2 months ago



Best Course on CDK on udemy , i am now more confident on writing cdk code with typescript and creating lambda and api server less resources.

very clearly described lambda and use cases.

Must watch for aws cdk with typescript hands-on knowledge.



Anish G.

★★★★★ 2 months ago

Excellent course...gives a very good start on using CDK V2 concepts handson



Tharan G.

★★★★★ 2 weeks ago



It was a pleasure to follow this course materials.... Real world example, and how the instructor leads into new topics. Gradually building and iterative approach is very well explained.



Gnaneswara R.

★★★★★ 3 weeks ago

High level teaching ,and very clear ' systematic way of teaching which can understand by a even non IT people



Raj D.

★★★★★ 2 months ago



It's a good course to understand CDK with lots of hands on scenarios. I really liked the last lesson on Industry best practices. Please add some more content to that section.

Mail me on trisalrahul@gmail.com for **Udemy Discount Coupons** for my other best selling courses on

AWS Lambda/Serverless, AWS CloudFormation and AWS Solution Architect Professional

Connect with me and share your AWS Learning Experiences

- I am **Rahul Trisal** working as an **AWS Community Builder** and Solution Architect in a Fortune 500 Organization
- Co-Founder of a AWS focused Startup – TechCloudByte – www.techcloudbyte.com
- 7X Cloud Certified – 5X AWS
 - **AWS Solution Architect – Professional**
 - AWS Solution Architect – Associate
 - AWS Certified SysOps
 - AWS Certified Developer
 - AWS Cloud Practitioner
 - Azure Fundamental
 - IBM Bluemix Developer
- 1000+ applications migrated working with Fortune 100 customers on large AWS Cloud Migration Programs



I would love to Connect with you and together learn AWS. Connect with me at below links:

- Instagram - https://www.instagram.com/aws_with_rahul/
- LinkedIn - <https://www.linkedin.com/in/rahul-trisal-7709628/>
- Youtube - <https://www.youtube.com/@trisalrahul/videos>
- AWS Startup - www.techcloudbyte.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...*

4. Install the AWS CDK Toolkit for VS Code :

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

<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

 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](#)

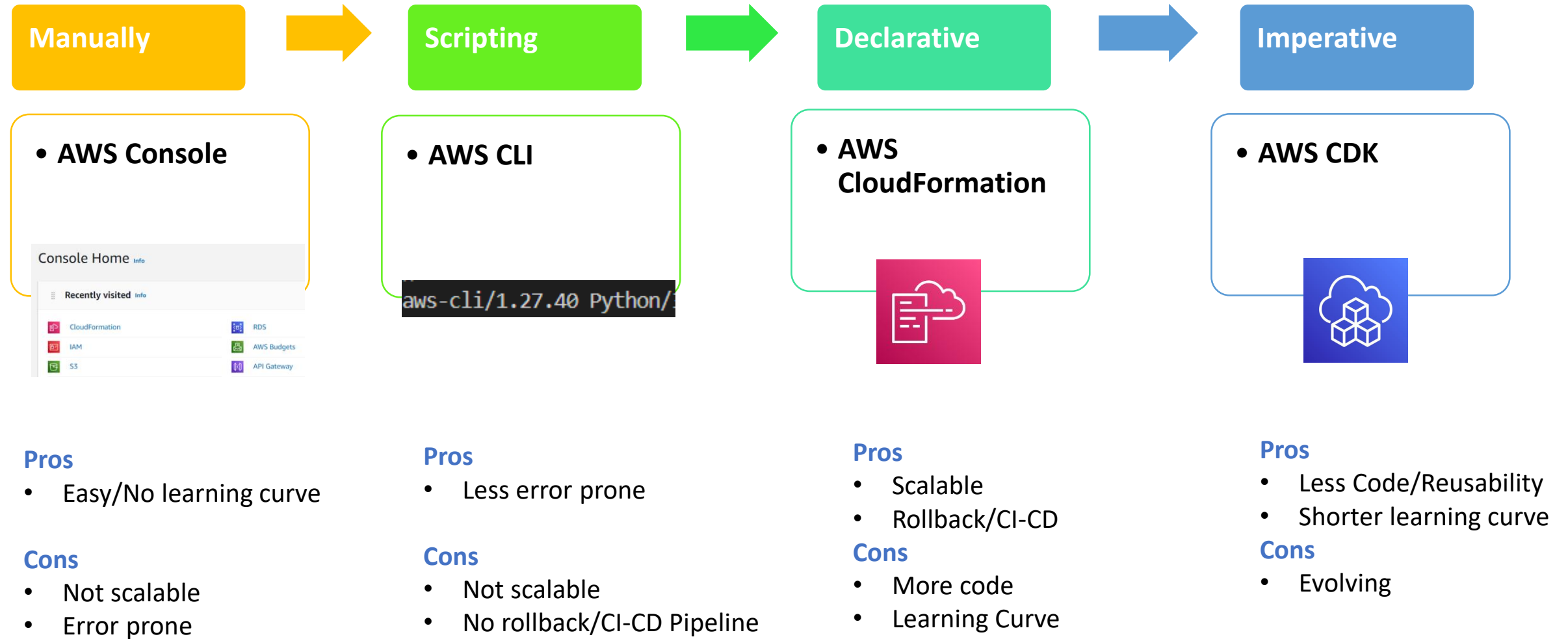
Section 3:

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



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

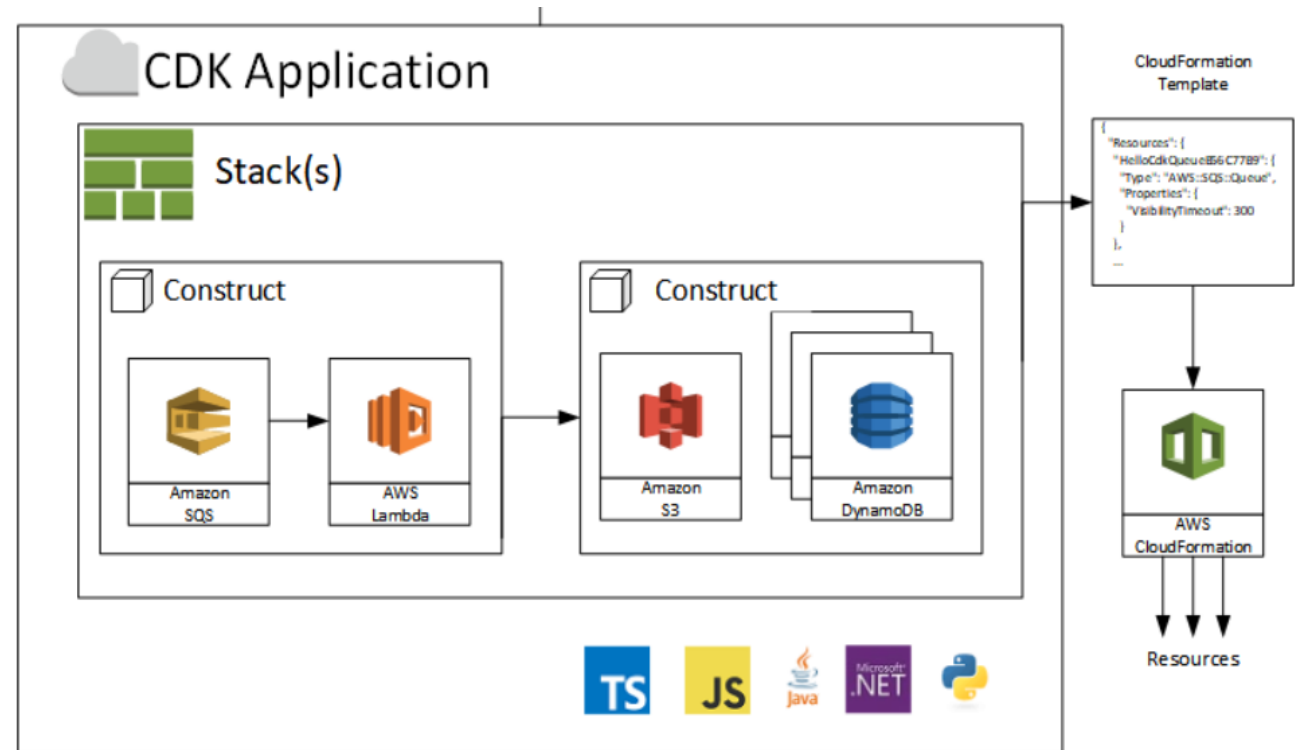
```
1  ---
2  AWSTemplateFormatVersion: "2010-09-09"
3  Description:
4    This is the first cloudformation template to
5    create S3 bucket and EC2 instance
6  Resources:
7    RahulS3bucket:
8      Type: AWS::S3::Bucket
9      Properties:
10       BucketName: demobucket09070
11    RahulEC2Instance:
12      Type: AWS::EC2::Instance
13      Properties:
14       ImageId: "ami-05fa00d4c63e32376"
15       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.**



source : aws

AWS CDK— Benefits

1. Write much less code

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

AWS CDK -15 lines of code

CloudFormation – 500+ Lines of code

- 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

TypeScript JavaScript Python Java C# Go

```
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
    });

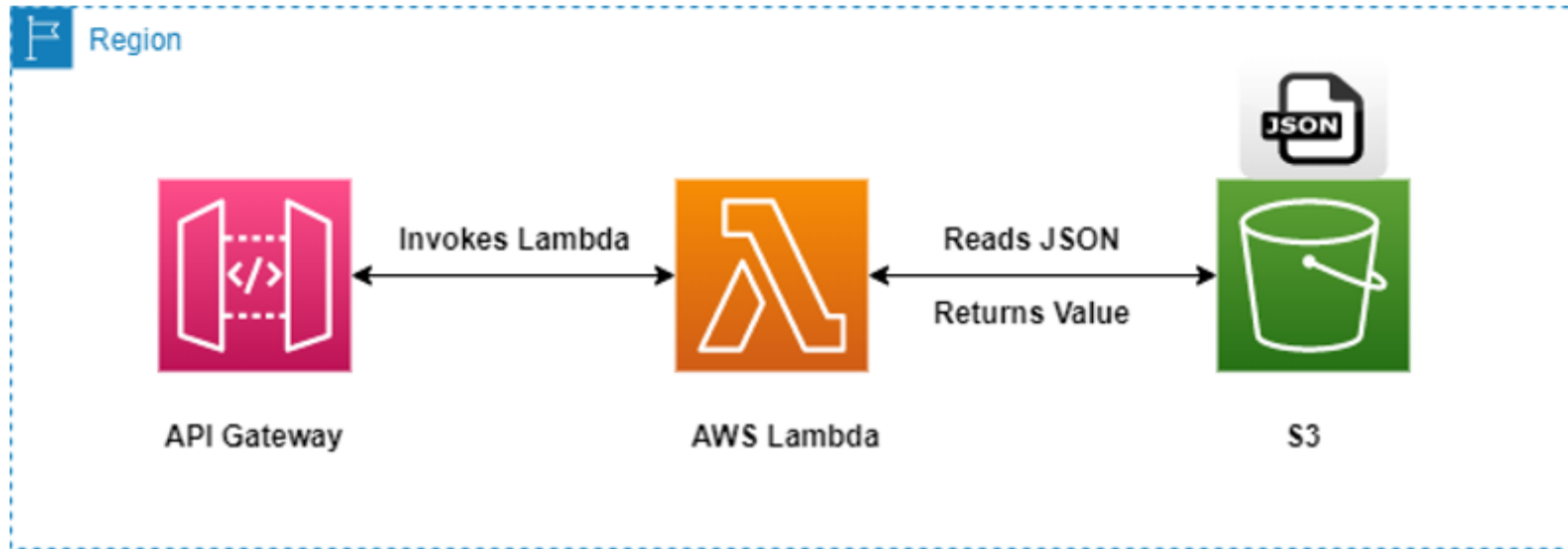
    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService", {
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample")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

515 lines (515 sloc) 15.8 KB

```
1 Resources:
2   MyVpcF9F0CA6F:
3     Type: AWS::EC2::VPC
4     Properties:
5       CidrBlock: 10.0.0.0/16
6       EnableDnsHostnames: true
7       EnableDnsSupport: true
8       InstanceTenancy: default
9       Tags:
10        - Key: Name
11          Value: MyEcsConstruct/MyVpc
12       Metadata:
13         aws:cdk:path: MyEcsConstruct/MyVpc/Resource
14   MyVpcPublicSubnet1SubnetF6608456:
15     Type: AWS::EC2::Subnet
16     Properties:
17       CidrBlock: 10.0.0.0/18
18       VpcId:
19         Ref: MyVpcF9F0CA6F
20       AvailabilityZone:
21         Fn::Select:
22           - 0
23           - Fn::GetAZs: ""
24       MapPublicIpOnLaunch: true
25       Tags:
26        - Key: Name
27          Value: MyEcsConstruct/MyVpc/PublicSubnet1
28        - Key: aws-cdk:subnet-name
29          Value: Public
30        - Key: aws-cdk:subnet-type
```

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

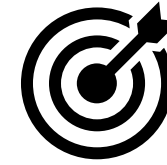
Serverless Use Case - using API Gateway, AWS Lambda and S3



- 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.



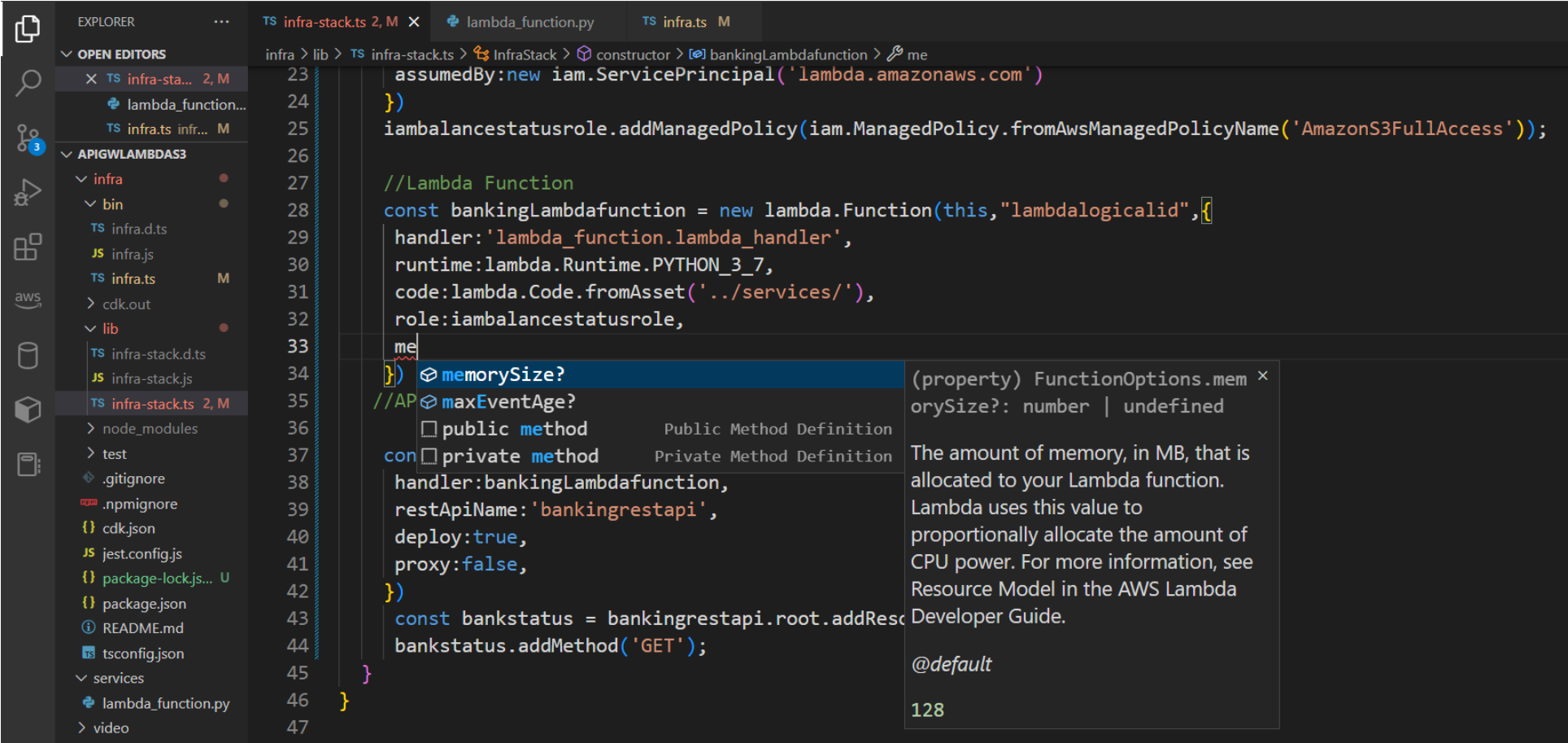
- No Need to learn JSON or YAML

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



AWS CDK— Benefits

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



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like `infra-stack.ts`, `lambda_function.py`, and `infra.ts`. The code editor shows the `infra.ts` file with the following code:

```
23 assumedBy:new iam.ServicePrincipal('lambda.amazonaws.com')
24 })
25 iambalancestatusrole.addManagedPolicy(iam.ManagedPolicy.fromAwsManagedPolicyName('AmazonS3FullAccess'));
26
27 //Lambda Function
28 const bankingLambdafunction = new lambda.Function(this,"lambdaLogicalid",{
29   handler:'lambda_function.lambda_handler',
30   runtime:lambda.Runtime.PYTHON_3_7,
31   code:lambda.Code.fromAsset('../services/'),
32   role:iambalancestatusrole,
33   me
34 })
35 //AP
36 //AP
37 const bankstatus = bankingrestapi.root.addResourceFunction('GET',{
38   handler:bankingLambdafunction,
39   restApiName:'bankingrestapi',
40   deploy:true,
41   proxy:false,
42 })
43 const bankstatus = bankingrestapi.root.addResourceFunction('GET',{
44   handler:bankingLambdafunction,
45   restApiName:'bankingrestapi',
46   deploy:true,
47   proxy:false,
```

The code completion dropdown menu is open, showing the following options:

- `memorySize?` (property) FunctionOptions.memorySize?: number | undefined
- `maxEventAge?` (property) FunctionOptions.maxEventAge?: number | undefined
- `public method` Public Method Definition
- `private method` Private Method Definition

The tooltip for the `memorySize?` property is displayed, showing the following text:

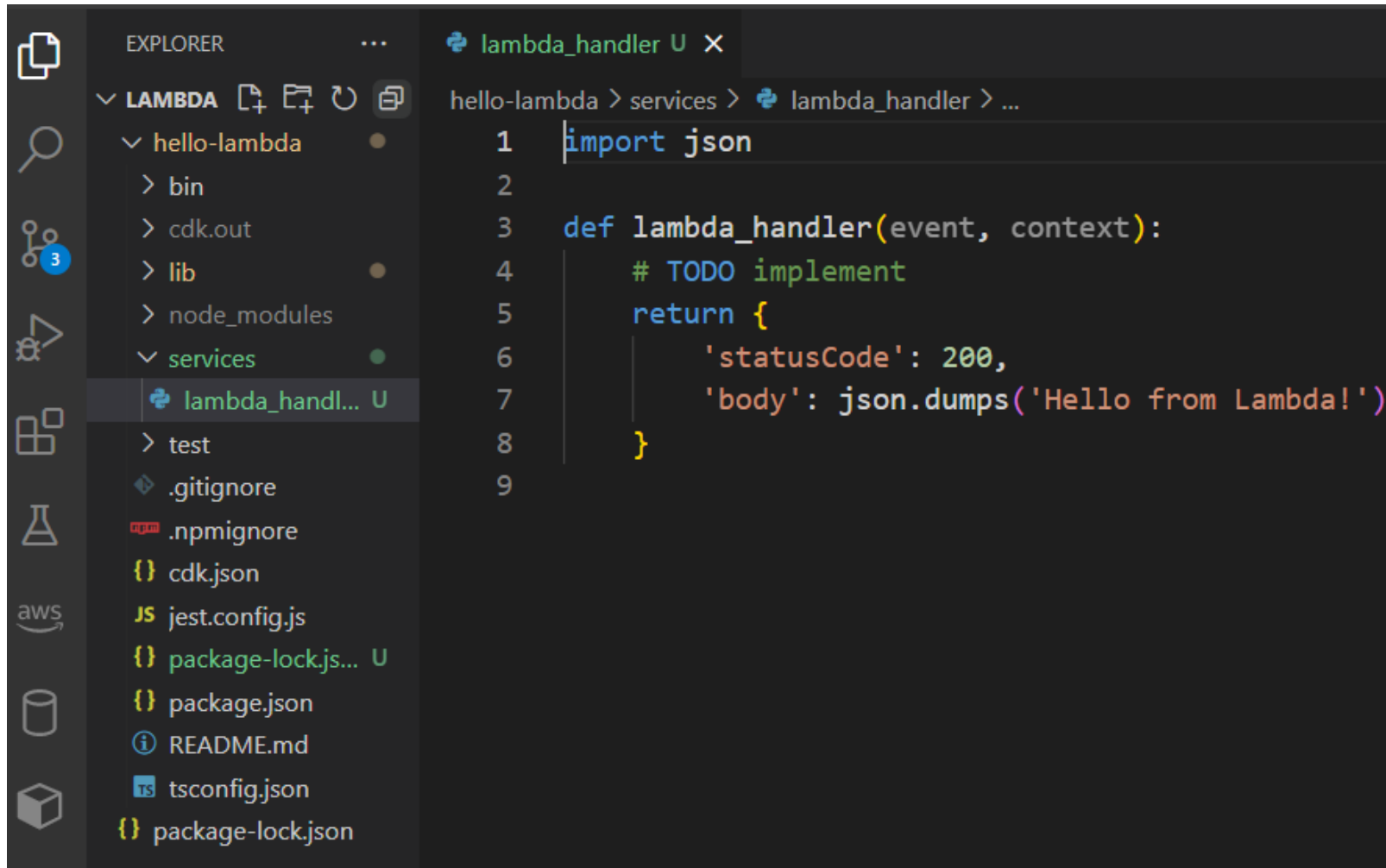
The amount of memory, in MB, that is allocated to your Lambda function. Lambda uses this value to proportionally allocate the amount of CPU power. For more information, see Resource Model in the AWS Lambda Developer Guide.

`@default`

`128`

AWS CDK— Benefits

4. Ability to execute application code with the infrastructure code



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree for a project named 'hello-lambda'. The tree includes a 'services' folder, which is expanded to show a file named 'lambda_handler... U'. The main editor area shows the contents of this file, which is a Python script. The script starts with 'import json' on line 1. On line 3, a function 'lambda_handler' is defined, taking 'event' and 'context' as arguments. The function body, starting on line 4, contains a comment '# TODO implement', followed by a 'return' statement on line 5. The return value is a dictionary with 'statusCode' set to 200 (line 6) and 'body' set to 'Hello from Lambda!' (line 7), which is converted to a JSON string using 'json.dumps'. The function ends with a closing brace on line 8. Line 9 is empty. The breadcrumb at the top of the editor reads 'hello-lambda > services > lambda_handler > ...'.

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

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



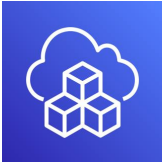
- **Don't need to configure every** attribute, Instead provided "**sensible defaults**" to easily spin up a resource.

L3 construct

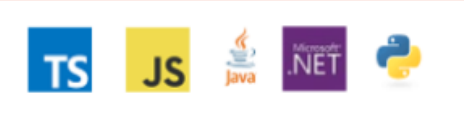

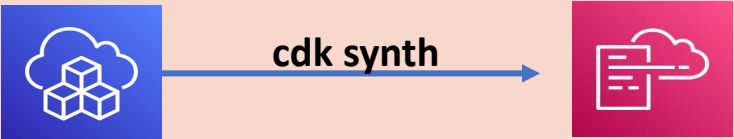


- A Level 3 construct represents various cloud resources that work together to accomplish a particular task called "**patterns**".
- **ApplicationLoadBalancedFargateService construct** will create an ECS cluster with Fargate, an ECR repository and ALB etc.

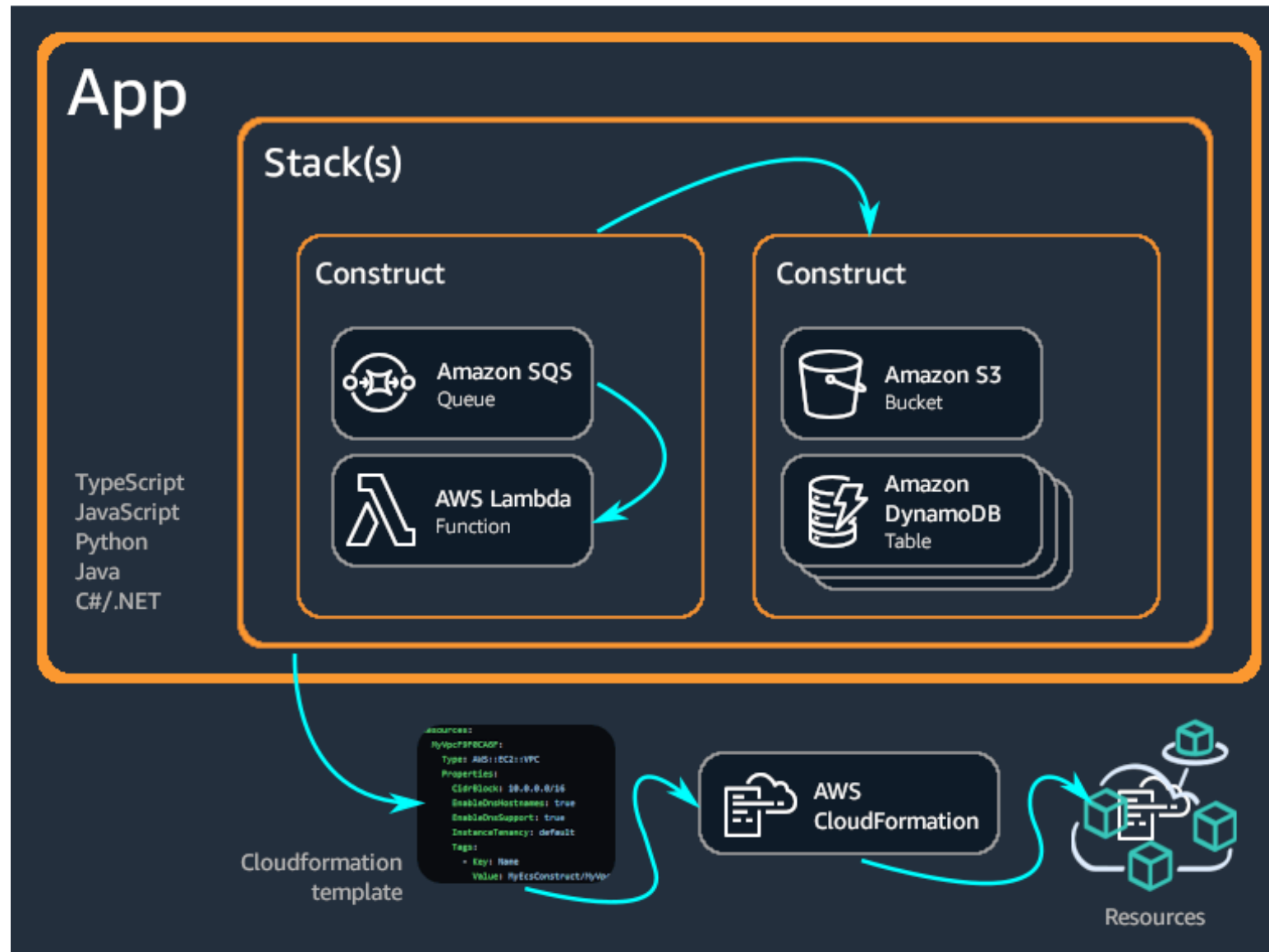
AWS CDK and AWS CloudFormation



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.	

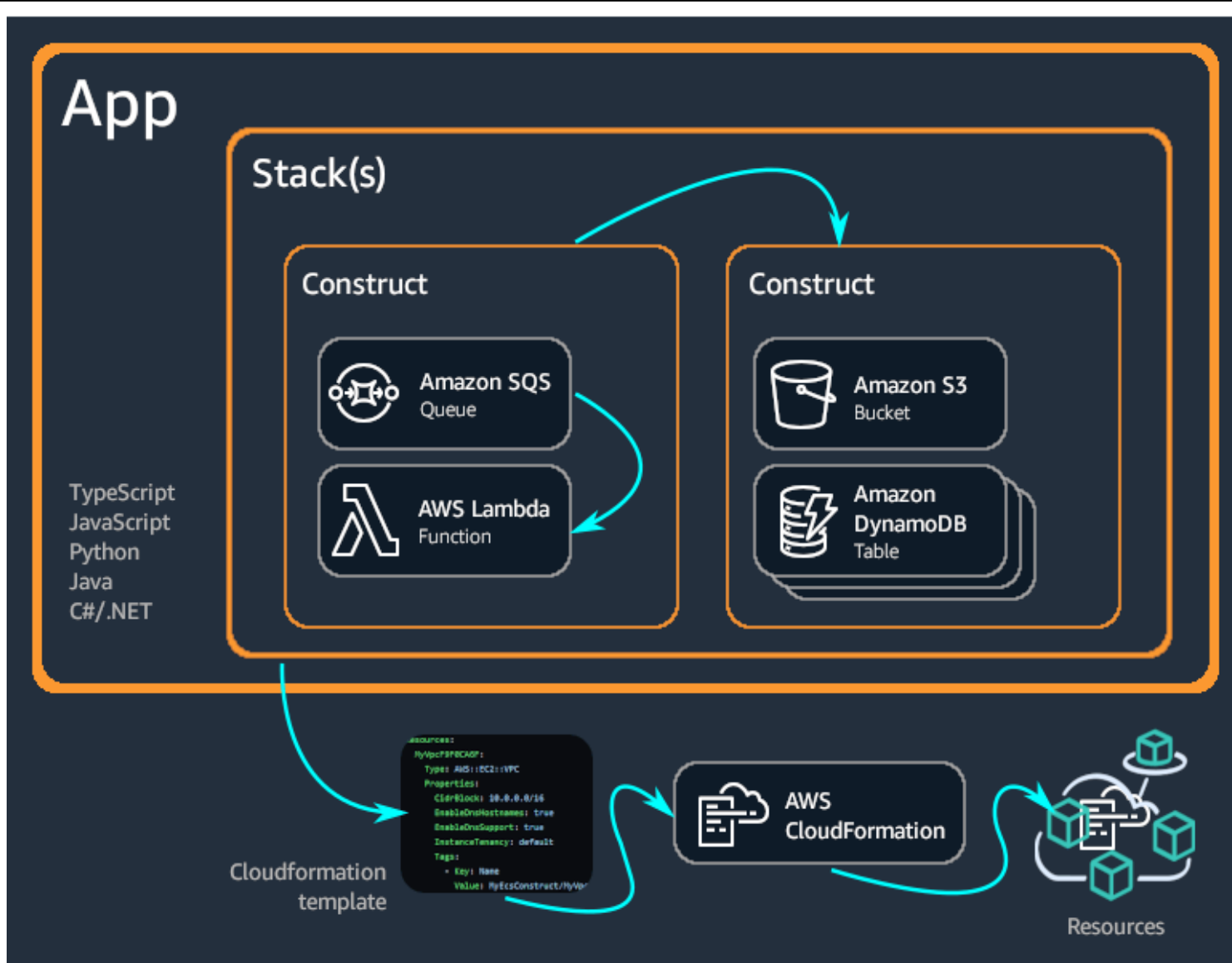
AWS CDK— Key Concepts



AWS CDK

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

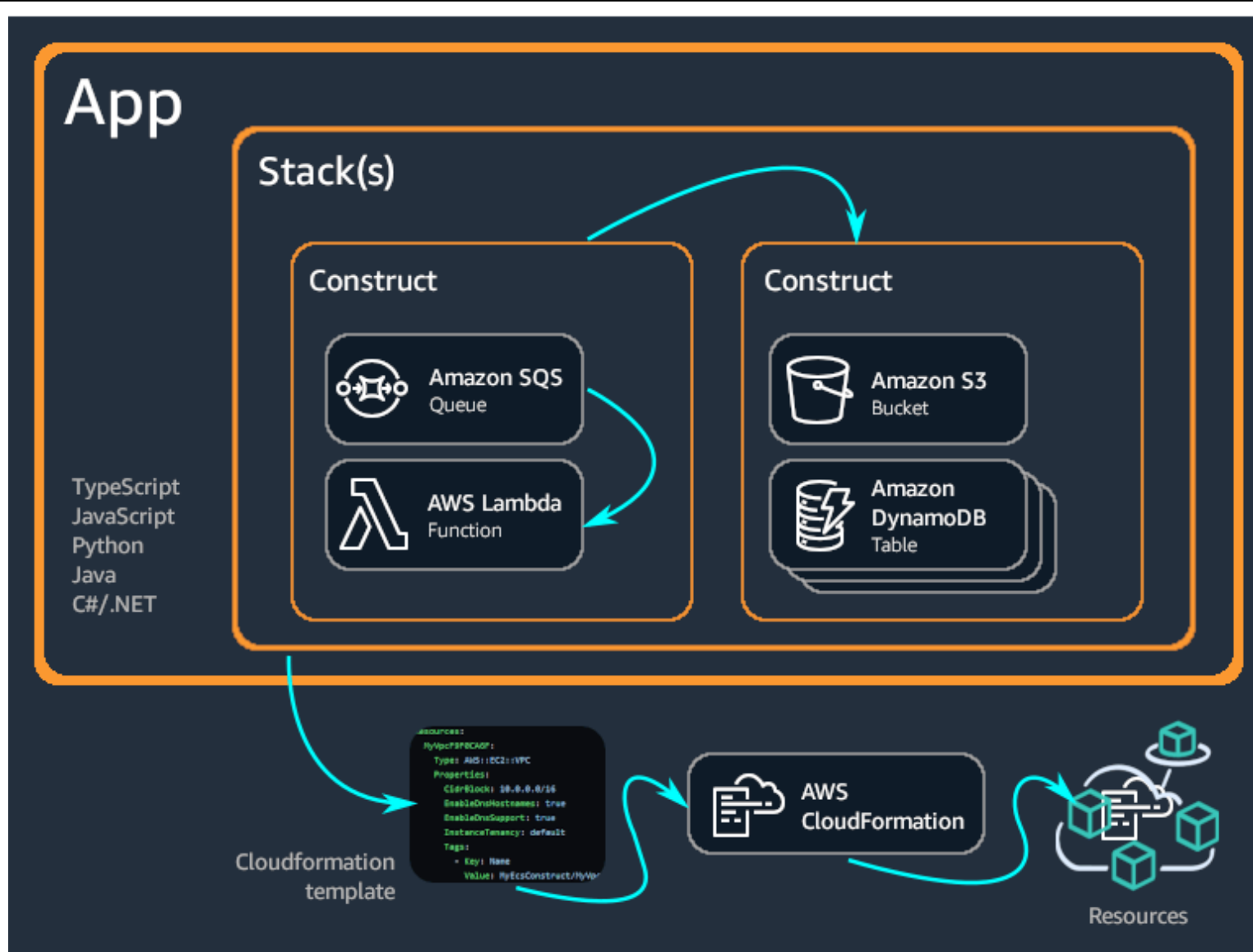
AWS CDK— Key Concepts



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

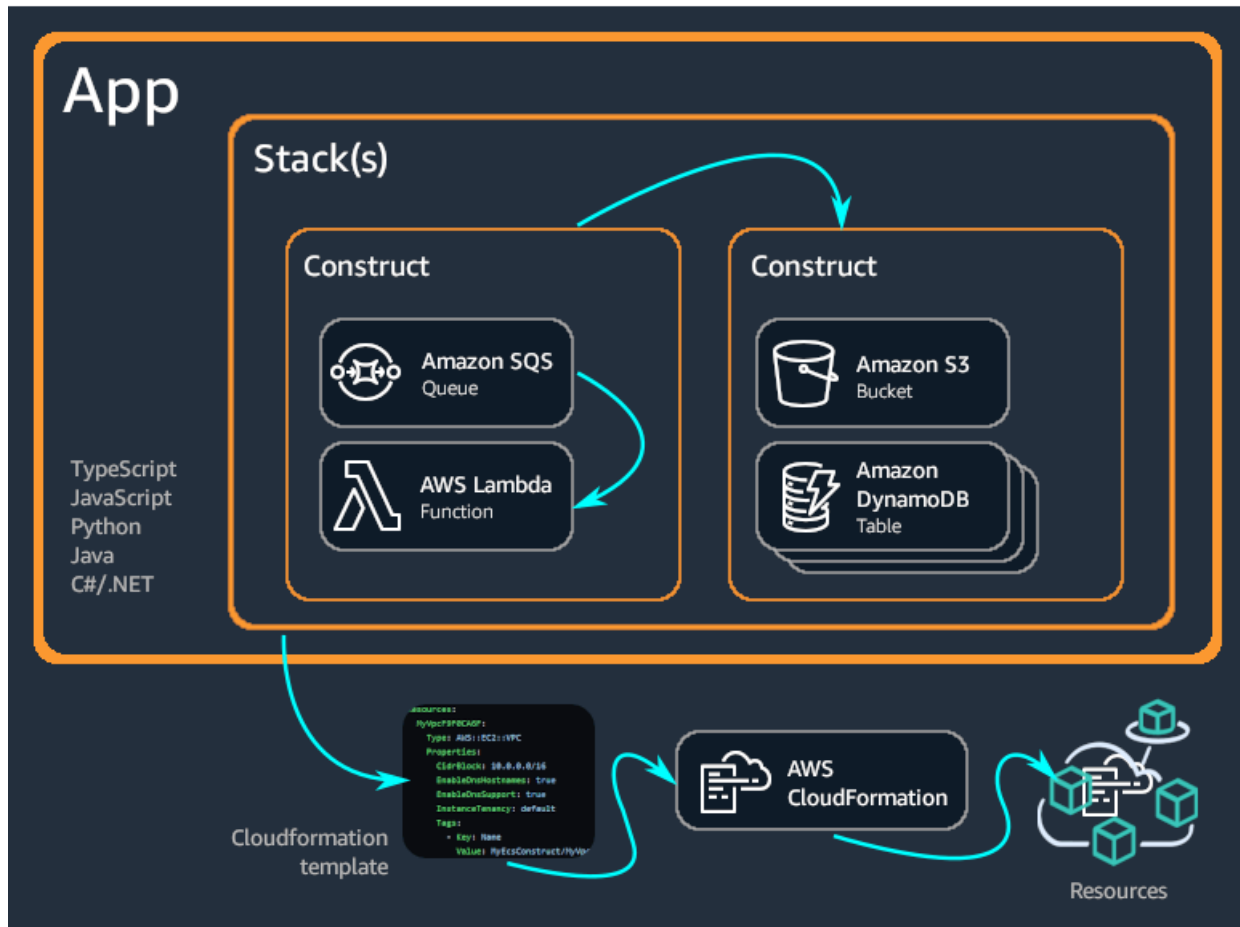
AWS CDK— Key Concepts



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



3. Constructs

- Constructs are **basic building blocks of AWS CDK apps**.
- 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.

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.



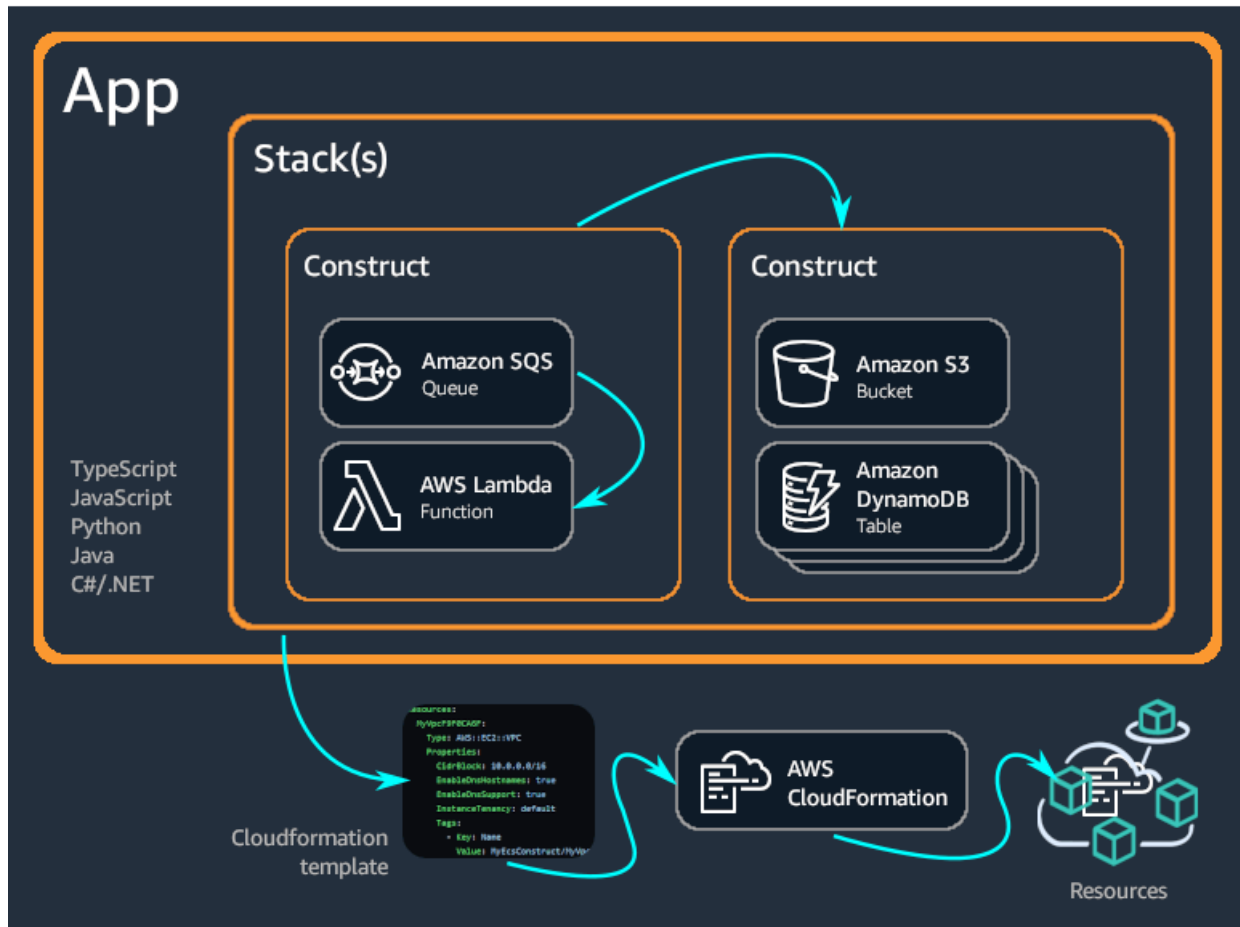
L3 construct

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

Example - **ApplicationLoadBalancedFargateService 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

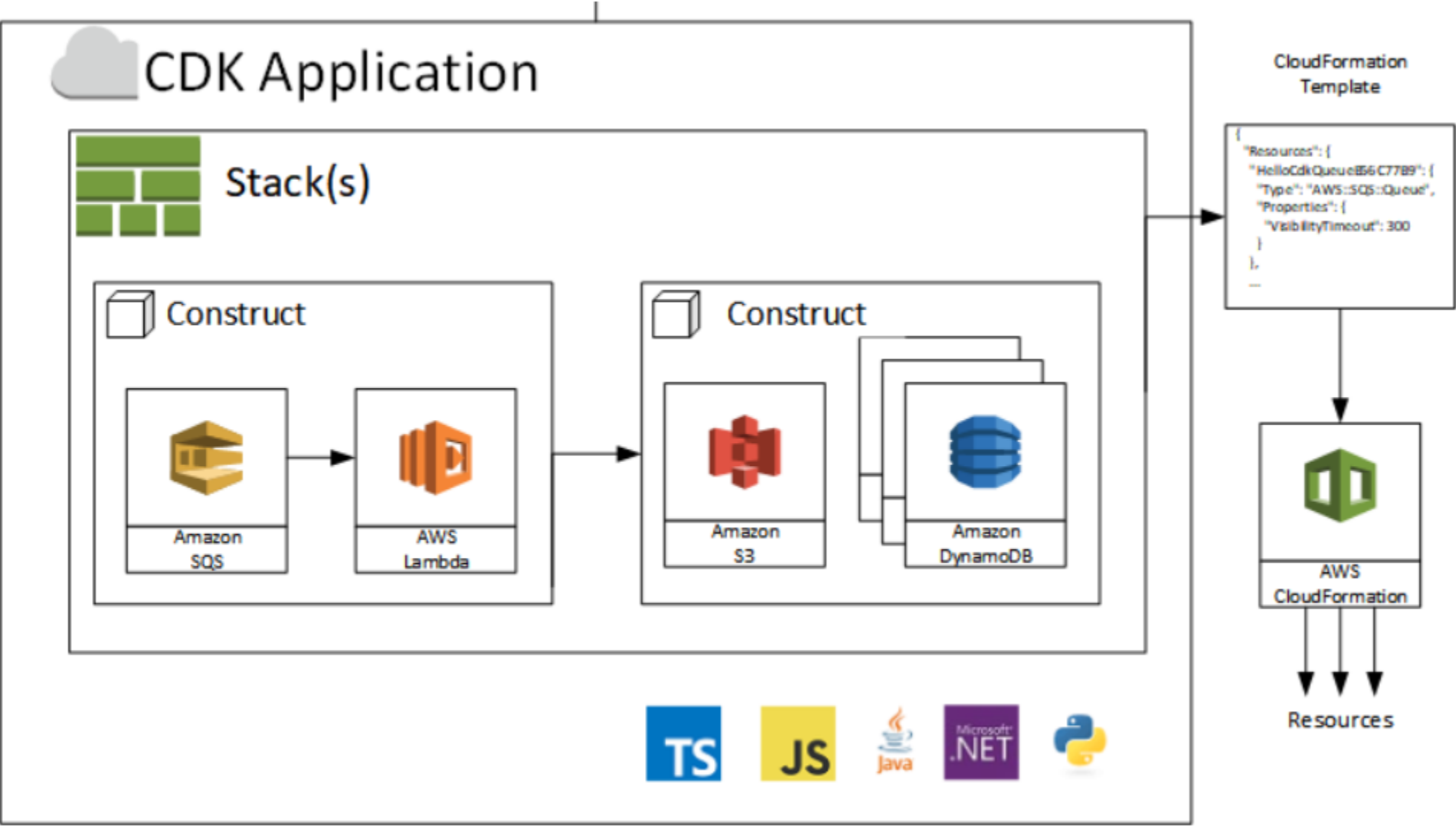


4. Resources

- 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.

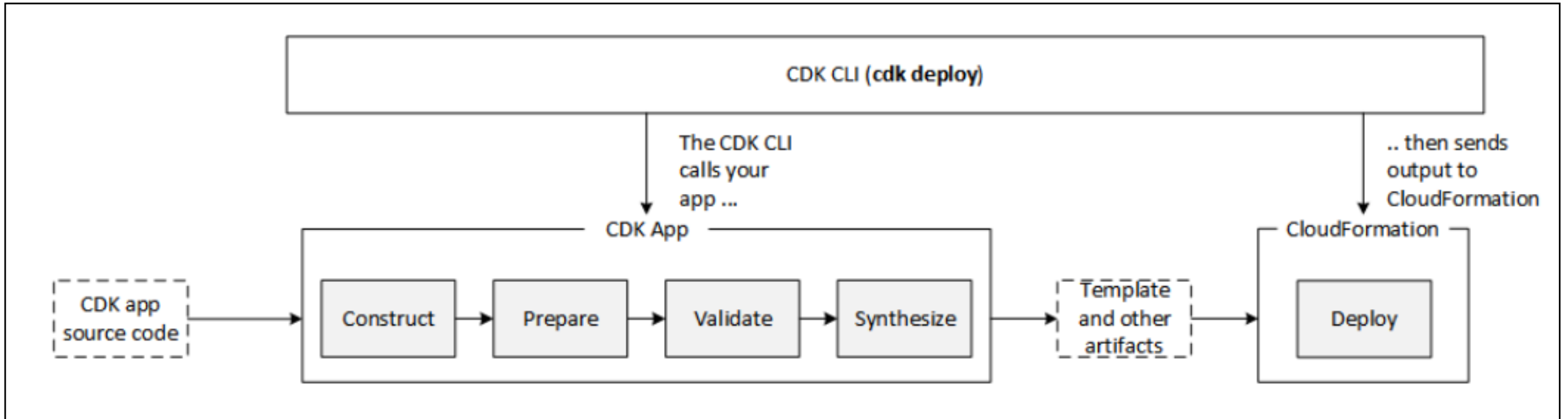
TypeScript	JavaScript	Python	Java	C#
<pre>import * as sqs from '@aws-cdk/aws-sqs'; new sqs.Queue(this, 'MyQueue', { encryption: sqs.QueueEncryption.KMS_MANAGED });</pre>				

AWS CDK— Key Concepts



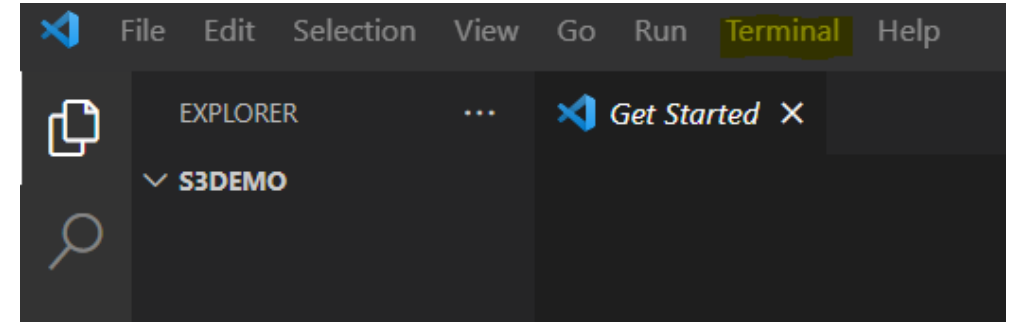
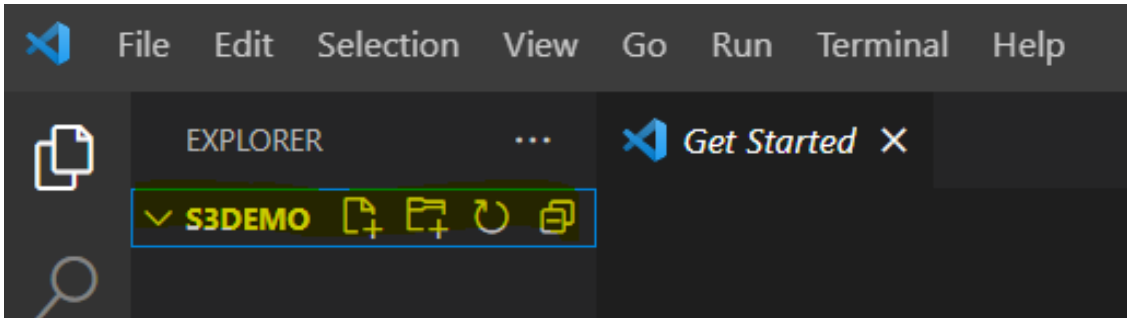
AWS CDK— How does AWS CDK work ?

How does AWS CDK work – Execution Steps



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*

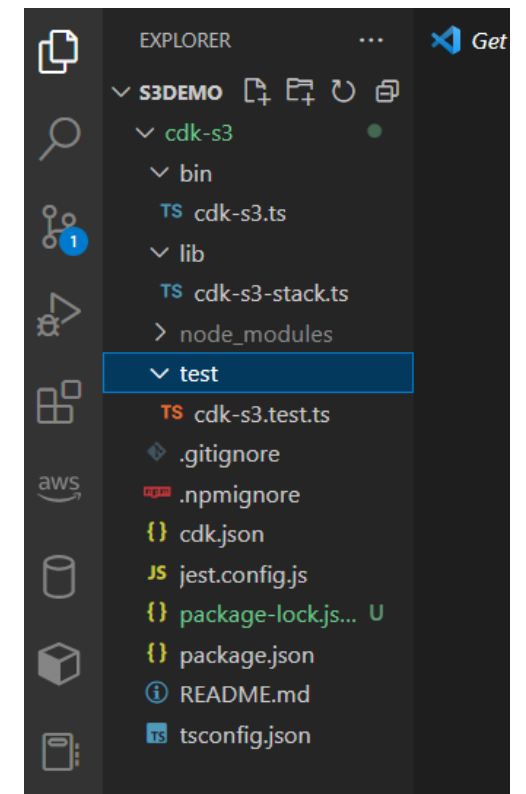
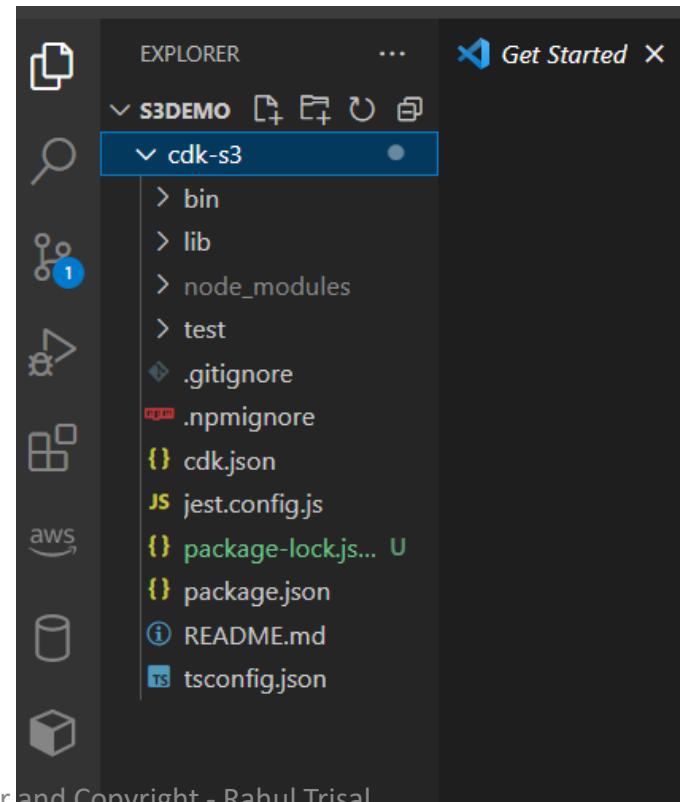
```
PS C:\Users\ADMIN\Desktop\AWS CDK\S3Demo> cd cdk-s3
PS C:\Users\ADMIN\Desktop\AWS CDK\S3Demo\cdk-s3>
```

AWS CDK – Project Structure

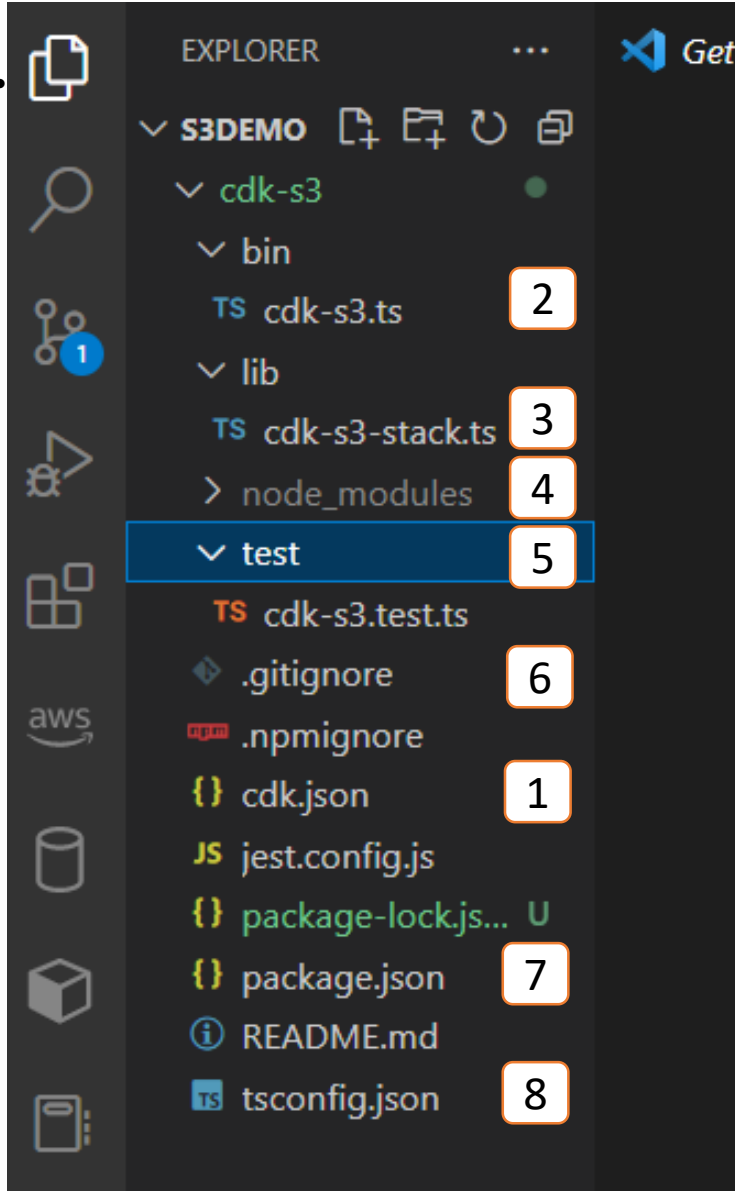
3. Initialize the app by using the **cdk init** command, specify the desired template ("**app**") and **programming language**

- `cdk init app --language typescript`

Below folder structure will be generated :



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

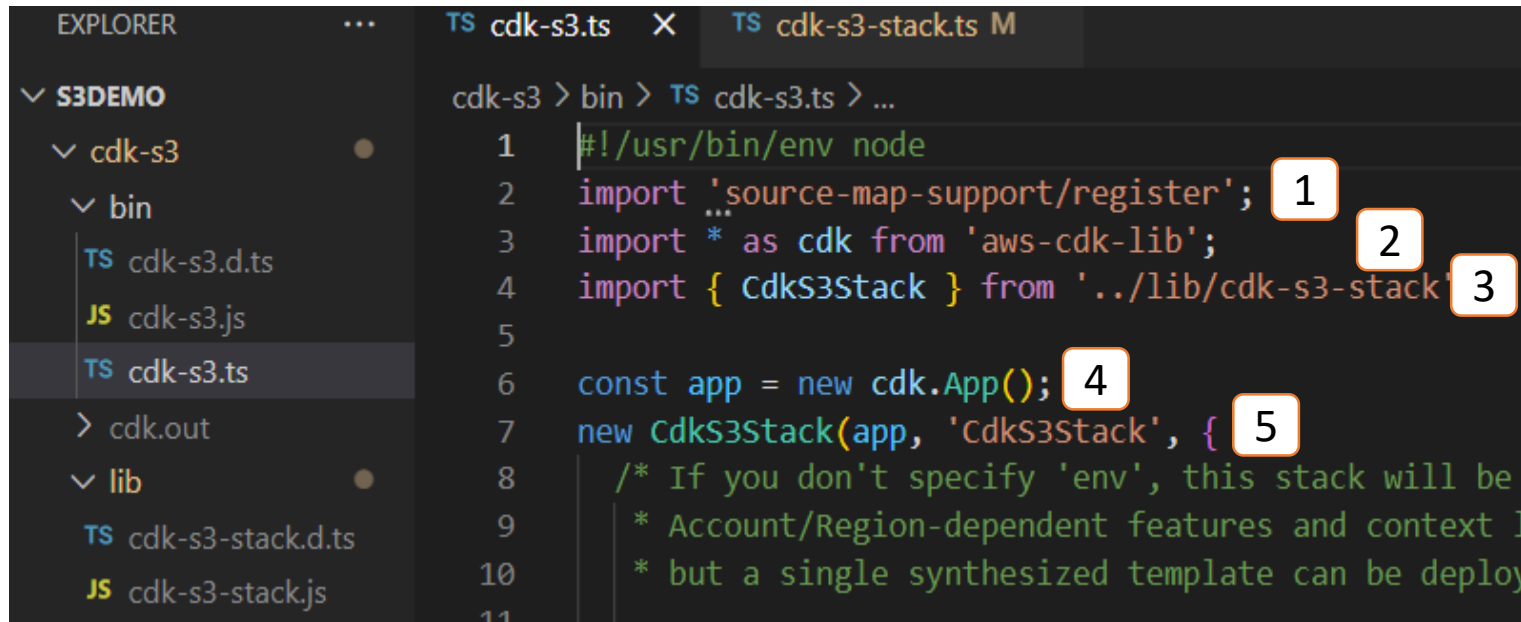
- Entrypoint of the CDK application.
- Will load the stack defined in lib/cdk-workshop-stack.ts.

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



The screenshot shows the VS Code interface with the Explorer on the left and the editor on the right. The Explorer shows a project named 'S3DEMO' with a folder 'cdk-s3' containing a 'bin' folder and a 'lib' folder. The 'bin' folder contains 'cdk-s3.d.ts', 'cdk-s3.js', and 'cdk-s3.ts'. The 'lib' folder contains 'cdk-s3-stack.d.ts' and 'cdk-s3-stack.js'. The editor shows the content of 'cdk-s3.ts' with the following code:

```
1  #!/usr/bin/env node
2  import 'source-map-support/register';
3  import * as cdk from 'aws-cdk-lib';
4  import { CdkS3Stack } from '../lib/cdk-s3-stack';
5
6  const app = new cdk.App();
7  new CdkS3Stack(app, 'CdkS3Stack', {
8    /* If you don't specify 'env', this stack will be
9     * Account/Region-dependent features and context l
10    * but a single synthesized template can be deploy
```

Numbered annotations are placed on the code:

- 1: `import 'source-map-support/register';`
- 2: `import * as cdk from 'aws-cdk-lib';`
- 3: `import { CdkS3Stack } from '../lib/cdk-s3-stack';`
- 4: `const app = new cdk.App();`
- 5: `new CdkS3Stack(app, 'CdkS3Stack', {`

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  import * as cdk from 'aws-cdk-lib';           1
2  import { Construct } from 'constructs';       2
3  import * as s3 from 'aws-cdk-lib/aws-s3';     3
4
5  // import * as sqs from 'aws-cdk-lib/aws-sqs';
6
7  export class CdkS3Stack extends cdk.Stack {    4
8      constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
9          super(scope, id, props);
10
11         // The code that defines your stack goes here
12
13         // example resource
14         // const queue = new sqs.Queue(this, 'CdkS3Queue', {
15         //     visibilityTimeout: cdk.Duration.seconds(300)
16         // });
17
18         // S3 resource
19         const bankings3 = new s3.Bucket(this, 'sampleLogicals3', {
20             bucketName: 's3demobucket07012023',  5
21             versioned : true
22         });
```

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'*

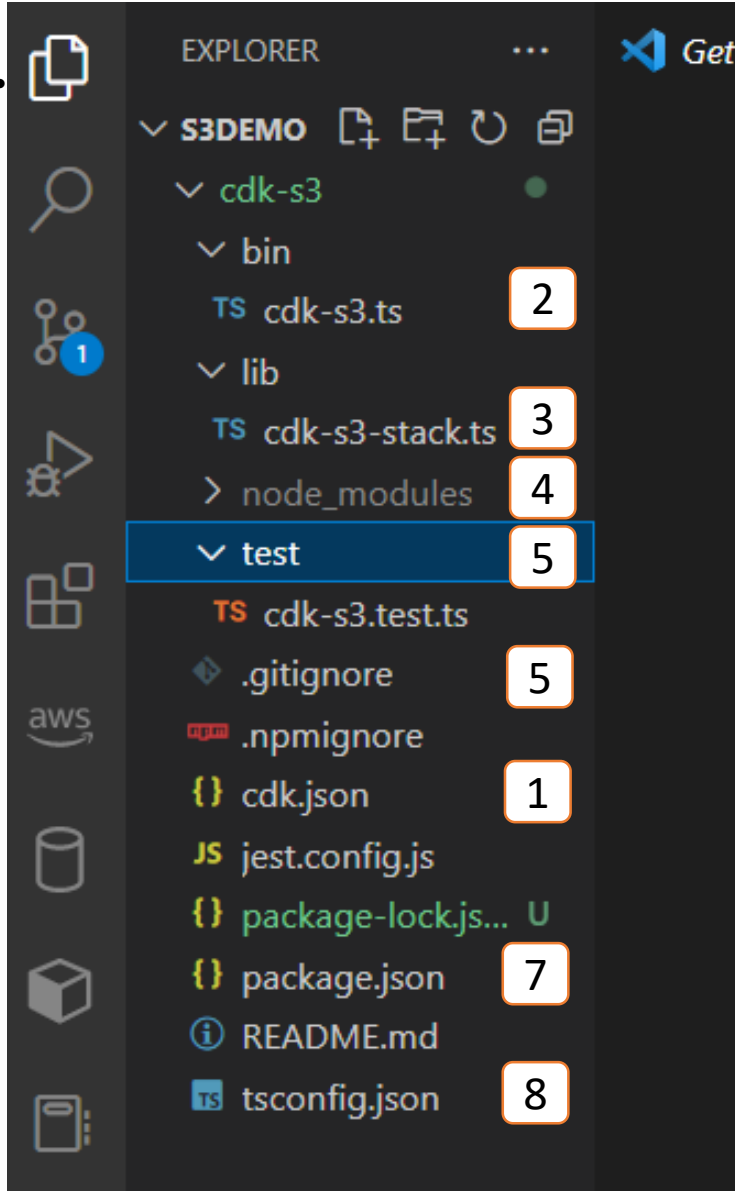
4. CDK Stack

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



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

Step 1.

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

Step 2.

Create the app

- Make a directory **infra** (or any other based on your choice)
 - *`mkdir infra`*
- Change directory with **cd**
 - *`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

Creator and Copyright - Rahu Tisa

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

Step 5

Build the app (Optional)

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*

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:

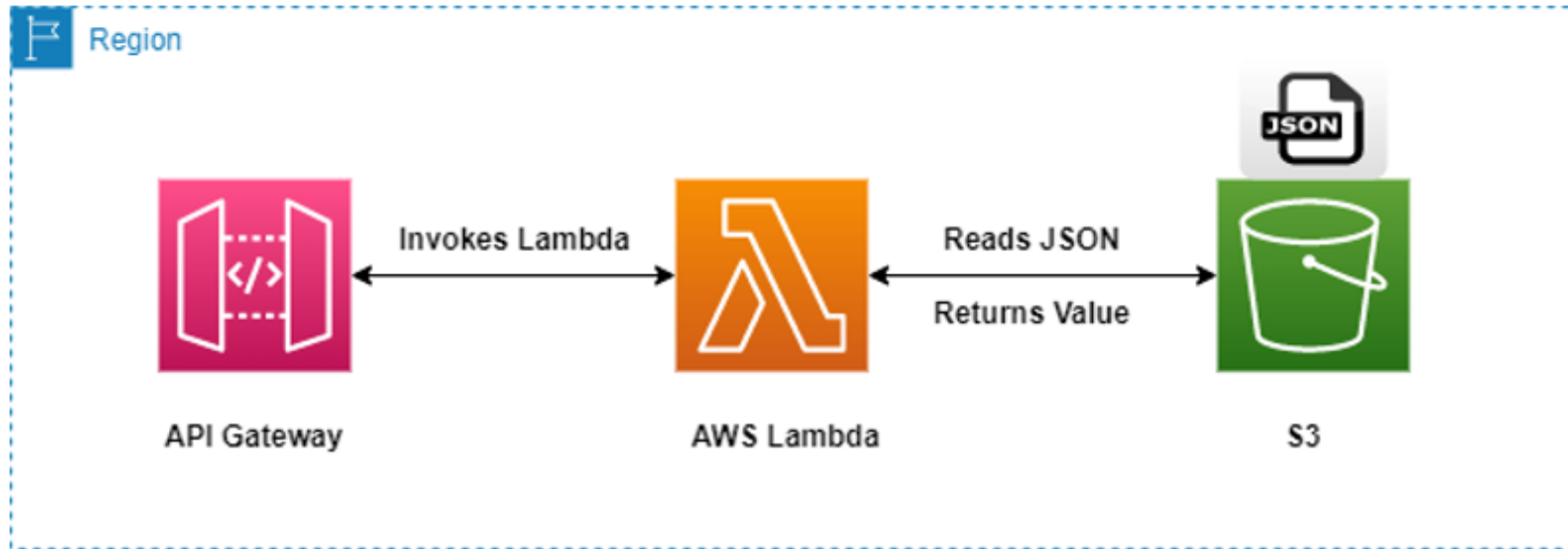
- Props :
 - evaluationPeriods
 - threshold
 - alarmName?
 - metric – metricErrors()

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*

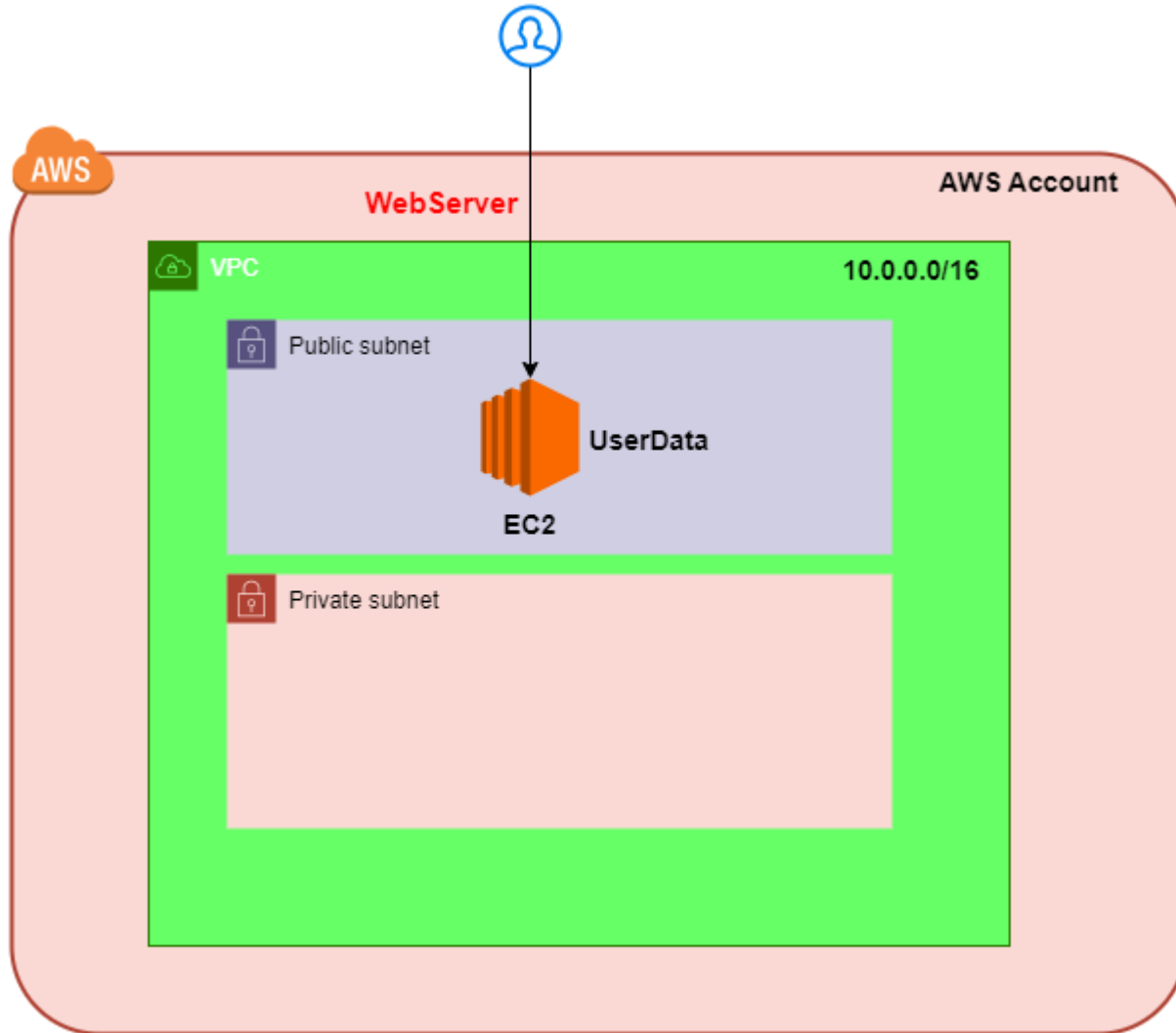
- *Resource - balanceStatus*
- *Method – GET*
- *Construct - LambdaRestAPI*

Section 6:

Use Case 2 - WebServer

AWS CDK v2 – VPC and EC2 as Web Server

Solution Architecture to Build – Web Server



AWS Services to Build

- VPC, Subnets and other VPC components
- AWS Security Group
- EC2 Instance with User Data

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 – VPC , Subnets, Route Tables and IG

AWS VPC and Associated Services:

- Props :
 - ipAddresses? --- > ~ 65,000 --- > 10.0.0.0/16 ($2^{32}-16$)
 - vpcName?
 - natGateways:0 (IMPORTANT!! – To not incur charges)

(<https://www.ipaddressguide.com/cidr>)

AWS CDK – EC2 with User Data

SecurityGroup:

- Props :
 - vpc
 - allowAllOutbound?
 - description?
 - securityGroupName?
 - addIngressRule(peer, connection, description?, remoteRule?)

AWS CDK – EC2 with User Data

EC2:

- Props :
 - VPC
 - Subnet
 - Security Group
 - `instanceType` (Instance Class and Size)
 - `machineImage` (latestAmazonLinux)
 - Key Pair
 - `userData` [`addUserData(...commands)`]

<https://nodejs.dev/en/learn/reading-files-with-nodejs/>

Section 7:

AWS CodeWhisperer and CK

AWS CodeWhisperer (Generative AI Tool powered by LLM)

- [Amazon CodeWhisperer](#) is an **AI coding** companion that **helps improve developer productivity** by **generating code recommendations based on their comments in natural language** and code in the (IDE).
- As you write code, **CodeWhisperer automatically generates suggestions** based on your existing code and comments.
- CodeWhisperer is powered by a **Large Language Model (LLM)** that is **trained on billions of lines of code**, and as a result, has learned how to write code in 15 programming languages.

“Accenture is using Amazon CodeWhisperer to accelerate coding as part of our software engineering best practices initiative in our Velocity platform,” says Balakrishnan Viswanathan, Senior Manager, Tech Architecture at Accenture. “The Velocity team was looking for ways to improve developer productivity. After searching for multiple options, we came across Amazon CodeWhisperer to reduce our development efforts by up to 30% and we are now focusing more on improving security, quality, and performance.”



Source : <https://aws.amazon.com/blogs/machine-learning/how-accenture-is-using-amazon-codewhisperer-to-improve-developer-productivity/>

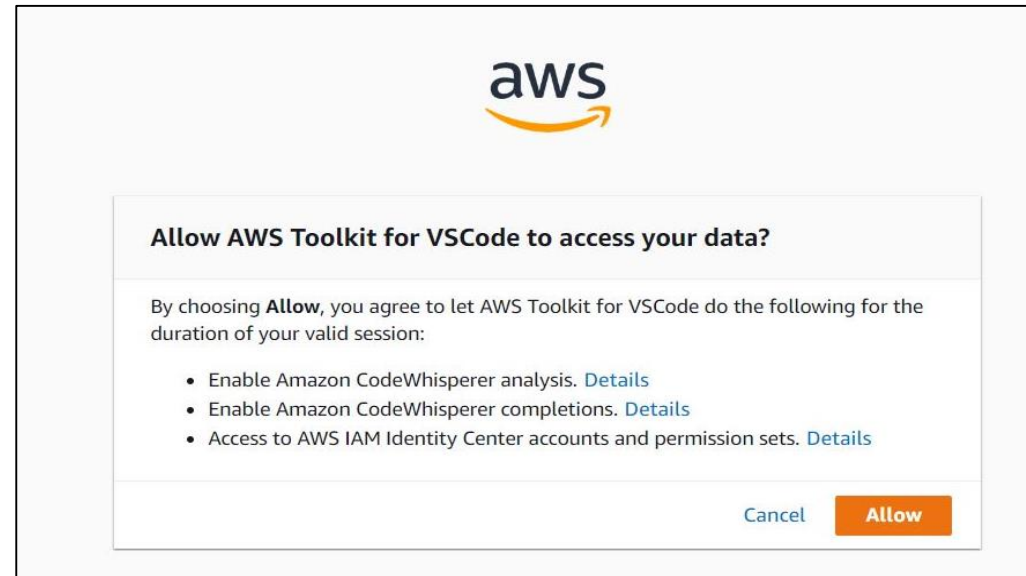
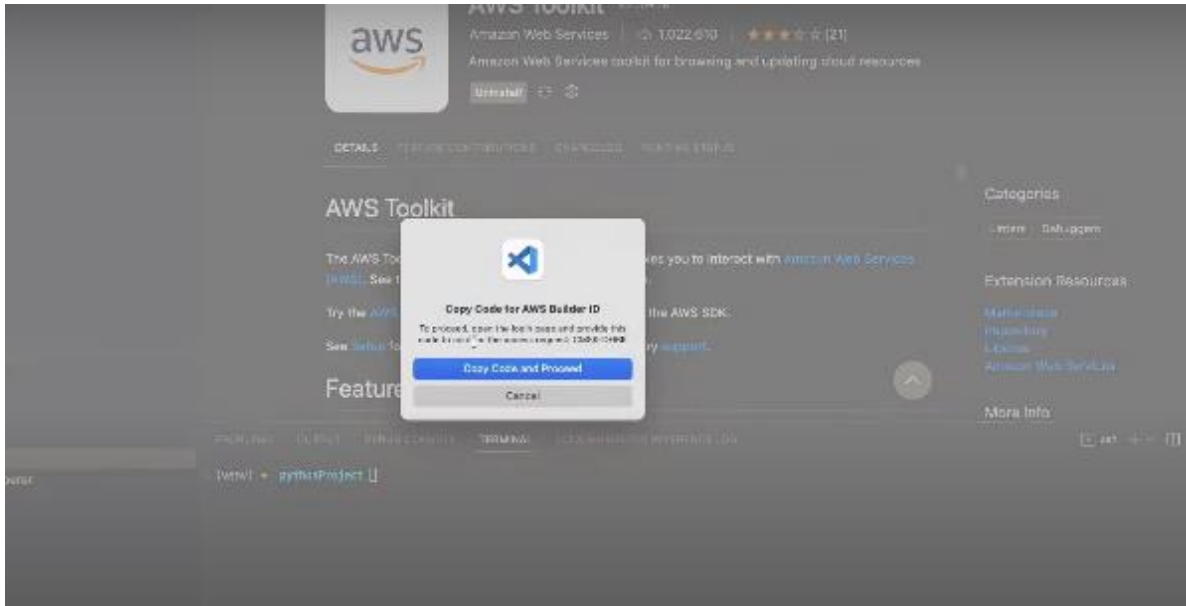
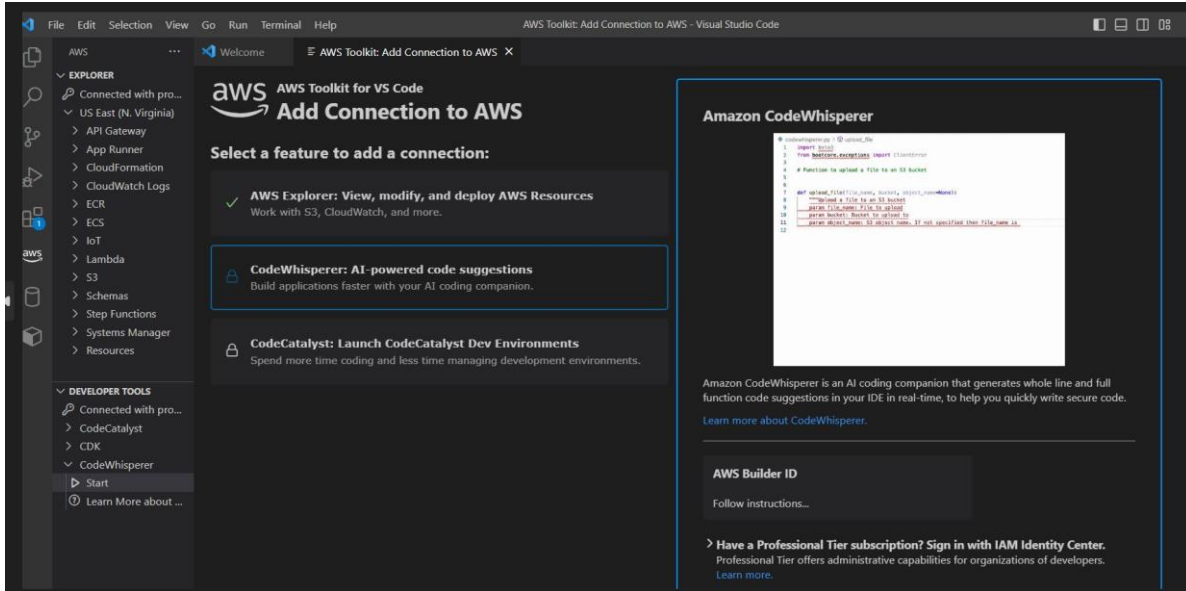
AWS CDK and CodeWhisperer (Generative AI powered by LLM)

Install Code AWS Whisperer on VSCode from below link:

- Use **Individual Tier** which is free to use and requires a simple sign-up to create an **AWS Builder ID**.
- <https://docs.aws.amazon.com/codewhisperer/latest/userguide/whisper-setup-indv-devs.html>
- <https://www.youtube.com/watch?v=rHNMfOK8pWI>

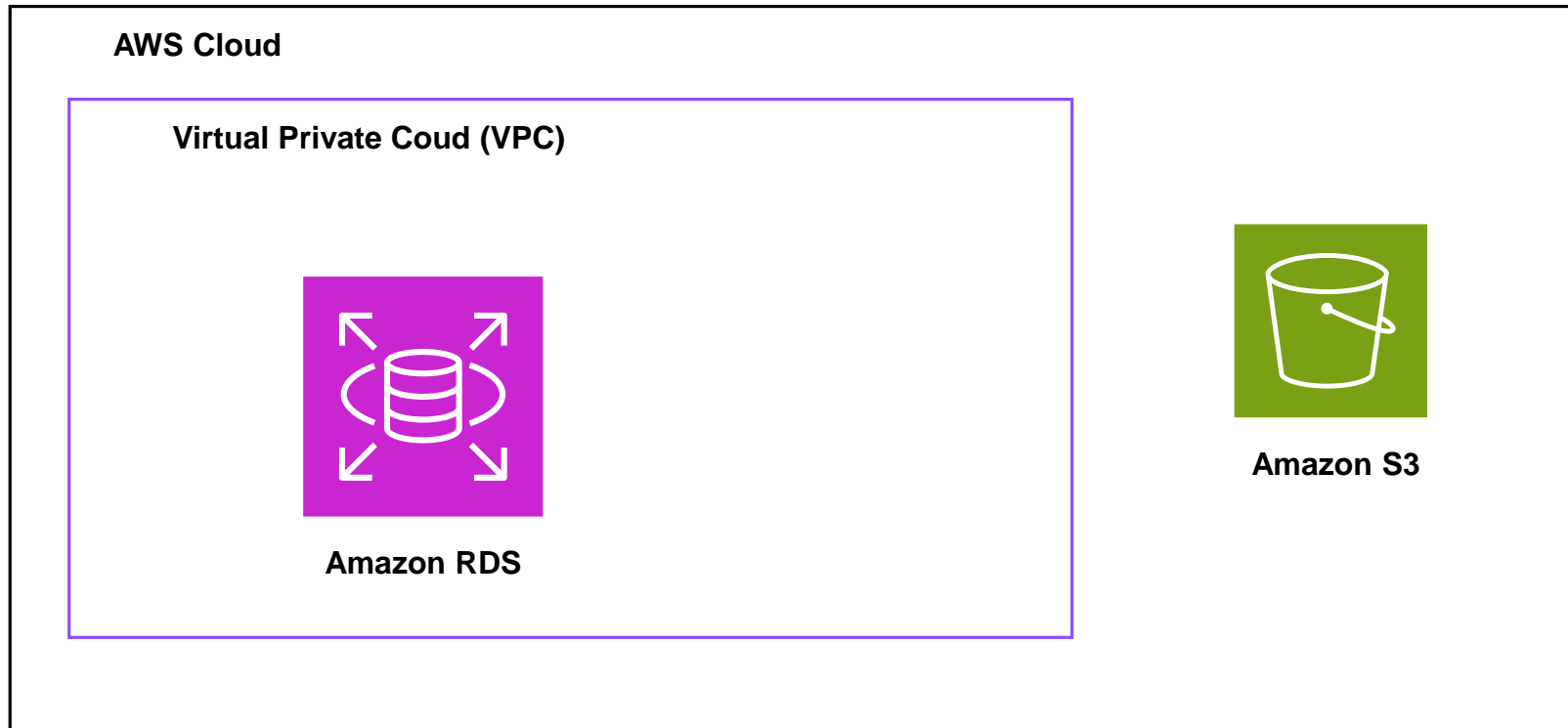


AWS CDK and CodeWhisperer (Generative AI powered by LLM)



AWS CDK and CodeWhisperer (Generative AI powered by LLM)

What will we build using AWS CodeWhisperer and CDK ?



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 the Directory - *`mkdir codewhisperer and cd codewhisperer`*
- 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 – CodeWhisperer

Keyboard Shortcut

Action

[TAB]

To accept suggestions from Code Whisperer

[left arrow] and [right arrow]

Navigate between suggestions.

Keep typing (or hit ESC).

To ignore the suggestions from Code Whisperer

MacOS: Option + C

Windows: Alt + C

Prompt Code Whisperer for Suggestions

AWS CDK v2 – CodeWhisperer – AWS Recommendations

- Write **descriptive comments**. “Function to upload a file to S3” will get better results than “Upload a file”.
- **Specify the libraries** you prefer by using import statements.
- Use **descriptive names for variable and functions**. A function called “upload_file_to_S3” will get better results than a function called “file_upload”.
- Give CodeWhisperer something to work with. The **more code your file contains**, the more context CodeWhisperer has for generating recommendations.

Source : AWS Documentation

Section 8:

AWS CDK - Intermediate Concepts 1 :

L1, L2 and L3 Constructs

L1, L2 and L3 Constructs in AWS CDK

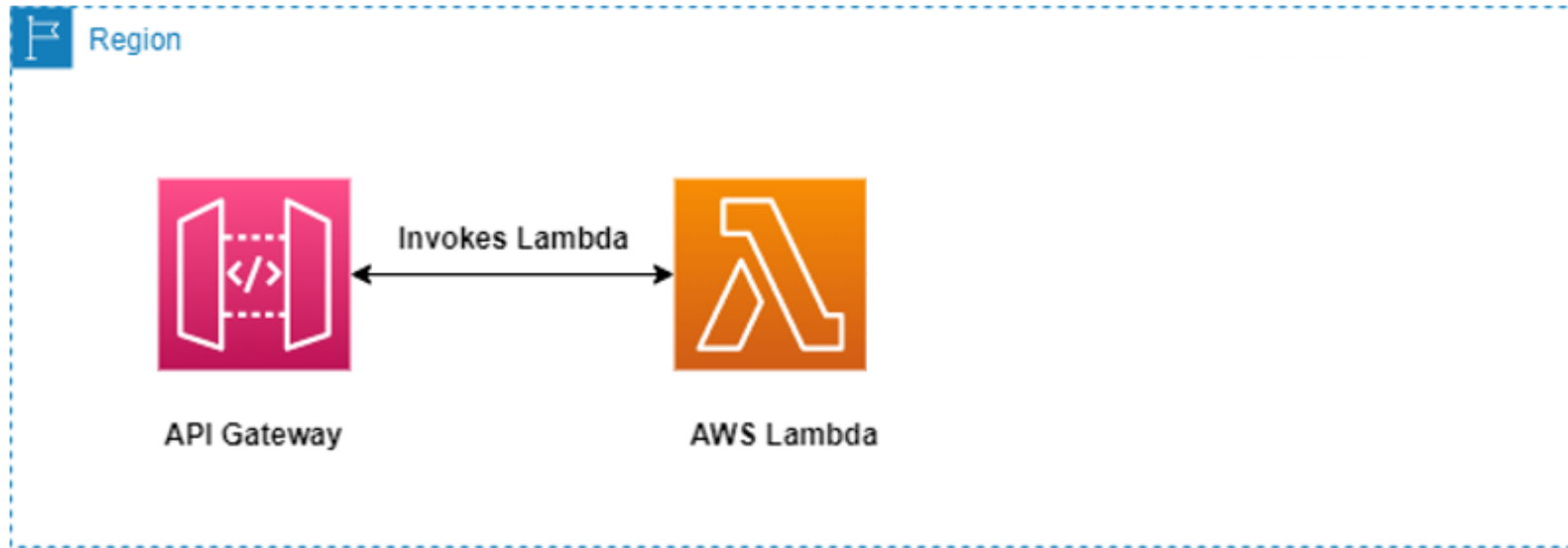
L1 Constructs	L2 Constructs	L3 Constructs
<ul style="list-style-type: none">• Low level Constructs or CFN Resources• 'Cfn' Prefix + naming convention - CfnBucket	<ul style="list-style-type: none">• Higher Level Constructs	<ul style="list-style-type: none">• Even higher level construct or a Pattern
<ul style="list-style-type: none">• 1:1 mapping to CloudFormation Resources	<ul style="list-style-type: none">• Intelligent Defaults	<ul style="list-style-type: none">• Focused on a specific use case
<ul style="list-style-type: none">• CfnBucket represents AWS::S3::Bucket• 282 lines of code for VPC	<ul style="list-style-type: none">• s3.Bucket• 3 lines of code for VPC	<ul style="list-style-type: none">• aws-apigateway.LambdaRestApi• aws-ecs <p>patterns.ApplicationLoadBalancedFargateService</p>
<ul style="list-style-type: none">• BucketName• Versioning	<ul style="list-style-type: none">• BucketName• Versioning	<ul style="list-style-type: none">• aws-apigateway.LambdaRestApi<ul style="list-style-type: none">• API Gateway• Lambda• REST API• IAM Role and other services

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 Cloud Development Kit (CDK) v2 – Serverless Use Case

L3 Construct - *aws-apigateway.LambdaRestApi*



- *Rest API*
- *Resource*
- *Method*
- *Stage*
- *IAM Roles*
- *Invocation Role*
- *Lambda Integration*

```
class LambdaRestApi (construct)
```

Section 9:

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)

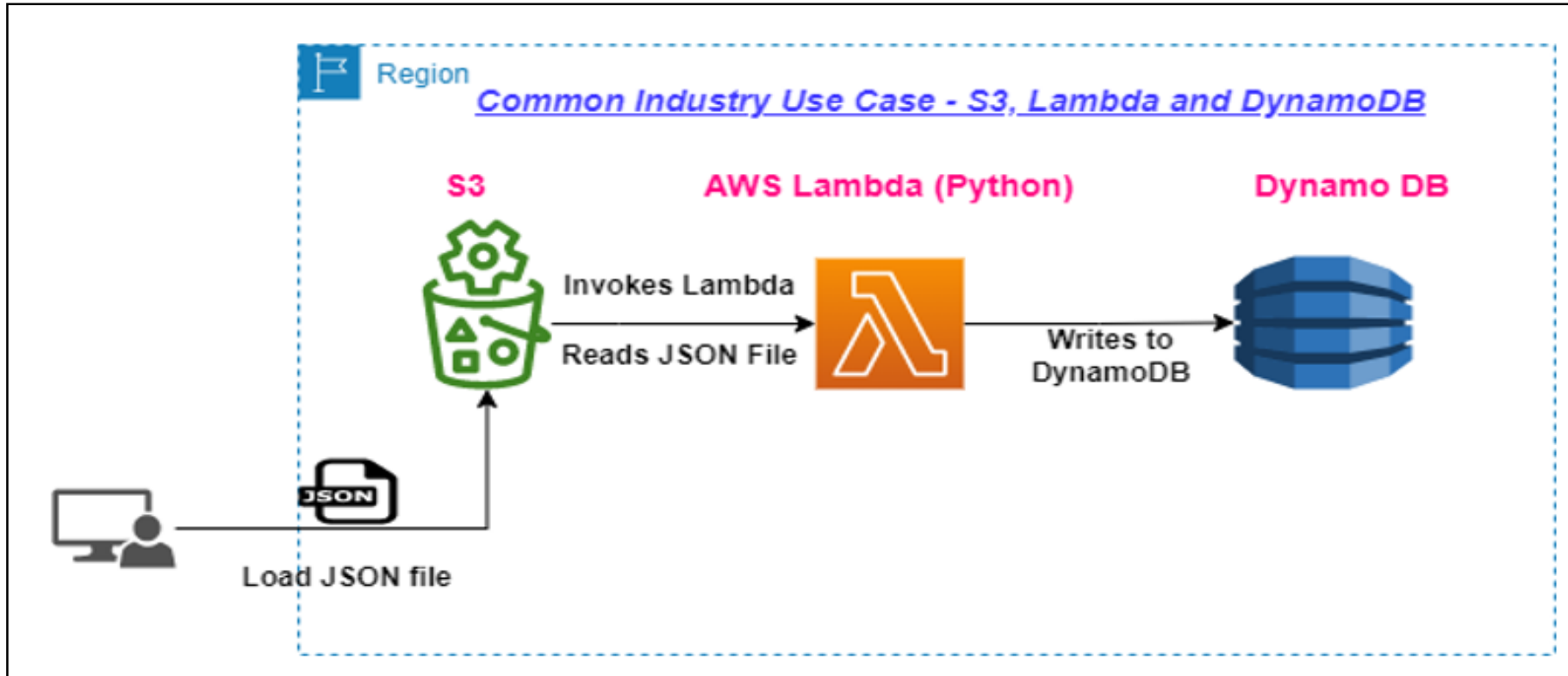
Refer to pdf uploaded in the relevant section for commands

Section 10:

Use Case 3 - 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

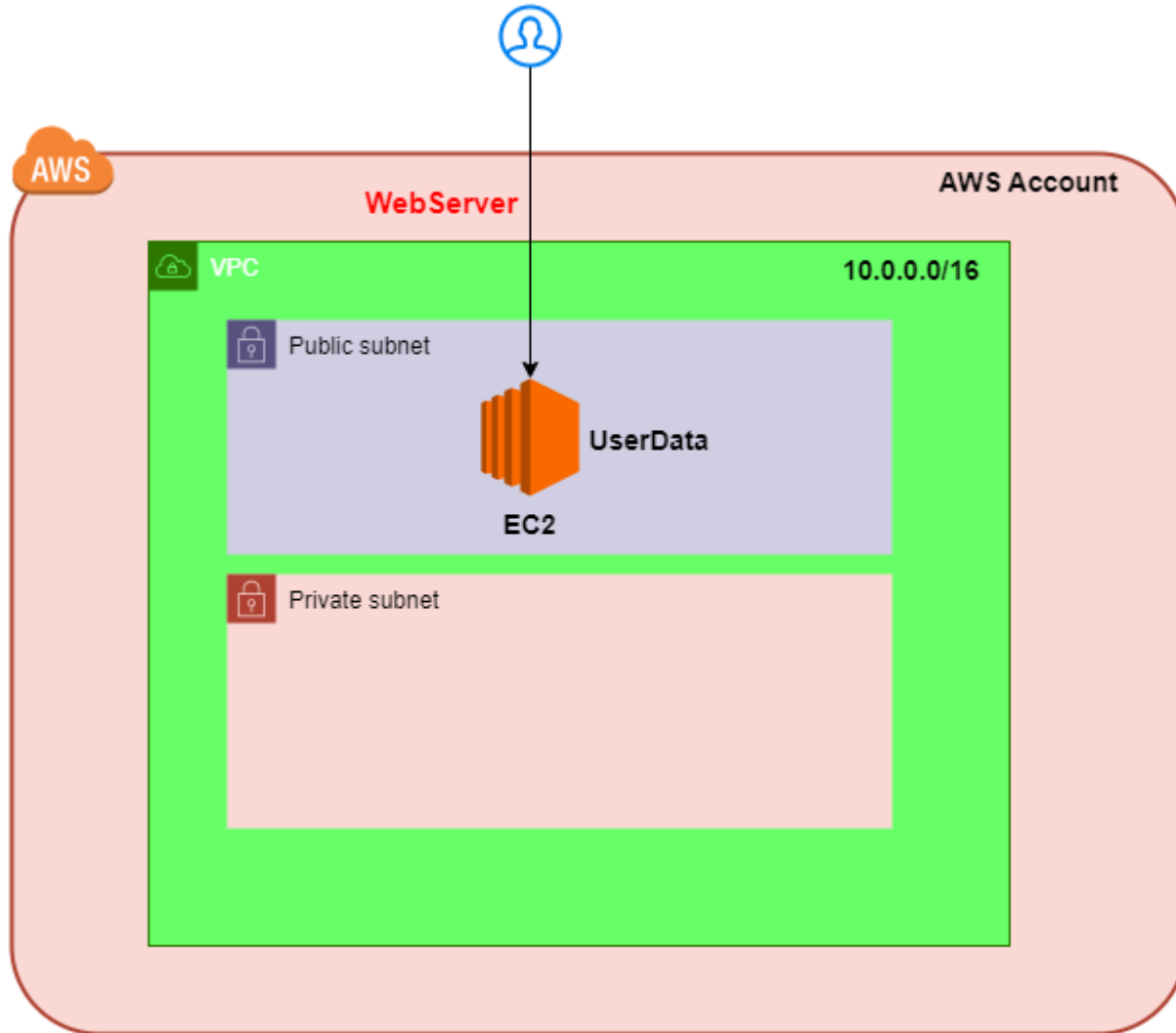
- *partitionKey – customername (String)*
- *tableName - inventoryfeedynamodb01*

Section 11:

CDK Advanced Concepts - Cross App & Multi-Stack Resource Sharing

CDK v2 – Multi-Stack/Resource Sharing across Apps/Stacks

Solution Architecture to Build – Web Server



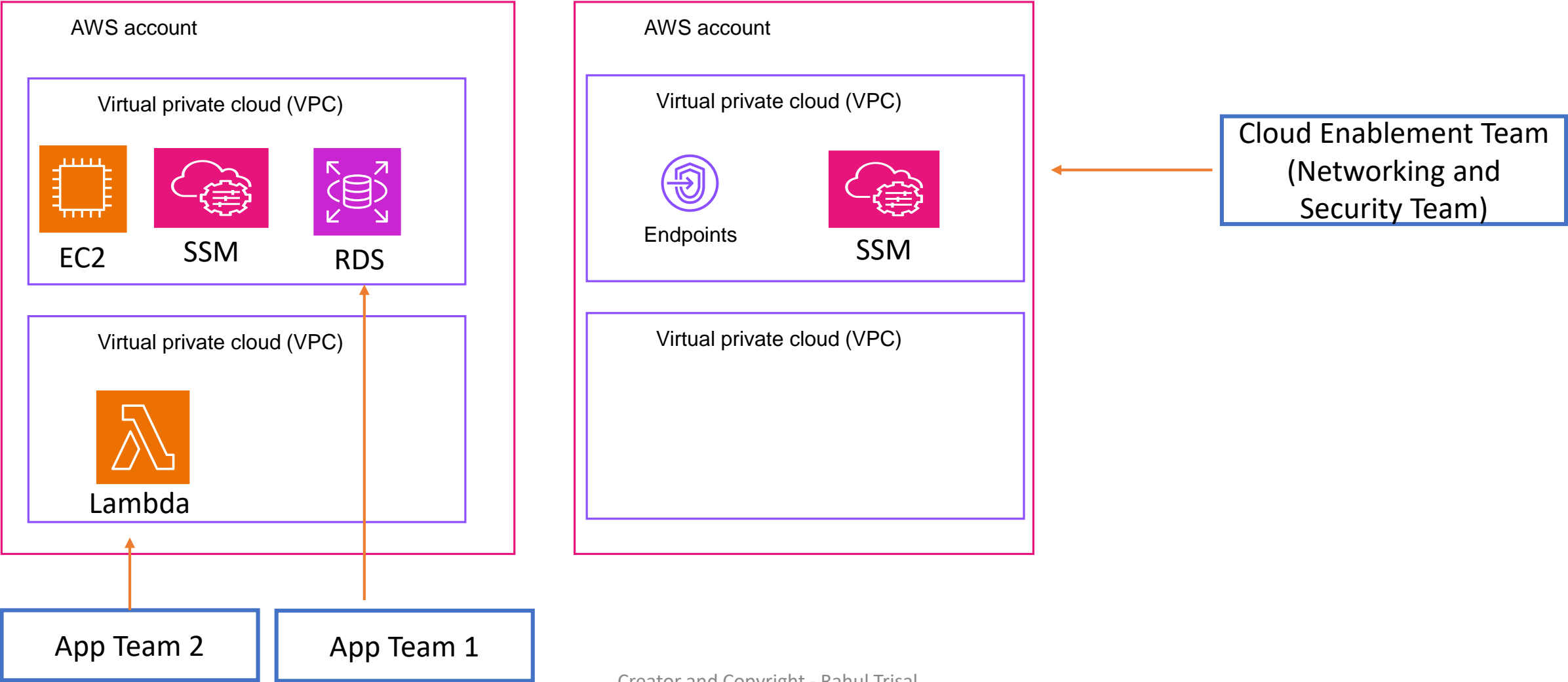
AWS Services to Build

- VPC, Subnets and other VPC components
- AWS Security Group
- EC2 Instance with User Data

AWS CDK v2 – Need for Sharing of Resource Across Apps

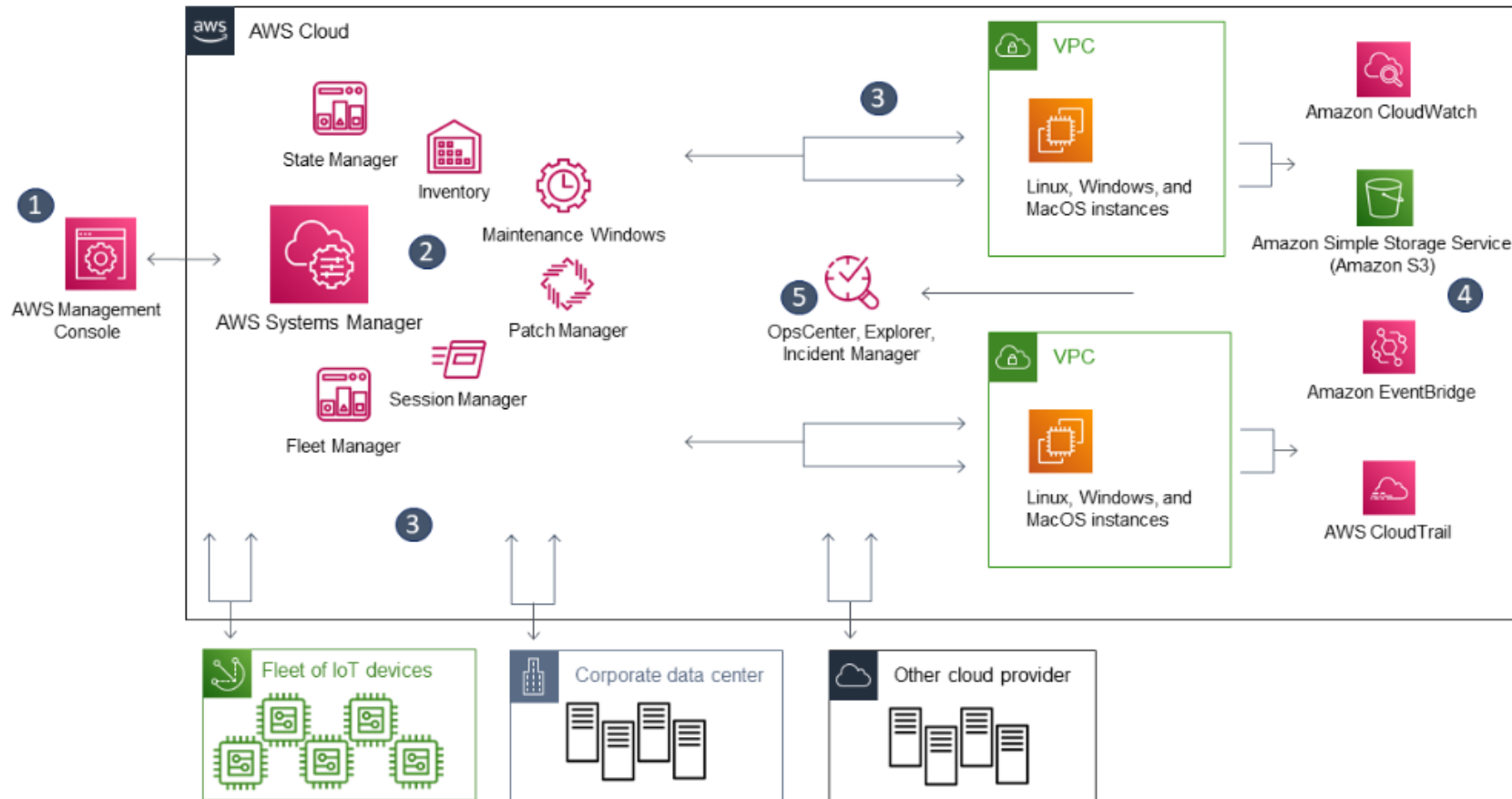
AWS Cloud

Large Organization AWS Account Structure - Simplified



AWS CDK v2 – AWS System Manager

AWS Systems Manager is the **operations hub for your AWS applications and resources** and a **secure end-to-end management solution** for hybrid and multi-cloud environments that enables secure operations at scale.



AWS CDK v2 – AWS SSM – Parameter Store

Parameter Store, a capability of AWS Systems Manager, provides secure, hierarchical storage for configuration data management and secrets management.

- You can store data such as **passwords, database strings, VPC details, Amazon Machine Image (AMI) IDs**, and license codes as parameter values.

Parameter Store provides support for three types of parameters: String, StringList, and SecureString.

String –

- String parameters consist of any block of text you enter. For example: abc123, Example Corp

StringList

- StringList parameters contain a comma-separated list of values - Monday, Wednesday, Friday

SecureString

- A SecureString parameter is any sensitive data that needs to be stored and referenced in a secure manner - such as passwords or license keys, create those parameters using the SecureString data type.

AWS CDK v2 – Sharing of Resource Information

Sharing of AWS Resources Information between Stack and Apps

- Different Stacks in the same App
- Different CDK Apps

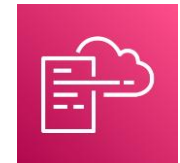
Three Possible Approaches and Pros/Cons

Approach 1 : **Using AWS SSM Parameters** to share information across Apps



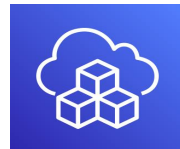
AWS System Manager – Parameter Store

Approach 2 : **Using class CfnOutput (construct)** to share information across Apps – **Not a very elegant/native solution**



AWS CloudFormation

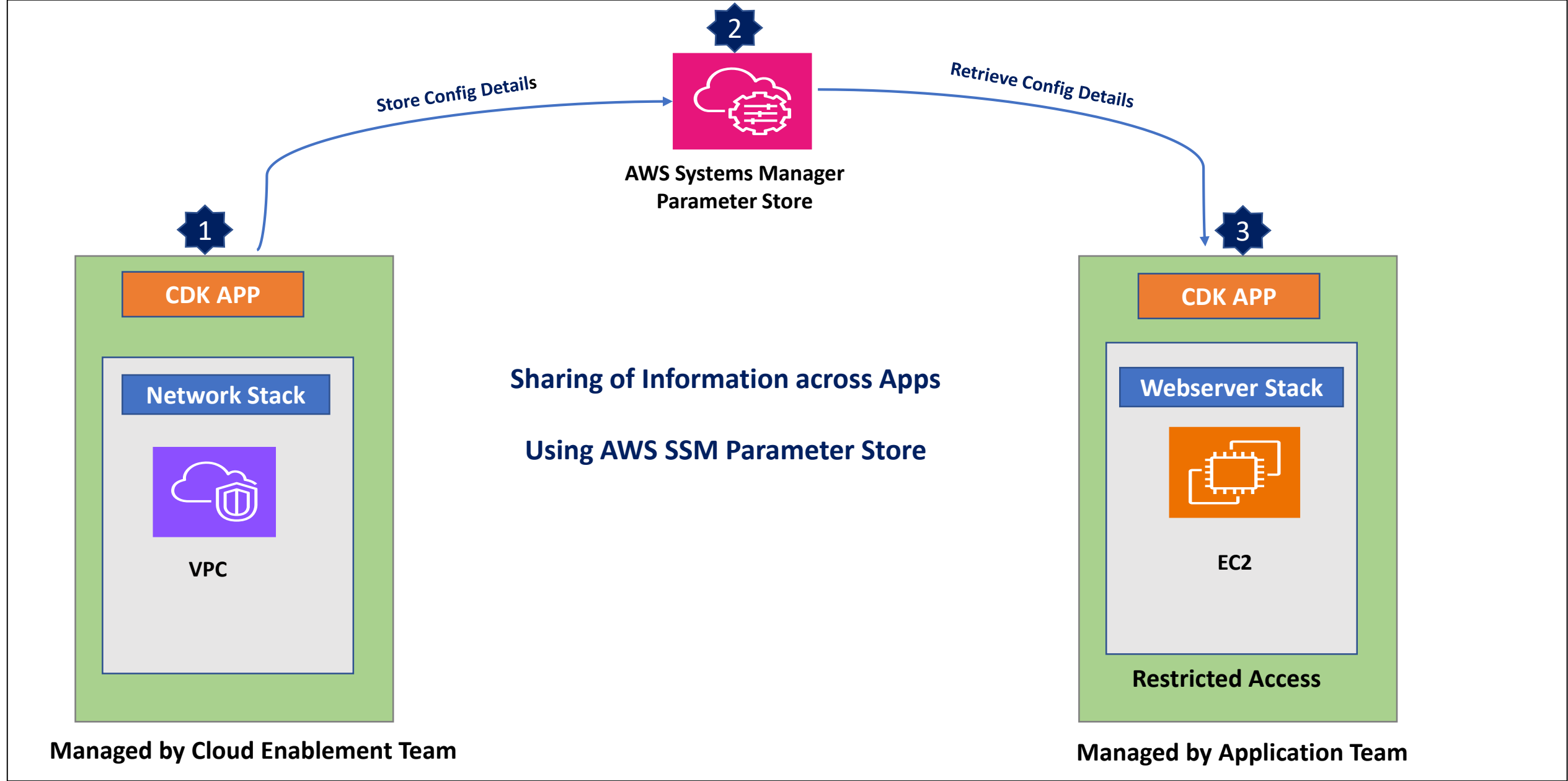
Approach 3 : **Create Multi-Stack App – Same App**



AWS CDK

Creator and Copyright - Rahul Trisal

AWS CDK – Sharing of Resource Information across Apps



AWS CDK v2 – Approach 1 using SSM Parameter Store

To implement solution in CDK with Approach 1, need to:

1. **Create the Resource (in the CDK)** that needs to be used by other Stack or App **such as VPC in this example.**
2. Provision the **SSM Parameter Store** using CDK and **store the values**
3. **Read values from the Systems Manager** Parameter Store, use either :
 - ``StringParameter.fromStringParameterName`` (Reads SSM values at **Deployment time** – supported by limited AWS Services)
 - ``StringParameter.valueFromLookup`` methods (Reads SSM values at **Synthesis time** – supported by limited AWS Services)
 - If the value is not already cached in `cdk.json` or passed on the command line, it is retrieved from the current AWS account.
 - Provide explicit **Account and Region information.**
4. Retrieve SSM Value
 - `const versionOfStringToken = ssm.StringParameter.valueForStringParameter(this, 'my-plain-parameter-name', 1); // version 1`

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

AWS CDK v2 – Issues and Troubleshooting

If you are following along and deploying the code, keep following things in mind :

- **Sequence of deployment**

1. Deploy Network Stack **first**
2. Deploy SSM Stack **Second**
3. Deploy WebServer Stack **Third**

Stacks (5)		
<input type="text" value="Filter by stack name"/>		
	Stack name	Status
<input checked="" type="radio"/>	WebserverStack	3 ✔ CREATE_COMPLETE
<input type="radio"/>	SsmStack	2 ✔ CREATE_COMPLETE
<input type="radio"/>	NetworkStack	1 ✔ CREATE_COMPLETE

- **Don't forget to modify the 'stringValue' in SSM Parameter app based on the VPCID value generated from Network Stack**

```
// The code that defines your stack goes here

// SSM Parameters resource
const demoSSM = new ssm.StringParameter(this, 'SSMParameters', {
  parameterName: '/vpc/demoVPCID',
// Important Tip-Dont forget to substitute the value of VPCID from Network Stack
  stringValue: 'vpc-03b9ec2aa32e645fa'
});
}
```

AWS CDK v2 – Issues and Troubleshooting

If you get an error like below VPC ID does not exist

Failed resources:

WebserverStack | 9:49:24 PM | CREATE_FAILED | AWS::EC2::SecurityGroup | demoSG (demoSG8C892E70) The vpc ID 'vpc-01271a310958efd69' does not exist (Service: AmazonEC2; Status Code: 400; Error Code: InvalidVpcID.NotFound; Request ID: ce7e1383-01c3-4af9-a9d2-efb6d1d3f5d1; Proxy: null)

✗ WebserverStack failed: Error: The stack named WebserverStack failed creation, it may need to be manually deleted from the AWS console: ROLLBACK_COMPLETE: The vpc ID 'vpc-01271a310958efd69' does not exist (Service: AmazonEC2; Status Code: 400; Error Code: InvalidVpcID.NotFound; Request ID: ce7e1383-01c3-4af9-a9d2-efb6d1d3f5d1; Proxy: null)
at FullCloudFormationDeployment.monitorDeployment (C:\Users\ADMIN\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:327:10235)
at process.processTicksAndRejections (node:internal/process/task_queues:95:5)

AWS CDK v2 – Issues and Troubleshooting

- Go to `cdk.context.json` or `cdk.json` file (in rare cases)
- Check the value of **VPCID** – if it is correct (should be same as value in SSM Parameter store, if it is incorrect)
- Run following command >>> `cdk context --clear`
- It will clear all the cached values in the `cdk.context.json` file
- Then Redeploy the 'webserver' stack and it should work
- Incase of errors – message me on Udemy or LinkedIn

The screenshot shows a VS Code editor with the `cdk.context.json` file open. The file contains the following JSON:

```
{
  "ssm:account=196715057542:parameterName=/vpc/demoVPCID:region=us-east-1": "vpc-01271a310958efd69",
  "vpc-provider:account=196715057542:filter.vpc-id=vpc-01271a310958efd69:region=us-east-1": {
    "vpcId": "vpc-01271a310958efd69",
    "vpcCidrBlock": "10.0.0.0/16",
    "availabilityZones": [],
    "subnetGroups": [
      {
        "name": "Isolated",
        "type": "Isolated",
        "subnets": [

```

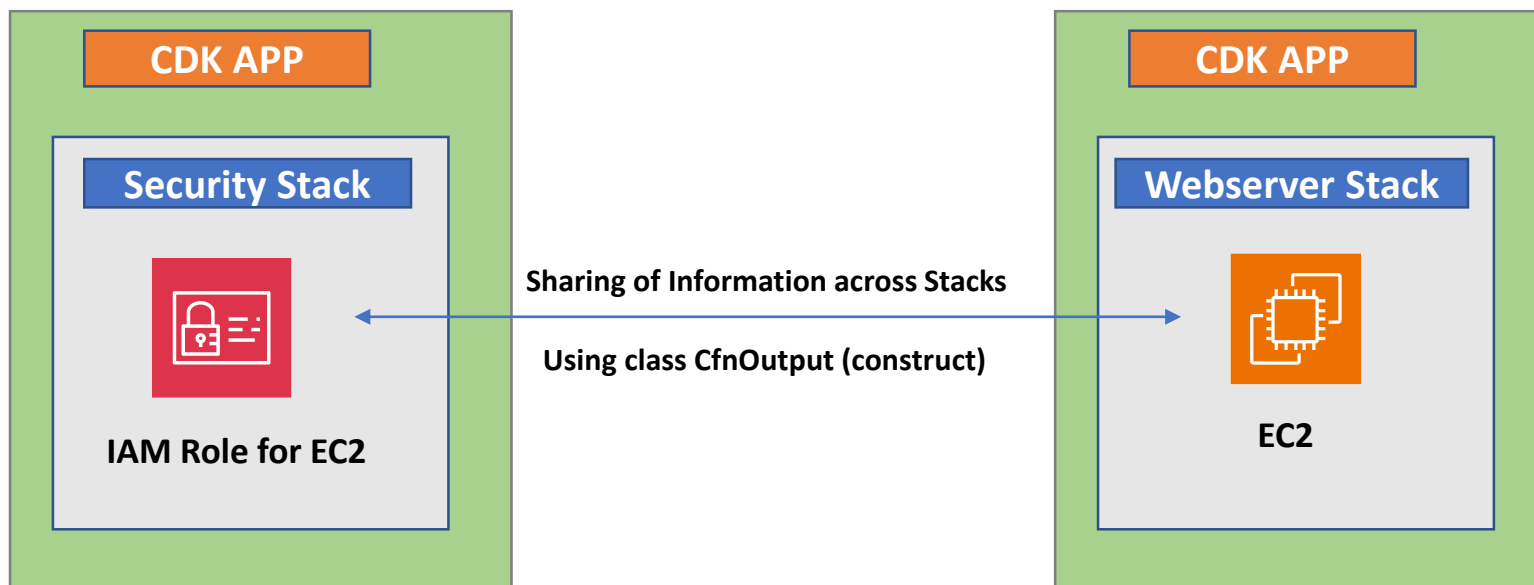
Below the editor, the terminal window shows the output of the `cdk deploy` command. The logs indicate that the `WebserverStack` was created successfully, but the `demoSG8C892E70` resource failed to create due to a rollback requested by the user.

```
WebserverStack | 1/6 | 9:49:24 PM | CREATE_COMPLETE | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
WebserverStack | 1/6 | 9:49:25 PM | ROLLBACK_IN_PROGRESS | AWS::CloudFormation::Stack | WebserverStack The following resource(s) failed to create: [demoSG8C892E70, d
emoEC2InstanceRoleE6912C8B]. Rollback requested by user.
WebserverStack | 1/6 | 9:49:27 PM | DELETE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
```

AWS CDK v2 – Approach 2 - CfnOutput (construct)

Approach 2 : Sharing of Resource Information across Apps using class CfnOutput (construct)

- CloudFormation feature, **to share information between stacks** - export a stack's output values and import the exported values in a different Stack
- This solution has few limitations:
- Uses **L1 constructs** - `CfnOutput` and `Fn.importValue` and require CloudFormation knowledge not a elegant native CDK solution
- `CfnOutput` values **only available at deployment to CDK**, rather than at synthesize time.



interface FromRoleArnOptions
https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_iam.FromRoleArnOptions.html

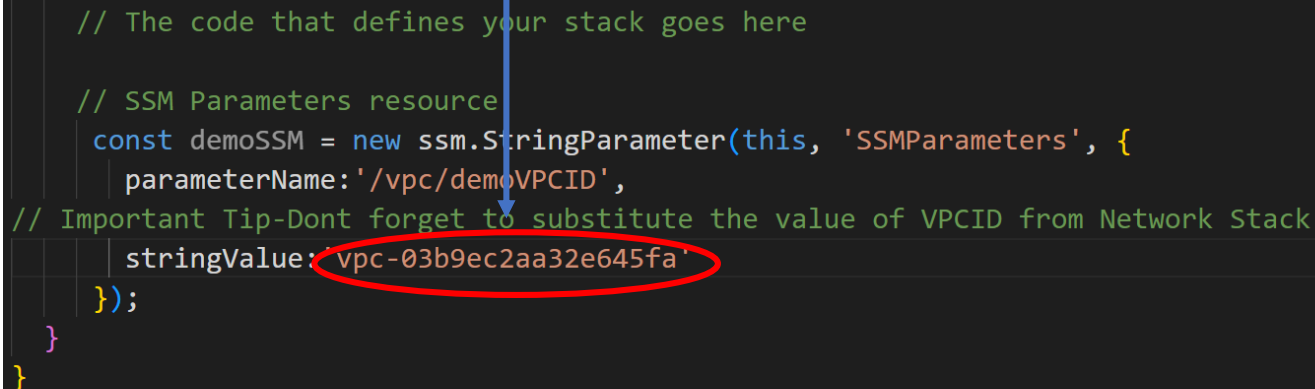
AWS CDK v2 – Issues and Troubleshooting

If you are following along and deploying the code, keep following things in mind :

- **Sequence of deployment**

1. Deploy Network Stack first
2. Deploy SSM Stack Second
3. Deploy Security Stack Third
4. Deploy WebServer Stack Fourth

- **Don't forget to modify the 'stringValue' in SSM Parameter app based on the VPCID value generated from Network Stack**



```
// The code that defines your stack goes here

// SSM Parameters resource
const demoSSM = new ssm.StringParameter(this, 'SSMParameters', {
  parameterName: '/vpc/demoVPCID',
// Important Tip-Dont forget to substitute the value of VPCID from Network Stack
  stringValue: 'vpc-03b9ec2aa32e645fa'
});
}
```

AWS CDK v2 – Issues and Troubleshooting

If you get an error like below VPC ID does not exist

Failed resources:

WebserverStack | 9:49:24 PM | CREATE_FAILED | AWS::EC2::SecurityGroup | demoSG (demoSG8C892E70) The vpc ID 'vpc-01271a310958efd69' does not exist (Service: AmazonEC2; Status Code: 400; Error Code: InvalidVpcID.NotFound; Request ID: ce7e1383-01c3-4af9-a9d2-efb6d1d3f5d1; Proxy: null)

✗ WebserverStack failed: Error: The stack named WebserverStack failed creation, it may need to be manually deleted from the AWS console: ROLLBACK_COMPLETE: The vpc ID 'vpc-01271a310958efd69' does not exist (Service: AmazonEC2; Status Code: 400; Error Code: InvalidVpcID.NotFound; Request ID: ce7e1383-01c3-4af9-a9d2-efb6d1d3f5d1; Proxy: null)
at FullCloudFormationDeployment.monitorDeployment (C:\Users\ADMIN\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:327:10235)
at process.processTicksAndRejections (node:internal/process/task_queues:95:5)

AWS CDK v2 – Issues and Troubleshooting

- Go to `cdk.context.json` or `cdk.json` file (in rare cases)
- Check the value of **VPCID** – if it is correct (should be same as value in SSM Parameter store, if it is incorrect)
- Run following command >>> `cdk context --clear`
- It will clear all the cached values in the `cdk.context.json` file
- Then Redeploy the 'webserver' stack and it should work
- Incase of errors – message me on Udemy or LinkedIn

The screenshot shows a VS Code editor with the `cdk.context.json` file open. The file contains the following JSON:

```
{  "ssm:account=196715057542:parameterName=/vpc/demoVPCID:region=us-east-1": "vpc-01271a310958efd69",  "vpc-provider:account=196715057542:filter.vpc-id=vpc-01271a310958efd69:region=us-east-1": {    "vpcId": "vpc-01271a310958efd69",    "vpcCidrBlock": "10.0.0.0/16",    "availabilityZones": [],    "subnetGroups": [      {        "name": "Isolated",        "type": "Isolated",        "subnets": [          {            "cidr": "10.0.0.0/24",            "availability": "AvailabilityZone",            "name": "IsolatedSubnet"          }        ]      ]    ]  }
```

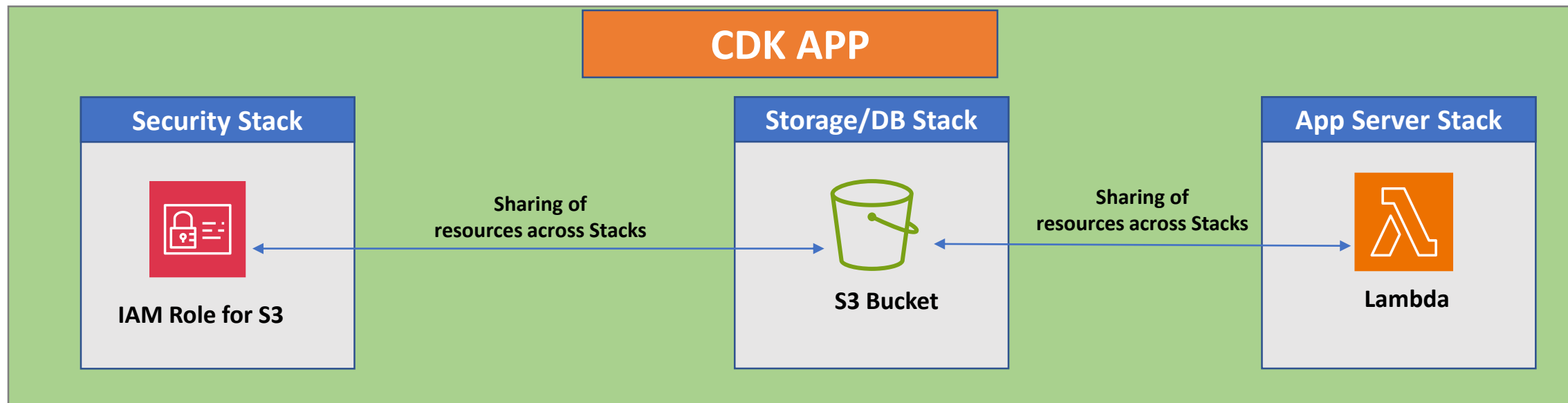
The terminal window shows the following logs:

```
WebserverStack | 1/6 | 9:49:24 PM | CREATE_COMPLETE | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
WebserverStack | 1/6 | 9:49:25 PM | ROLLBACK_IN_PROGRESS | AWS::CloudFormation::Stack | WebserverStack The following resource(s) failed to create: [demoSG8C892E70, d
emoEC2InstanceRoleE6912C8B]. Rollback requested by user.
WebserverStack | 1/6 | 9:49:27 PM | DELETE_IN_PROGRESS | AWS::CDK::Metadata | CDKMetadata/Default (CDKMetadata)
```

AWS CDK v2 – Approach 3 – Multi Stack (Same App)

Sharing of AWS Resources(Information) between Stack in Same App

- 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]



Section 12:




Additional CDK Concepts

AWS CDK v2 – Beginner to Advanced

Termination Protection in CDK

class Stack (construct)

This page is available in another version. [Click here for the v1 documentation.](#)

Language	Type name
 .NET	Amazon.CDK.Stack
 Go	github.com/aws/aws-cdk-go/awscdk/v2#Stack
 Java	software.amazon.awscdk.Stack
 Python	aws_cdk.Stack
 TypeScript (source)	aws-cdk-lib » Stack

infra -- > bin ---- > infra.ts file

terminationProtection?	boolean	Whether to enable termination protection for this stack.
------------------------	---------	--

<https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Stack.html>

AWS CDK – Main Commands

#	Command	Description
1	<i>cdk init app --language typescript</i>	Initialize the app
2	<i>npm run build</i>	Build the app (Optional)
3	<i>cdk bootstrap</i>	Bootstrap (One Time) - Deploys the CDK Toolkit staging stack
4	<i>cdk synth</i>	Synthesize an AWS CloudFormation template for the app
5	<i>cdk deploy</i>	Deploying the stack - Deploys one or more specified stacks
6	<i>cdk diff</i>	Compares the specified stack and its dependencies with the deployed stacks or a local CloudFormation template
7	<i>cdk docs (doc)</i>	Opens the CDK API Reference in your browser
8	<i>cdk list (ls)</i>	Lists the stacks in the app
9	<i>cdk metadata [Stackname]</i>	Displays metadata about the specified stack
10	<i>cdk deploy - - hotswap</i>	--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	<i>cdk destroy</i>	Destroys one or more specified stacks

AWS CDK – CfnOutput

Creates an CfnOutput value for this stack.

InfraStack

Delete

Update

Stack actions ▼

Create stack ▼

Stack info

Events

Resources

Outputs

Parameters

Template

Change sets

Outputs (1)

Search outputs

< 1 >

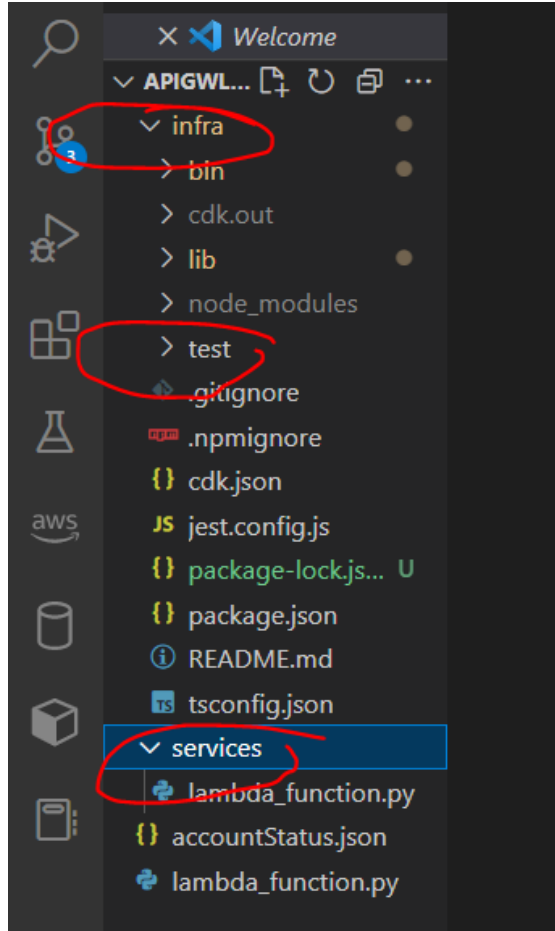
Key ▲	Value ▼	Description ▼	Export name ▼
S3BucketName	infrastack-demobucketadaa17f5-1gj9uat88mz1w	-	-

Section 13:

CDK Best Practices

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

1. Separate the Infrastructure and Application Code into separate folders



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 wrapper 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

Section 14:

CDK Testing

AWS CDK Testing – 1. Fine-grained assertions Tests

There are two categories of tests that you can write for AWS CDK apps.

Fine-grained assertions

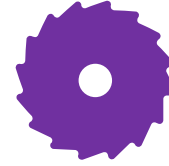


JEST

- AWS CDK apps uses the AWS CDK's [assertions](#) module and test frameworks like [Jest](#) for TypeScript
- Test **specific aspects** of the generated AWS CloudFormation template - this **property with this value**
- Detect **regressions**.
- **Test-driven development**- write a test first, then make it pass by writing a correct implementation
- Most **frequently** used tests.

AWS CDK Testing – Fine-grained assertions Tests

Steps to follow for testing using Fine-grained assertions



JEST

1. Write the CDK code for AWS Service being created with required properties or attributes
2. Write the CDK test for properties that need to be tested for AWS Infrastructure Service using Assertion module and Jest
3. Run the test using – npm run test command
4. AWS CDK validates against the synthesized CloudFormation template in the 'cdk.out' folder and not the deployed CloudFormation code in the Production environment.

AWS CDK Testing – 2. Snapshot Tests

Snapshot Tests



- Test the **synthesized AWS CloudFormation** template against a **previously stored baseline template**.
- Snapshot tests let you **refactor** freely
- If the changes are **intentional**, you can accept a new **baseline** for future tests.

AWS CDK Testing – Snapshot Tests



JEST

Steps to follow for testing using Snapshot Test

1. Write **the CDK code for AWS Service being created** with required properties or attributes
2. Write the **CDK test and define the baseline/master CloudFormation template** against which the new templates will be compared.
3. Run the test using – “**npm run test**” command
4. A **baseline CloudFormation template** will be generated under the “**__snapshots__**” folder.
5. When any changes are made to the CDK Code, the synthesized template will be tested against the baseline template
6. You can modify the template if changes are inadvertent or you can accept using “**npm test -- -u**” command, which will

Thank You