AWS CDK For Serverless Apps

AWS Phnom Penh User Group Meeting

Mark Ilott

Head of Application Architecture and Integration

16 October 2022

Application Development at J Trust Royal Bank

- Loosely coupled microservices, with communication between services via API's
- Serverless by default – using API Gateway, Lambda, StepFunctions, SQS, EventBridge…
- Containers where required – deployed on Fargate
- Servers for legacy applications only – deployed on VMWare or EC2

Deployment Tools

CDK	Terraform
AWS specific – creates and deploys CloudFormation templates	Multi-cloud and multi-vendor
Creates and deploys CloudFormation stacks, does not maintain state independently	Maintains state in state files, and deploys using vendor API's
High-level languages like Typescript/Javascript, Python, Java…	Hashicorp proprietary, declarative language
Uses language features like loops, conditions, functions and classes to create complex dependency graphs and logic	Limited looping and conditions
Easy for developers to learn, but harder if you have no dev experience	Easier to learn for non-devs, but requires deeper understanding of the infrastructure

Deployment Tools

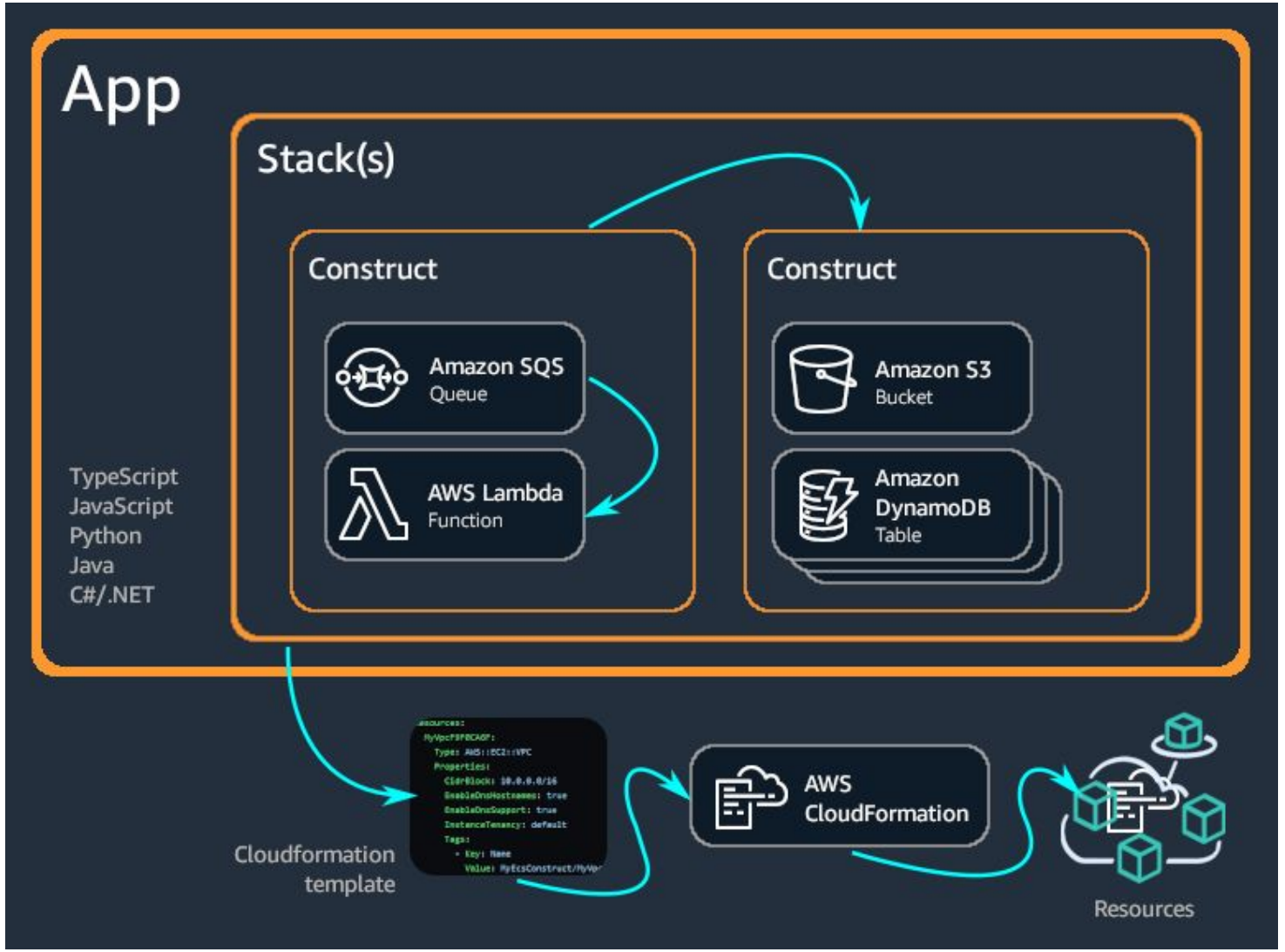
CDK	Terraform
Zero-downtime updates and rollbacks. Import existing infra and drift detection and remediation is limited.	Deployment uses API calls, with rollback handled with a second update. Import, drift detection and remediation are better.
Cross-region and cross-account deployments can be tricky (CloudFormation limitations)	No problem handling cross-region and cross-account deployments
Infra and application code managed in the same language and the same project	Infra and applications are separate
Custom CloudFormation resources easily created and deployed	Custom resources require more work and must be written in Go, but many common API calls are already available as Terraform modules

Languages

TypeScript	Java
Used for all serverless applications and web development (Vue)	Used where required for complex transformations, interaction with file-systems, interaction with legacy systems
Deployed in Lambda Functions and static web sites	Deployed in containers on Fargate
Applications, AWS resources, pipelines in a single monorepo	Apps in dedicated repos, AWS resources and pipelines in the CDK monorepo
CI/CD via CDK Pipelines	

CDK Constructs

CDK Deploys CloudFormation Stacks



L1 Constructs

- Level 1 Constructs are abstractions for CloudFormation resources
- They allow CDK to create anything that CloudFormation can create
- L1 constructs directly translate to CloudFormation resources, without any additional configuration or resource attachments

```
// AWS Transfer SFTP server
new CfnServer(this, 'sftpServer', {
    domain: 'S3',
    endpointType: 'VPC',
    identityProviderType: 'SERVICE_MANAGED',
    loggingRole, // CloudWatch Log Role
    protocols: ['SFTP'],
    endpointDetails: {
        addressAllocationIds, // Elastic IPs
        vpcId,
        subnetIds,
        securityGroupIds: // SecurityGroup Ids,
    },
    certificate, // ACM Certificate
});
```

L2 Constructs

- Level 2 Constructs are native CDK implementations of the CloudFormation resources
- L2 constructs simplify creation of resources, and include sensible/secure defaults
- The constructs expose additional configuration, and can easily be linked together
- The example here creates an S3 Bucket with versioning enabled and public access blocked
- The bucket config includes CDK customization that allow us to delete a bucket including objects – not possible in CloudFormation
- We then create a Lambda Function, and grant it access to put objects into the bucket

```
// S3 Bucket
const archiveBucket = new Bucket(this, 'ArchiveBucket',
{
    versioned: true,
    blockPublicAccess: BlockPublicAccess.BLOCK_ALL,
    removalPolicy: RemovalPolicy.DESTROY,
    autoDeleteObjects: true, // CDK custom resource to
delete objects on Stack delete
});

// Archive function
const archiveFnc = new Function(this, 'ArchiveFnc', {
    description: 'Lambda Archive Function',
    runtime: Runtime.NODEJS_16_X,
    handler: 'index.handler',
    code:
Code.fromAsset(`${__dirname}/lambda/archive`),
    environment: {
        ARCHIVE_BUCKET: archiveBucket.bucketName,
    },
});
archiveBucket.grantPut(archiveFnc); // Adds bucket
access to the Lambda execution role
```

L3 Constructs

- Level 3 Constructs build on L2 by combining them together, adding additional build options, or adding additional custom functions
- L3 constructs are included in the CDK repo and are shared by the developer community
- They are very easy to create in your own projects!
- The example here creates a Lambda function from a Typescript source – using esBuild to bundle and compile to Javascript

```
// TypeScript Lambda Function
const fnc = new NodejsFunction(this, 'Fnc', {
    description: 'Lambda function from Typescript
source',
    runtime: Runtime.NODEJS_16_X,
    logRetention, // CDK customisation to set CloudWatch
log retention
    layers, // Add Lambda layers
    bundling: {
        sourceMap: true,
        externalModules: ['moment'], // Exclude modules
included in a layer
    },
    entry: `${__dirname}/lambda/my-fnc/index.ts`,
    environment: {
        NODE_OPTIONS: '--enable-source-maps',
    },
});
```

Create your own L3 Constructs

- Example use case:
- Create an S3 Bucket with encryption enabled, public access blocked, retention policies set, access logs
- In addition, the custom resource has created a dedicated access-logs bucket and configured logging
- (not showing the code behind the custom construct here because it is a bit too long)

```
// Standard S3 Bucket
new Bucket(this, 'ArchiveBucket', {
    versioned: true,
    blockPublicAccess: BlockPublicAccess.BLOCK_ALL,
    removalPolicy: RemovalPolicy.DESTROY,
    autoDeleteObjects: true, // CDK custom resource to
delete objects on Stack delete
});

// Custom S3 Bucket (with the same settings)
new JtrbS3Bucket(this, 'ArchiveBucket', {});
```

CDK AWS Custom Resources

- CloudFormation can be extended using Lambda functions
- Very useful for getting/setting configuration or interacting with other systems
- Creating and managing Lambda Custom Resources in CloudFormation can be tricky, but CDK makes it very simple
- The example here configures SMS settings in SNS
- This is the simple version using standard AWS SDK API calls. It is also possible to create fully custom Lambda functions for more complex logic

```
// Configure SNS SMS Settings using custom resource
new AwsCustomResource(this, 'SnsConfig', {
    onUpdate: {
        service: 'SNS',
        action: 'setSMSAttributes', // Javascript SDK
Method
        parameters: {
            attributes: {
                MonthlySpendLimit: '20',
                DefaultSenderID: 'MyCompany',
                DefaultSMSType: 'TRANSACTIONAL',
                UsageReportS3Bucket: 'reports-bucket',
            },
        },
        physicalResourceId: {},
    },
    // Add permissions to the custom function based on the
SDK methods
    policy: AwsCustomResourcePolicy.fromSdkCalls({
resources: ['*'] }),
});
```

Demo – CDK and Lambda PowerTools

THANK YOU

Find me and some more examples at:

- Medium: https://markilott.medium.com/
- GitHub - https://github.com/markilott
- LinkedIn - https://www.linkedin.com/in/markilott/