healthverity

AWS CDK & Projen

Running IaC along with your application

May, 2023

Synchronize the Science℠

cposc.org · Lancaster, PA · Saturday, April 6

Agenda

- → CloudFormation refresher
- → Getting started with CDK
- → All about constructs
- → CDK demo
- → Testing your infrastructure
- → All about Projen
- → Projen demo
- → Why *not* CDK/CFN?

CDK & CloudFormation

CloudFormation

As a Service

- IaC (technically)
- Manages resources and their state in a black box
- Can interact with other services such as CodePipeline

Templates

- Templates written in JSON or YAML
- Hard to get real-time validation
- Template files can get large (1,000+ lines)
- Often difficult to read
- Has certain intrinsic functions

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Main stack with an EC2 instance and a nested stack

Parameters:
  InstanceType:
    Type: String
    Description: EC2 instance type
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t3.micro
      - t3.small

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0abcdef1234567890
      InstanceType: !Ref InstanceType
      KeyName: MyKeyPair
      SecurityGroups:
        - default

  MyNestedStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: https://s3.amazonaws.com/example-bucket/nested-stack.yaml
      Parameters:
        NestedParameter1: "Value1"

Outputs:
  EC2InstanceId:
    Description: "The Instance ID of the created EC2 instance"
    Value: !Ref MyEC2Instance
```

What is CDK?

Amazon's Cloud Development Kit (CDK)

Takes runtime languages such as Python and TypeScript and compiles them to CloudFormation.

Released in 2019, v2 released in 2021

Utilizes Amazon's JSII project to create polyglot libraries from its TS core.

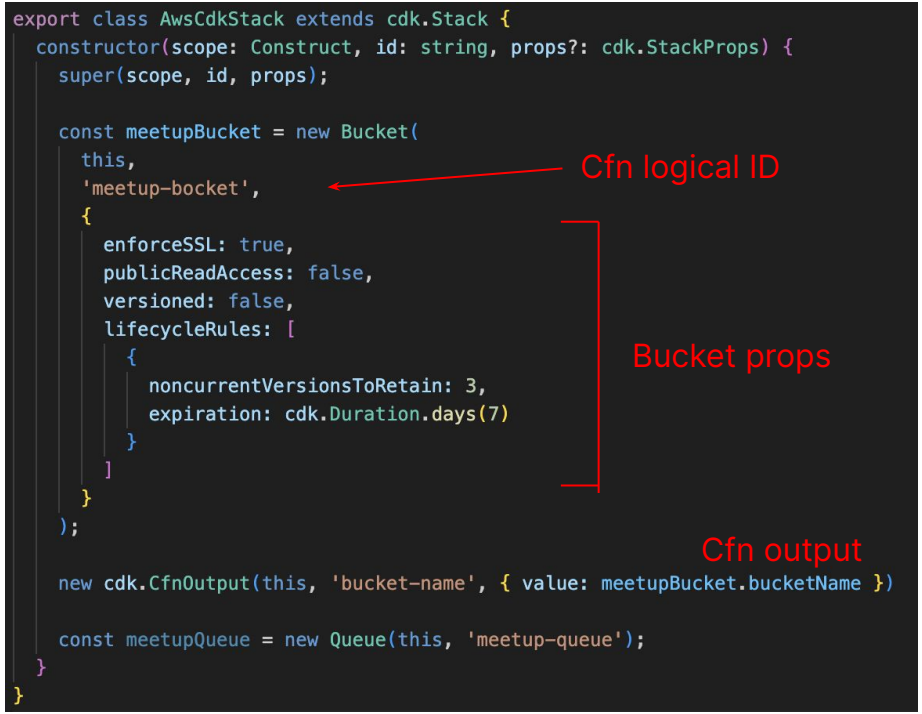
```typescript
import * as cdk from '@aws-cdk/core';
import * as ec2 from '@aws-cdk/aws-ec2';
import { NestedStack } from './nested-stack';

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

    // Define a VPC
    const vpc = new ec2.Vpc(this, 'MyVPC', {
      maxAzs: 2, // Default is all AZs in the region
    });

    // Define an EC2 instance
    new ec2.Instance(this, 'MyInstance', {
      instanceType: new ec2.InstanceType('t2.micro'),
      machineImage: new ec2.AmazonLinuxImage(),
      vpc,
    });

    // Include the nested stack
    new NestedStack(this, 'MyNestedStack');
  }
}
```

What are Constructs?

Each resource is represented as a “Construct”, which can be made up of other constructs.

A stack is a construct, an RDS cluster is a construct, an S3 bucket is a construct! (You get it.)

Each construct is treated as an object, complete with its own constructor and member functions.



To see constructs for AWS CDK, CDK8s, CDKTF, and more, check out the AWS construct hub: https://constructs.dev/

More on Constructs

Constructs come in different “levels” or abstractions.

“Level 1” constructs closely resemble CloudFormation resources. “Level 2” are more developer-friendly representations, and “Level 3” are often a group of resources abstracted away for convenience.

Anything you can create in Cfn, you can create in CDK.

Any level 1 constructs start with “Cfn” and are highly representative of how they would be defined in a raw CloudFormation template.

Level 2 constructs are easier to read and have better documentation on how to interact with them.

Level 3 constructs provide a convenient way to spin up complicated architectures without a lot of code.

How does CDK work?

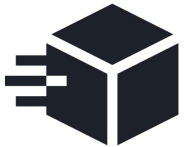
At the end of the day, CDK still compiles down and deploys CloudFormation.

Write code

Write your IaC as you normally would, saving your work in some versioning control like Git or SVN.

Synthesize (“synth”)

Code compiles down to a CloudFormation template and corresponding deployment packages.



Deploy

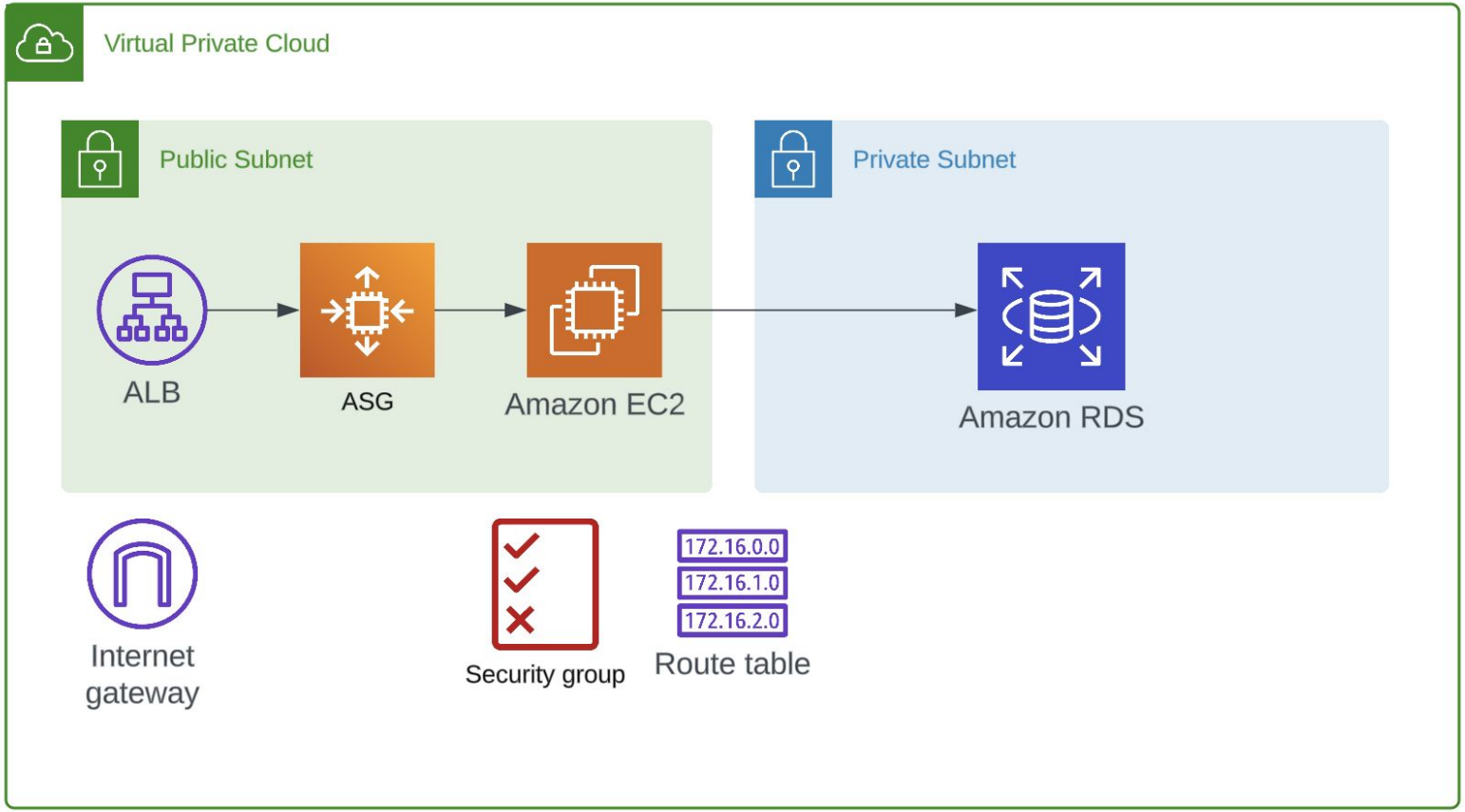
Cloud assembly (template + code) uploads to AWS and calls the CloudFormation API to deploy changes.

Getting started with AWS CDK

```
$ > npm install -g aws-cdk

$ > cdk bootstrap aws://[account-number]/[region]
```

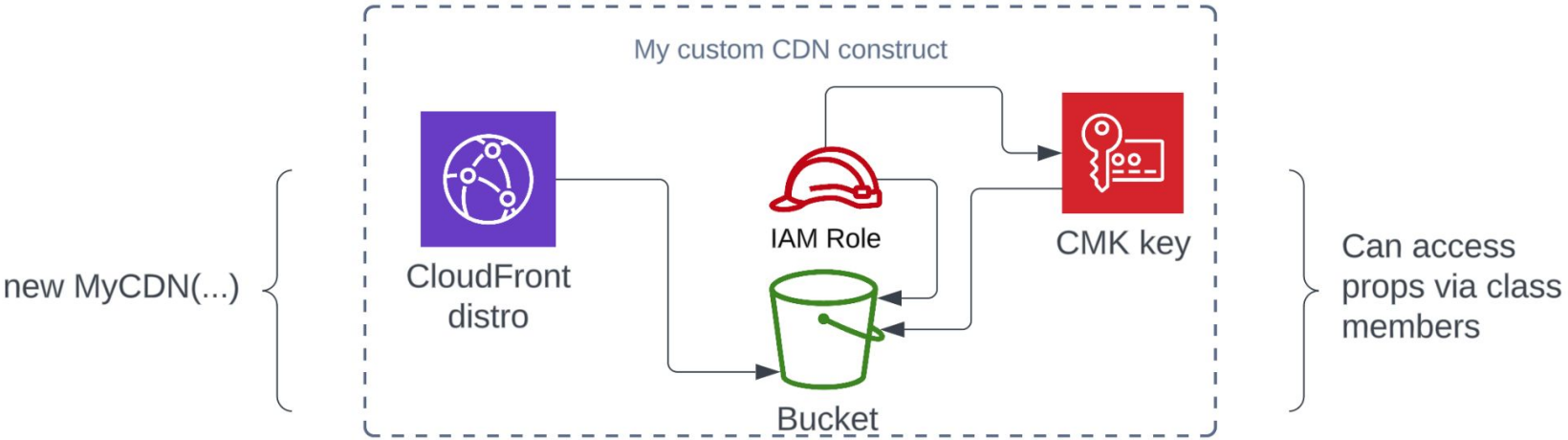
Deploys a CloudFormation stack containing resources needed for CDK

```
$ > cdk init app --language typescript

// Code...

$ > cdk synth

$ > cdk deploy
```



Authoring custom constructs

Just as you can define custom Terraform modules or Pulumi components, you can define custom constructs.



Unit test your infrastructure

You can run fine-grained assertions on resource properties, or assert the full JSON object.

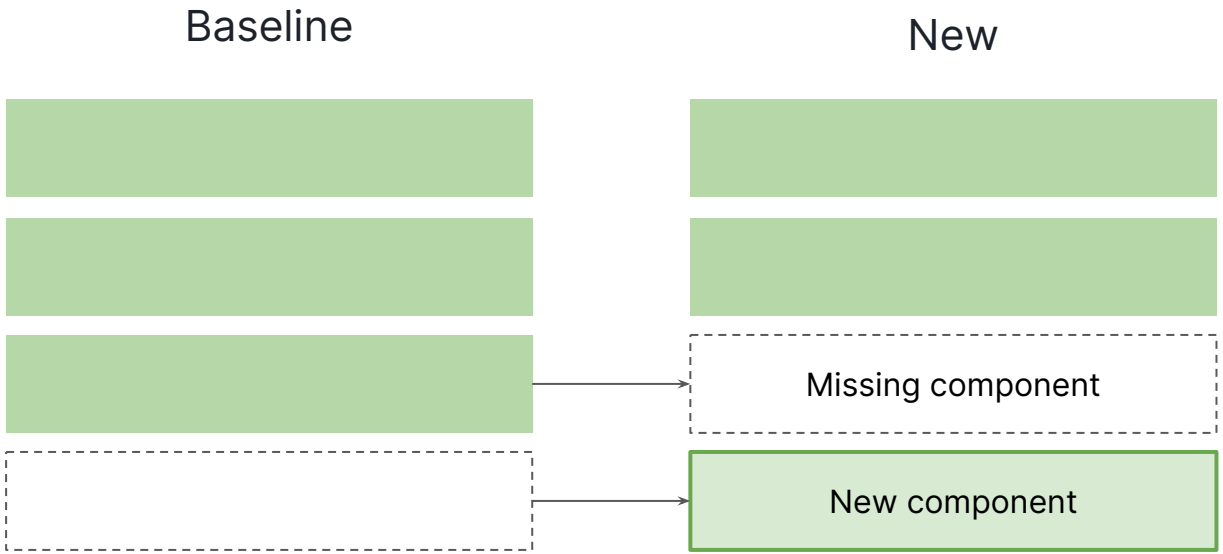
```
// Assert functions have the correct runtime...
template.hasResourceProperties("AWS::Lambda::Function", {
  Handler: "handler",
  Runtime: "nodejs20.x",
});

// Check that the subscription has been created...
template.resourceCountIs("AWS::SNS::Subscription", 1);
```

```
// Fully assert on the state machine's IAM role with matchers.
template.hasResourceProperties(
  "AWS::IAM::Role",
  Match.objectEquals({
    AssumeRolePolicyDocument: {
      Version: "2012-10-17",
      Statement: [
        {
          Action: "sts:AssumeRole",
          Effect: "Allow",
          Principal: {
            Service: {
              "Fn::Join": [
                "",
                ["states.", Match.anyValue(), ".amazonaws.com"],
              ],
            },
          },
        },
      ],
    },
  })
);
```

Snapshot tests

You can run tests to compare entire stacks to a stored baseline template.

Though it doesn't catch "regressions", it does catch changes between stacks.



Dos & Don'ts

Do

- Build constructs to represent business functions, small and discrete
- Refactor early and often
- Snapshot test stacks
- Stateful resources can be placed in their own stack*

Don't

- Import other resources into your custom resources, that should be handled at the stack level
- Bind a stack to a specific environment (acct + region)**
- Use environment variables in your infrastructure definition
- Use **external data** in your IaC

External Data

Though technically you *can* make API calls and other functions at time of synthesis, it is highly not recommended, because it will lead to some weird states.

```
async function someApiCall() : Promise<ScanCommandOutput> {
  return DynamoDBDocumentClient.from( client: new DynamoDBClient( [configuration]: {}))
    .send( command: new ScanCommand( input: { TableName: 'MyServiceAccountUsers' }));
}

someApiCall().then(data : ScanCommandOutput => {
  new ServiceAccountUsersStack( scope: app, id: 'ServiceAccountUsers', props: {
    serviceAccounts: data.Items!,
  });
}).catch(console.error);
```

Projen

What is projen?

"**Projen** allows you to define and maintain complex **project configuration through code**. It lets you generate, or synthesize project configuration files from a well-typed definition."

- *Introduction, Projen*

What is projen?

Project file management done for you

Things like package.json, requirements.txt, and more are managed by projen for you.

Automation built in to keep things up to date

Automatic weekly PRs into your repository to update dependencies and even projen itself.

Built on JSII

Like CDK, Projen is built on JSII so you can synthesize any project into any supported language.

Standard npm tooling, with easy ways to extend them

CLI commands can be defined, customized, and extended easily during project authoring.

CHANGELOG files generated based on commit history

When creating releases in Github, you can generate release notes with just a click of a button.

Easily customizable and extendable

You can create your own project types with your own specifications, generating new workflows and automation processes easily.

Project types

Projen provides tooling for not just AWS CDK, but many other languages and frameworks.

AWS CDK

- awscdk-app-java
- awscdk-app-py
- awscdk-app-ts
- awscdk-construct

Other CDKs

- cdk8s-app-py
- cdk8s-app-ts
- cdk8s-construct
- cdktf-construct

Node & TS

- node
- typescript
- typescript-app

Web

- nextjs
- nextjs-app
- react
- react-ts

Other

- java
- python
- jsii
- project

Getting started with Projen

```
$ > npx projen new awscdk-app-ts
```

```
$ > alias pj=”npx projen”
```

An easy alias to make future commands easier

```
$ > pj synth
```

```
$ > pj deploy
```

Projen Demo

Creating, synthesizing, and deploying a project

Extending projen

You can extend projen project types with your own classes, enabling the ability to do things like

- Get customized CI/CD pipelines out of the box
- Unit testing and code scanning
- Common libraries, tooling, and dependencies – like linters!

```
pytest: true, /* Include pytest tests. */
cdWorkflowsOptions: {
  environments: [
    {
      name: 'dev',
      awsAccountId: '1234567890',
      instancePrefix: 'hv-env-dev',
      deployOnMerge: true,
      cdkDeployParameters: [
        { name: 'ReplicaRegions', value: 'us-west-2'},
        { name: 'DeployMap', value: '...'},
        { name: 'NotificationEmail', value: 'lfarr@healthverity.com'},
        { name: 'OidcThumbprints', value: '6938fd4d98bab03faadb97b34396831e3780aea1,1c58a3a8518e8759bf075b76b750d4f2df264fcd'},
      ]
    },
```

Why not CDK?

And when wouldn’t you want to use CloudFormation?

Why not CDK/CFN?

SST, an IaC framework for building full-stack applications on AWS, recently announced that they're moving away from CDK. Why?

- Creating infrastructure with CloudFormation is a black box. You pass in some properties, and it does some things under the hood, then *bam* you have infra.
- "CDK is not IaC, it's a CFN hack."
  - The order you code in is not the order it gets deployed in.
  - Certain operations cannot be done because it gets synth'd into one big JSON object.
  - Rollback hell – it's slow, it can get stuck, things get weird.

Why not CDK/CFN? (cont.)

SST cites other practical problems with their approach:

- Resources needing information from each other need two separate deployments. *Ex. a next.js site with an env var of a Dynamo table, but needing to deploy the table first to retrieve it.*
- Cyclical dependencies are hard. When Resource A needs a property from Resource B, but B needs A to deploy, it's hard to sort the two out.
- "Export in use" – stack resources that are imported into other stacks... who's managing who?
- And more

... But are you doing the same kind of work?

Many problems in CloudFormation can be navigated effectively through good architecture and change in reference. Things like stack limits, cross-stack resources, and dependencies are entirely avoidable.

If you're building frameworks to provision infrastructure directly through the use of AWS APIs, you may want to consider different paths depending on what is important to you.

Thank you

Logan Farr
lfarr@healthverity.com

Synchronize the Science℠

Appendix

- → AWS CDK Github repo
- → Getting started with AWS CDK docs
- → AWS JSII project
- → Advanced AWS CDK: Lessons learned from 4 years of use
- → projen.io
- → Moving away from CDK, sst.dev