TNG TECHNOLOGY CONSULTING

AWS-CDK Workshop

Thomas Kiendl
Tom Zirke
Tom Kriel

2021-04-22

Cloud Development Kit - Why?

Kit: "A collection of items needed for a specific purpose" (wiktionary)

- collection of items: more on that later... (primer: constructs)
- purpose: defining and deploying cloud infrastructure

Alternatives:

- Imperative: AWS CLI and AWS SDKs (java, python, typescript, ...)

```
aws s3api create-bucket --bucket "Workshop Bucket"
```

  - Problem: Life Cycle Management
- Declarative: AWS Cloudformation

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  WorkshopBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: "Workshop Bucket"
```

Cloudformation

Some problems

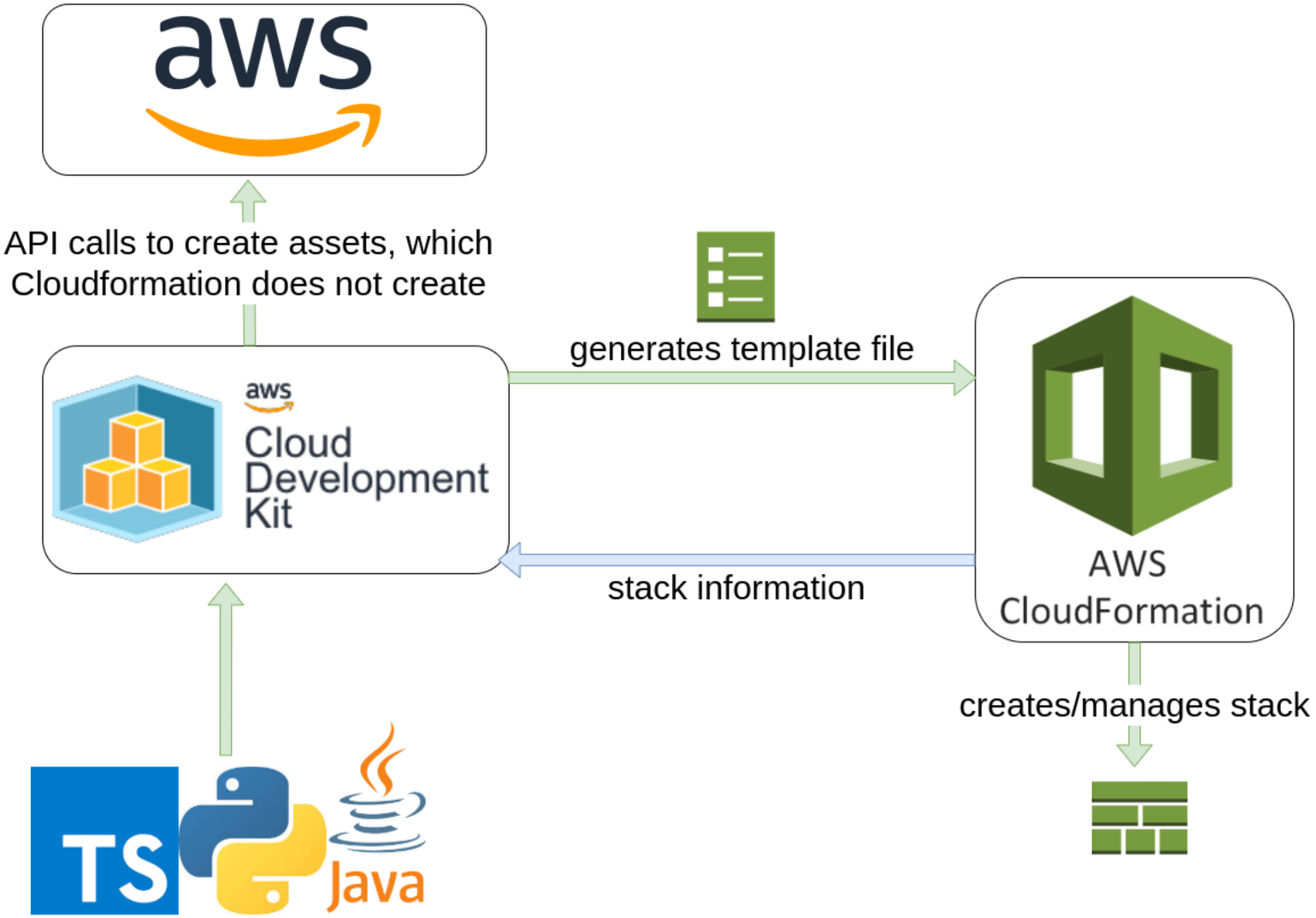
- Lots of boilerplate
- Complex and verbose permission management (IAM)
- Missing features of modern programming languages (e.g. hard to define abstractions)

```yaml
Resources:

  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Ref 'ServiceName'
      Cpu: !Ref 'ContainerCpu'
      Memory: !Ref 'ContainerMemory'
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn:
        Fn::ImportValue:
          !Join [':', [!Ref 'StackName', 'ECSTaskExecutionRole']]
      TaskRoleArn:
        Fn::If:
          - 'HasCustomRole'
          - !Ref 'Role'
          - !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: !Ref 'ServiceName'
          Cpu: !Ref 'ContainerCpu'
          Memory: !Ref 'ContainerMemory'
          Image: !Ref 'ImageUrl'
          PortMappings:
            - ContainerPort: !Ref 'ContainerPort'

  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: !Ref 'ServiceName'
```

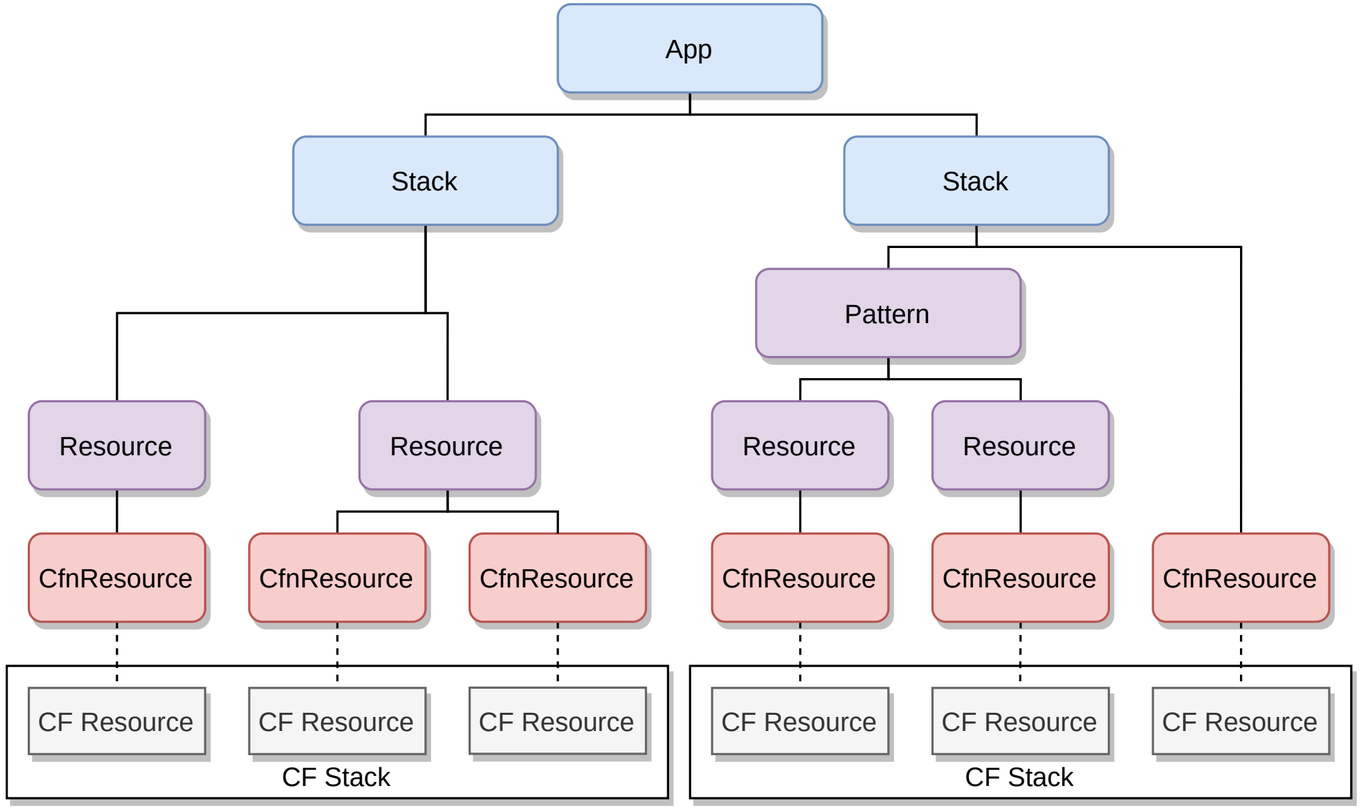
AWS-CDK basics

What is the AWS-CDK?

With the CDK you can write infrastructure as a code for AWS in a high-level programming language



Code is structured as a tree of so-called constructs



- Patterns: Many intertwined resources (e.g. `ApplicationLoadBalancedFargateService`)
- Resource: High level (e.g. `s3.Bucket`)
- CF Resources: Low level, same interface as in Cloudformation up to casing (e.g. `s3.CfnBucket`)

Constructs

The constructor of a construct has the following signature:

```
class SomeConstruct extends cdk.Construct {
    constructor(scope: cdk.Construct, id: string, props?: SomeProps) {
        super(scope, id, props);
        // Instantiation of child constructs
    }
}
```

- **scope**: Determines parent in tree structure. Constructs gets initialized in constructor of parent and scope should be set to **this** (the parent).
- **id**: Cloudformation remembers state of resources via the name. CDK generates name of construct using its ID and the IDs of its ancestor constructs. Hence IDs have to be unique in each scope.
- **props**: The parameters used to customize the construct.

Time to deploy

- `cdk synth`: Create the cloudformation template
- `cdk diff`: What changes will be made to our stack when we deploy?
- `cdk deploy`: You are good to go? Time to change the world.
- `cdk destroy`: Deletes the stack.

Set the environment variables accordingly to specify your AWS profile

```
AWS_PROFILE=playground AWS_REGION=eu-west-1 cdk deploy
```

Advanced topics

Referencing existing AWS resources in your stack

- In order to reference an AWS resource from outside your CDK app, go to the high-level CDK class representing this resource and look for static factory methods:

```
const myBucket1 = s3.Bucket.fromBucketName(this, 'MyBucket1', 'my-bucket-name');
const myBucket2 = s3.Bucket.fromArn(this, 'MyBucket2', 'arn:aws:s3:::my-bucket-name');
```

- Remember that inside your app, you can pass references of constructs to other constructs. There is normally no need to use identifiers like the arn.

Customization of constructs

- First, try to customize via the props argument of the constructor.

```
const bucket = new s3.Bucket(this, 'bucket', {bucketName: 'example'})
```

- Constructs expose methods to attach or manipulate resources

```
bucket.addEventNotification(
    s3.EventType.OBJECT_CREATED,
    new LambdaDestination(myFunction)
);
```

- Via fields constructs expose dependent resources.

```
const bucketPolicy = bucket.policy
```

- As a last resort, you can add "raw" CF-template code in your app and add overriding rules (Link to documentation)

Testing

CDK ships with @aws-sdk/assert library. Main features are

- Snapshot tests: Generate a cloudformation template and compare it with the previously saved master template:

```
const stack = new MyStack();
expect(SynthUtils.toCloudFormation(stack)).toMatchSnapshot();
```

- Test whether resources matching a set of sub-properties are created:

```
expect(stack).to(haveResourceLike(
    'AWS::S3::Bucket',
    {BucketName: 'TestBucket'},
));
```

Warning: Tests run against the CF template and properties adhere to the Cloudformation interface.

Aspects

Aspects offer a convenient way to apply some operations to all constructs in scope.
They are used internally by CDK for tagging resources.

- Aspects implement the visitor pattern:

```
interface IAspect {
visit(node: IConstruct): void;}
```

- Also useful to check global constraints to all resources of a type, e.g.

```
class BucketNameChecker implements IAspect {
  public visit(node: IConstruct): void {
    if (node instanceof s3.CfnBucket) {
      if (!node.bucketName.startsWith(`${process.env.AUTHOR}-`)) {
        node.node.addError('Bucket name does not start with author name');
      }
    }
  }
}
```

  Don't forget to apply this to the stack

```
Aspects.of(stack).add(new BucketNameChecker());
```

- More on aspects: here

Higher level IAM API

- Many resources expose methods to attach high-level policys directly.

```
const bucket = new s3.Bucket(this, 'bucket', {bucketName: 'example'});
const lambdaFunction = lambda.Function.fromFunctionArn(this,'my-lambda', 'some-lambda-ar
bucket.grantRead(lambdaFunction);
bucket.grant(lambdaFunction, 's3:PutObject')
```

- Many commands involving multiple AWS resources create the necessary permissions for interaction automatically.

Pros and Cons

- ✓ Type-safety and code completion
- ✓ Avoid code duplication by writing your own constructs (collections of AWS resources)
- ✓ Test IaC
- ✓ For each resource (combination) there can be arbitrary many default configurations which greatly reduces boilerplate code
- ✓ Higher-level API for permission management
- ✓ IaC concepts can be shared across teams and best practices can be enforced by the organization
- ✓ Validation of parameters possible
- ✓ Conditional resource creation easier (for instance prod and dev)
- ✖ High-level constructs set many parameters implicitly. Danger of misconfiguration and too broad permissions (Possible solution: Snapshot tests for the resulting template)
- ✖ Dependency on CloudFormation: Super new resources could be available in CloudFormation but not in CDK yet

Authoring your own constructs

- Constructs have to be a subclass of cdk.Construct.
- Expose instantiated resources as properties to allow customization.

```
class MyBucketLambdaConstruct extends cdk.Construct {
    // Expose child constructs to allow customization
    readonly bucket: s3.Bucket;
    readonly lambda: lambda.Function;

    constructor(scope: cdk.Construct, id: string, props?: MyBucketProps) {
        super(scope, id);

        // Instantiation of child constructs
        const bucket = new s3.Bucket(...);
        const myLambda = new lambda.Function(...)

        // now combine these two in some way
    }

    customizeSomething(someInput: any) {
        // some code
    }
}

interface MyBucketProps {
    lambdaProps: lambda.FunctionProps
    bucketName: string
}
```

Migration From Cloudformation to CDK

- Step-by-step migration is possible by including Cloudformation templates into the CDK app (use cloudformation-include package).

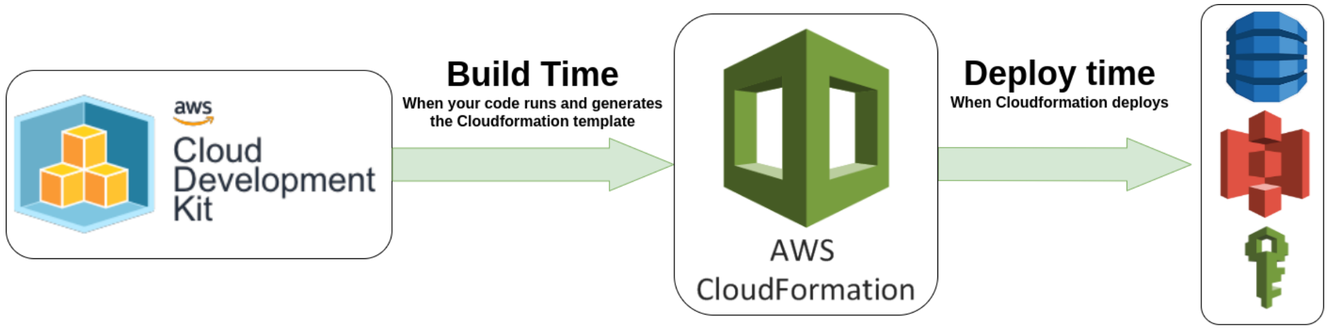
```
  new cloudformationInclude.CfnInclude(this, "ExistingInfrastructure", {
          template: 'path-to-yaml-file'
      })
```

- For stateful resources one has to overwrite the automatically generated logical ID.

```
const bucket = new s3.Bucket(this, 'bucket', {bucketName: 'example'});
const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;
cfnBucket.overrideLogicalId("LogicalId");
```

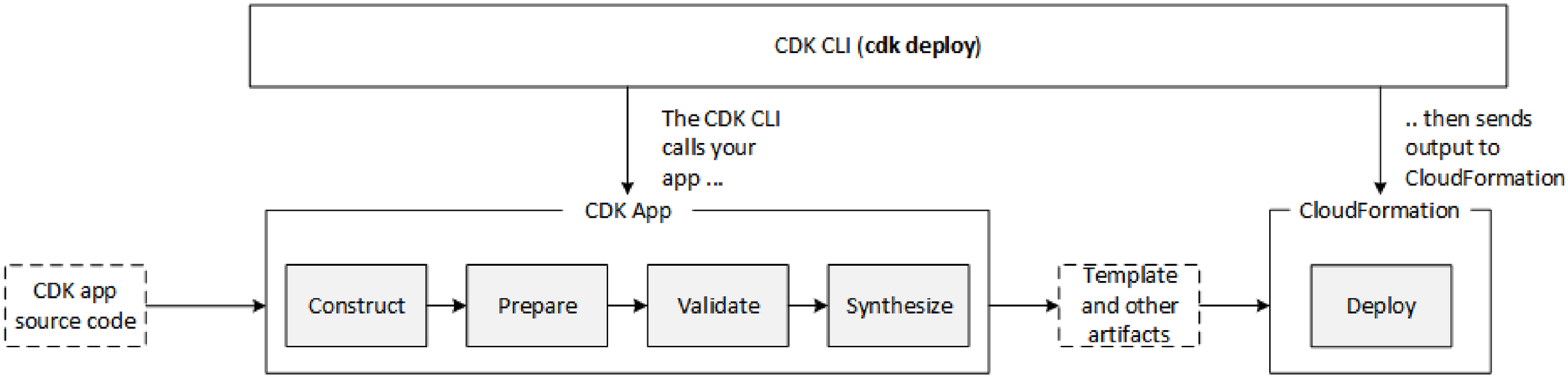
- It can be a bit painful to migrate stateful Cloudformation stacks to totally clean CDK apps. An example is to migrate away from CF parameters which are discouraged in CDK apps.

Tokens



- Some values are only resolved at deploy time.
- Examples: A service's IP address, automatically generated bucket names, ...
- Instead of the actual value, tokens are used:
  `"${TOKEN[Bucket.Name.1234]}"`
- They can be used like normal values and passed to constructs.
- However string comparison is of course not possible.

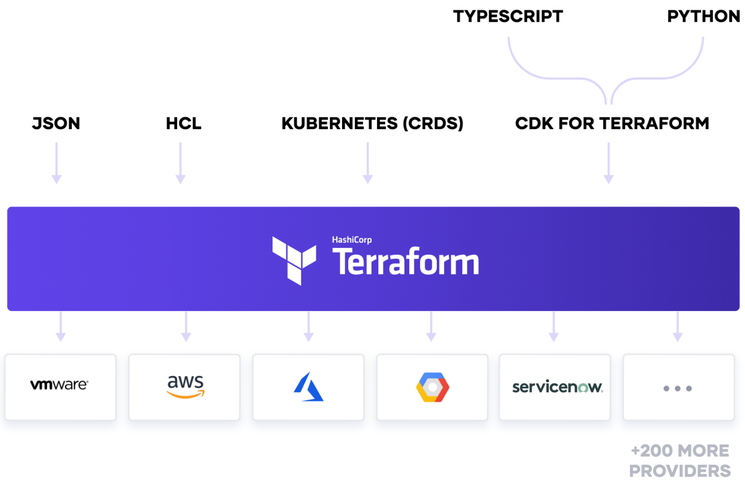
CDK lifecycle



- **Construction**: All constructors are invoked.
- **Validate**: You can implement the **validate** method but it is preferred to throw errors in the Construct phase.
- **Synthesis**: Prepare artifacts for the deployment (CloudFormation template and assets like Docker images or Lambda code)

Beyond CloudFormation

cdktf (CDK for Terraform)



- Generate HCL terraform code
- github.com/hashicorp/terraform-cdk
- not production-ready yet

cdk8s (CDK for Kubernetes)

kubernetes

- Declare infrastructure in Typescript or Python

Further Outlook

- AWS CDK Pipelines
  - High-level configuration of CI/CD pipelines in CDK app
  - Self-modifying automatically
  - Still in developer preview
  - Only CodeCommit and Github supported
- CDK v2 coming up
  - Simplified module structure
  - Better backwards compatibility
- Growing Ecosystem
  - AWS Solutions Constructs: https://aws.amazon.com/solutions/constructs/
  - CDK Patterns: https://cdkpatterns.com/
  - Awesome CDK: https://github.com/kolomied/awesome-cdk