Deep dive on CDK

Developing constructs & libraries

Andreas Sieferlinger

@webratz@hachyderm.io

https://github.com/webratz/cdk-lib-examples

Personio

Who has already actively worked with AWS CDK?

Who has already written CDK constructs and shared them in any way?

I want to create my own CDK construct library!

But how do I get started?

What language do I write this in?

What tools do I need?

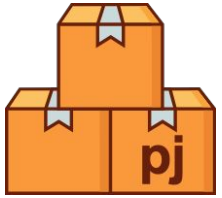
How can I make it easy to use and configure?

How do I make sure this continues this continues to work?

Project setup

01

Should I use projen?



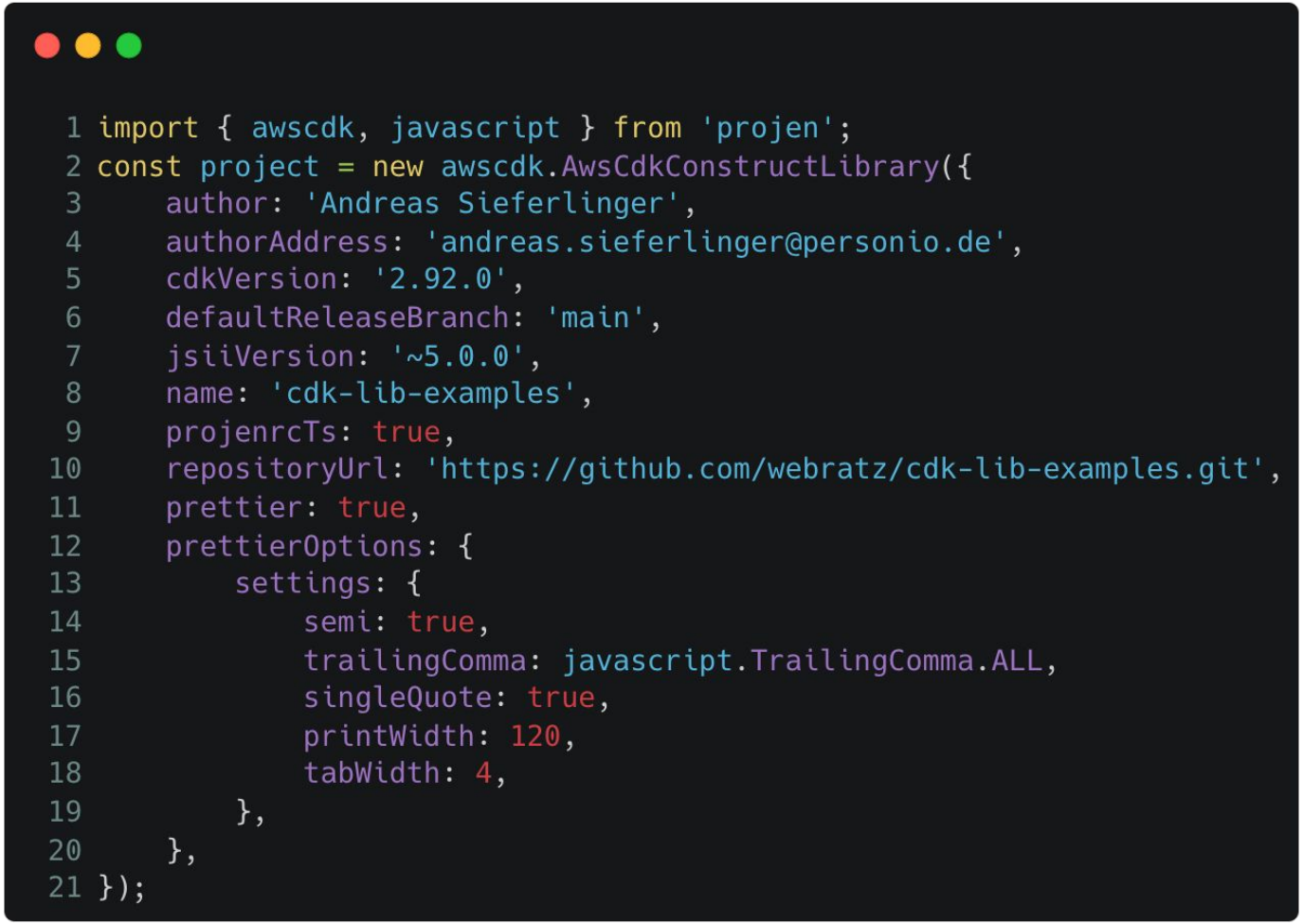
What is projen even?

Projen is a project generator - like a cookiecutter, but re-generates each file every time.

- Same origins as CDK
- Similar concepts as CDK
- Typescript
- One single config file for complete project setup
- Opinionated
- Ensures state = overwrites manual changes

When to use?

- Great if you don’t have a company wide standard project setup (for TS) yet
- Happy with default toolchain and setup provided by it
- Fine to invest bit more time to learn and have slightly different workflow



```
import { awscdk, javascript } from 'projen';
const project = new awscdk.AwsCdkConstructLibrary({
    author: 'Andreas Sieferlinger',
    authorAddress: 'andreas.sieferlinger@personio.de',
    cdkVersion: '2.92.0',
    defaultReleaseBranch: 'main',
    jsiiVersion: '~5.0.0',
    name: 'cdk-lib-examples',
    projenrcTs: true,
    repositoryUrl: 'https://github.com/webratz/cdk-lib-examples.git',
    prettier: true,
    prettierOptions: {
        settings: {
            semi: true,
            trailingComma: javascript.TrailingComma.ALL,
            singleQuote: true,
            printWidth: 120,
            tabWidth: 4,
        },
    },
});
```

Language choices

Which language to choose for your library?

typescript

- Same language as upstream CDK
- Can be used with other supported languages via jsii
- Wide range of examples
- Easy to learn
- Can be easier to debug with upstream CDK

[python|java|golang|c#]

- Only target is users in exactly one of these languages
- Technically can’t use typescript (eg organizational restrictions)

jsii

Friend or foe?

pro

- Code can be used with all target languages
- Many checks & rules to avoid incompatibilities
- Create simple API docs out of the box

con

- Restricts several typescript language features
- 3rd party libs might be incompatible
- Used to only support outdated typescript versions (fixed in recent releases)

Constructs

02

Passing properties to *default* constructs

Constructs that add defaults to existing constructs

Directly adding additional properties

- Quick and easy be extending original interface
- Risk of overlapping with existing or later added properties
- Useful for “default” constructs with few additional settings

Only expose custom settings

- Restricts users to only allowed parameters
- More complex to override defaults

New Interface with original props

- Full flexibility
- Clear what belongs where (for multiple resources)
- Slightly different interface than default
- Use Partial<>

```typescript
/**
 * This extends the existing bucket properties with our own settings
 */
export interface CustomBucketProps extends s3.BucketProps {
    /**
     * Set to true if DataAccess can interact with the bucket
     * @default false
     */
    readonly dataAccessEnabled?: boolean;
}

/**
 * Create a private / internal S3 Bucket with all required settings to be compliant
 * The defaults can not be overriden out of the boy by a user in this example

 */
export class CustomBucket extends Construct {
    public bucket: s3.Bucket;
    constructor(scope: Construct, id: string, props: CustomBucketProps = {}) {
        super(scope, id);

        this.bucket = new s3.Bucket(this, 'CustomBucket', {
            ...props, // whatever the user sets in props, if its set below it will be overwritten
            // basic settings for really all S3 Buckets that are private
            blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
            enforceSSL: true,
            encryptionKey: Alias.fromAliasName(this, 'KmsKeyAlias', 'data'),
            bucketKeyEnabled: true,
            versioned: true,
        });

        Tags.of(this.bucket).add('dataAccess:enabled', String(props.dataAccessEnabled));
```

```typescript
export interface CustomBucketPropsHidden {
    /**
     * Set to true if DataAccess can interact with the bucket
     * @default false
     */
    readonly dataAccessEnabled?: boolean;

    /**
     * Should bucket versioning be enabled
     */
    readonly versioned?: boolean;
}

/**
 * Create a private / internal S3 Bucket with all required settings to be compliant
 * The defaults can not be overriden out of the boy by a user in this example

 */
export class CustomBucketHidden extends Construct {
    public readonly bucket: s3.Bucket;
    constructor(scope: Construct, id: string, props: CustomBucketPropsHidden = {}) {
        super(scope, id);

        this.bucket = new s3.Bucket(this, 'CustomBucketHidden', {
            blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
            enforceSSL: true,
            encryptionKey: Alias.fromAliasName(this, 'KmsKeyAlias', 'data'),
            bucketKeyEnabled: true,
            versioned: props.versioned ?? true,
        });
```

```typescript
/**
 * This extends the existing bucket properties with our own settings
 */
export interface CustomBucketPropsSeparated {
    /**
     * Set to true if DataAccess can interact with the bucket
     * @default false
     */
    readonly dataAccessEnabled?: boolean;

    /**
     * Properties of original Bucket
     */
    readonly bucketProps?: s3.BucketProps;
    // readonly bucketProps?: Partial<s3.BucketProps>; // This does not work with JSII
}

/**
 * Create a private / internal S3 Bucket with all required settings to be compliant
 * The defaults can not be overriden out of the boy by a user in this example

 */
export class CustomBucketSeparated extends Construct {
    public bucket: s3.Bucket;
    constructor(scope: Construct, id: string, props: CustomBucketPropsSeparated = {}) {
        super(scope, id);

        this.bucket = new s3.Bucket(this, 'CustomBucket', {
            ...props.bucketProps, // whatever the user sets in props, if its set below it will be overwritten
            // basic settings for really all S3 Buckets that are private
            blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
            enforceSSL: true,
            encryptionKey: Alias.fromAliasName(this, 'KmsKeyAlias', 'data'),
            bucketKeyEnabled: true,
            versioned: true,
        });

        Tags.of(this.bucket).add('dataAccess:enabled', String(props.dataAccessEnabled));
```

Naming in CDK / CloudFormation libs

Naming resources

Not explicitly naming resources

- CDK / cloudformation auto generates unique names
- No risk of name clashes!
- Hard to read names

Naming resources

- Predictable resource names
- Easy readable
- Potentially more useful information
- Might require resource destruction on change
- Risk of duplicate names

Tenets for libs

- Needs to be deployable multiple times, within Stack and Account
- Default to not naming things explicit

Naming in CDK / CloudFormation libs

Naming constructs via IDs

- Follow AWS CDK Design Guidelines
- IDs need to be unique in Scope only
- Variables should not be added to IDs - use a new Construct level then
- IDs should be in PascalCase
- Changing IDs is a breaking change / change of contract!

```
1 new CustomBucket(stack, 'ExampleBucket', { bucketName: 'example-bucket' });
```

Avoiding duplicate shared constructs within a Stack

Offload to user

- Create once on higher level
- Pass in as mandatory property
- Inconvenient for users

Singleton pattern

- Detect if resource is already there
- Only created when not yet in tree
- Invisible to users

```ts
export class LowerCaseHelper extends Construct {
    /**
     * will return a lower cased string for the given input
     */
    public static lower(scope: Construct, id: string, inputString: string) {
        const stack = Stack.of(scope);
        const lcInstance =
            (stack.node.tryFindChild('FGEC1999ClcHelper') as LowerCaseHelper) ??
            new LowerCaseHelper(stack, 'FGEC1999ClcHelper'); // scope is stack here, not any other construct. always adding on fixed level in tree

        if (id.includes('${Token')) {
            throw new Error(`id contains a Token string: ${id}`);
        }

        // we need a custom suffix that is stable, to ensure for each value we get a unique resource id
        // We are using id as a stable suffix
        const lowerHelper = new CustomResource(scope, `LowerCase${id}`, {
            serviceToken: lcInstance.lowerCaseProvider.serviceToken,
            properties: {
                inputString: inputString,
            },
        });
        return lowerHelper.getAttString('transformedString');
    }
```

Testing

03

See also Talk at 12:30 in “Wien”
Building Reliable Serverless Applications with AWS CDK and Testing

Integration test vs unit tests

Unit tests

- Quick & easy to write
- Use many, but small ones

Integration test

- Often need more effort & time to create
- Have fewer in number but wider in scope
- Only run once (without change) - do not significantly slow down your pipeline!
- Ensure multiple deployments in same environment can work
- Need to take special care for cleanup of resources with persistent data

```ts
const bucketName = 'cdk-integ-test-custombucket';
export class TestStack extends Stack {
    resource: IBucket;
    constructor(scope: Construct, id: string, props?: StackProps) {
        super(scope, id, props);
        this.resource = new CustomBucket(this, 'MyCustomBucket', {
            bucketName: bucketName,
            removalPolicy: RemovalPolicy.DESTROY,
        }).bucket;
    }
}
const app = new App();
const testCase = new TestStack(app, 'CdkIntegBucketStack', {});

const integ = new IntegTest(app, 'BucketIntegTest', {
    testCases: [testCase],
    // stackUpdateWorkflow: false,
});

const message = integ.assertions.awsApiCall('S3', 'listObjectsV2', { Bucket: bucketName });
message.provider.addToRolePolicy({
    Effect: 'Allow',
    Action: ['s3:ListObjectsV2', 's3:ListObjects', 's3:ListBucket'],
    Resource: ['*'],
});

message.expect(
    ExpectedResult.objectLike({
        Name: bucketName,
    }),
);

```

Reducing repeated test code

Jest beforeEach

- Built in
- Only works for standard patterns

Setup helper function

- Can accept props
- Make use of Partial for less code (even with JSii)

```
1
2  describe('Private S3 Bucket Security configuration', () => {
3      function getTestAssets(props: Partial<CustomBucketProps>) {
4          const app = new App();
5          const stack = new Stack(app, 'TestStack');
6          const bucket = new CustomBucket(stack, 'TestBucket', props);
7          const template = Template.fromStack(stack);
8          return { template, stack, app, bucket };
9      }
10
11     test('stack has no error annotations', () => {
12         const { stack } = getTestAssets({});
13         const annotations = Annotations.fromStack(stack);
14         annotations.hasNoError('*', Match.anyValue());
15     });
16
17     test('Public access is blocked', () => {
18         const { template } = getTestAssets({});
19
20         template.hasResourceProperties('AWS::S3::Bucket', {
21             PublicAccessBlockConfiguration: {
22                 BlockPublicAcls: true,
23                 BlockPublicPolicy: true,
24                 IgnorePublicAcls: true,
25                 RestrictPublicBuckets: true,
26             },
27         });
28     });
29
30     test('SSE Alorithm is KMS', () => {
31         const { template } = getTestAssets({});
32
33         template.hasResourceProperties('AWS::S3::Bucket', {
34             BucketEncryption: {
35                 ServerSideEncryptionConfiguration: [
36                     {
37                         BucketKeyEnabled: Match.anyValue(),
38                         ServerSideEncryptionByDefault: {
39                             KMSMasterKeyID: Match.anyValue(),
40                             SSEAlgorithm: 'aws:kms',
41                         },
42                     },
43                 ],
44             },
45         });
46     });
```

Checking &
enforcing changes on all constructs

04

Hooking into users code

Applying defaults and running checks

On Stack level

- Can be soft enforced via qualifiers (see bootstrapping)
- Easy to remember to use “MyCompanyStack”

Adding hook construct

- Needs to be added as additional resource once in tree
- No need to have custom variant of core construct like Stack
- Bundling of multiple aspects (eg adding cdk-nag)

```ts
export class S3Aspect implements IAspect {
    public visit(node: IConstruct): void {
        if (node instanceof s3.CfnBucket) {
            // enforce versioning on all S3 Buckets
            node.versioningConfiguration = { status: 'Enabled' };
        }
    }
}

/**
 * One way to add the Aspect to your setup and reach everything else in it.
 * Depending on your expectactions this should be set to another scope (eg construct or even App)
 */
export class S3Checks extends Construct {
    constructor(scope: Construct, id: string) {
        super(scope, id);
        const stack = Stack.of(this);
        Aspects.of(stack).add(new S3Aspect());
    }
}
```

The People Operating System