AWS Black Belt Online Seminar

AWS Cloud Development kit (CDK)

Advanced #1
AWS CDKにおける開発とテスト

工藤 朋哉

Prototyping Engineer
2023/12

自己紹介

- 工藤 朋哉
- Prototyping Engineer

- 好きなAWSサービス：  AWS CDK

アジェンダ

1. CDKにおけるテスト
2. 開発にまつわる Tips

CDKにおけるテスト

CDK におけるテストとバリデーション

Snapshot Test

- 合成された CloudFormation テンプレートを
  スナップショットと比較
- 導入が簡単で効果が高くおすすめ
- jest など言語固有のテストコマンドで実行

Validation Test

- アプリ開発におけるユニットテストの一部
- Construct のコンストラクタや addValidation() で
  意図しない入力で例外を送出することをテスト
- jest など言語固有のテストコマンドで実行

Integration Test

- デプロイを行い CloudFormation テンプレートが
  有効であることを確認
- デプロイされたリソースに対するテストも行える
- Integ-runner CLIで実行

Fine-Grained Assertion Test

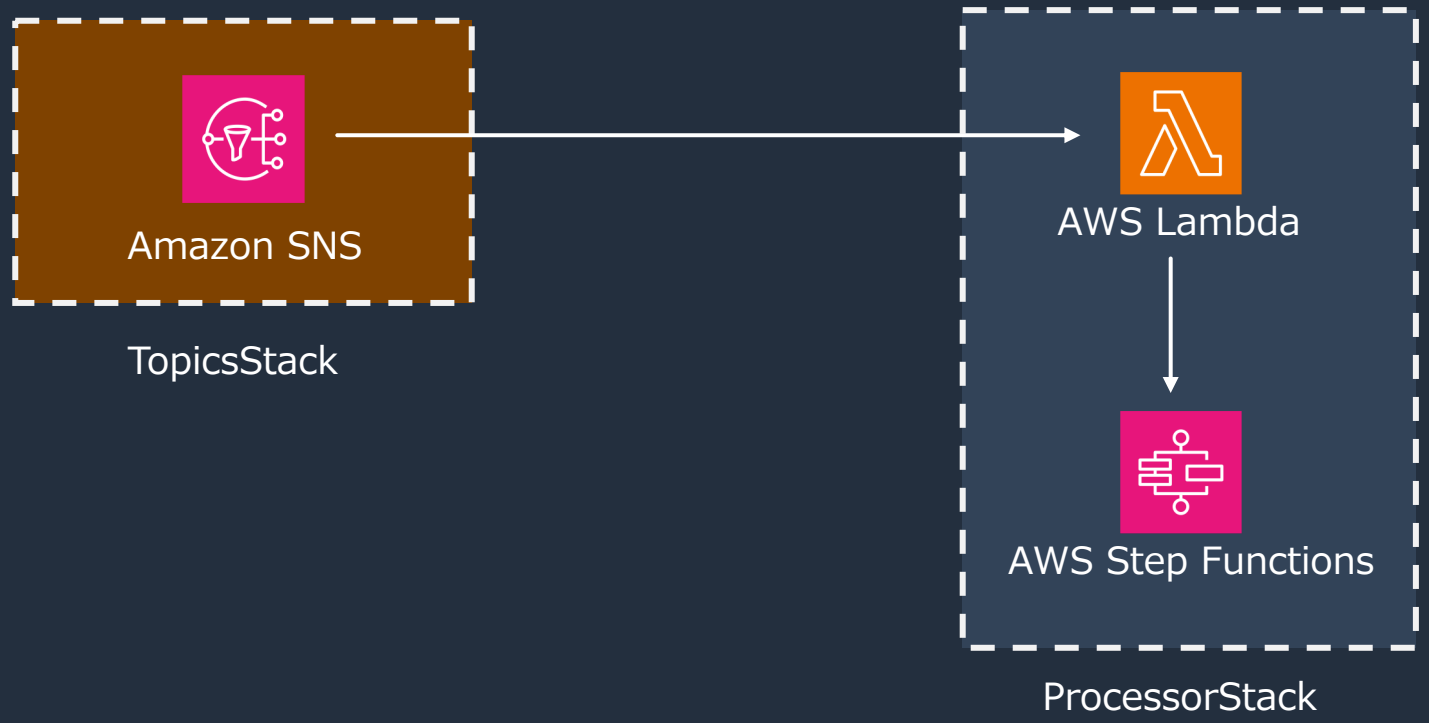
- アプリ開発におけるユニットテストの一部
- Construct が与えられた入力から
  期待通りの CloudFormation テンプレートを
  合成することをテスト
- aws-cdk-lib.assertions モジュールを利用
- jest など言語固有のテストコマンドで実行

Policy Validation

- Construct がポリシーに従って構成されているか確認
  - Aspect を使用して Construct ツリーを巡回し
    違反があれば Annotation をつける (cdk-nag など)
  - CloudFormation テンプレート用の Policy as Code ツールを
    合成後に適用 (CloudFormation Guard など)
- cdk synth 時に実行 （jest などでの実行も可能）

テストシナリオ

- 下図のようなアプリケーションを作成して、各テストの書き方の例を説明します
- スペースの都合上、ソースコードは一部を抜粋しています。
  import文を含む全体のコードは以下のデベロッパーガイドをご覧ください。
  https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/testing.html



サンプルアプリケーションの作成

- CDK アプリケーションを作成

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

- テスト用のディレクトリを作成

```
mkdir test
```

- Jestの設定
  - package.json を編集

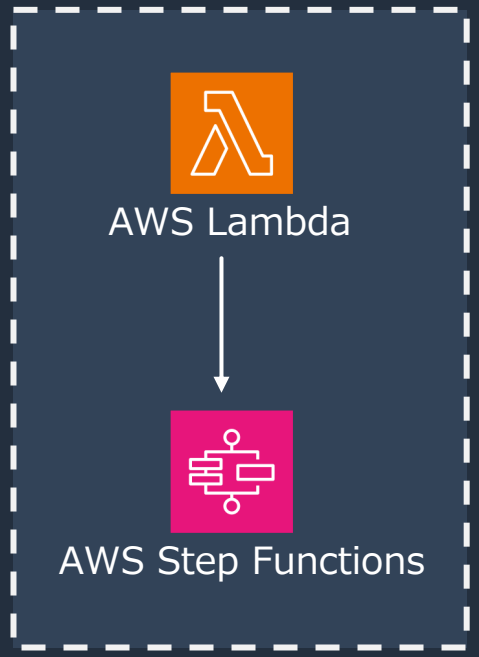
    その後 `npm install`

追加する →

package.json

```
{
  ...
  "scripts": {
    ...
    ”test": "jest"
  },
  "devDependencies": {
    ...
    "@types/jest": "^24.0.18",
    "jest": "^24.9.0"
  },
  "jest": {
    "moduleFileExtensions": ["js"]
  }
}
```

サンプルアプリケーションの実装(抜粋)



lib/processor-stack.ts

```
...
const stateMachine = new sfn.StateMachine(this, "StateMachine", {
  definition: new sfn.Pass(this, "StartState"),
});

const func = new lambda.Function(this, "LambdaFunction", {
  runtime: lambda.Runtime.NODEJS_18_X,
  handler: "handler",
  code: lambda.Code.fromAsset("./start-state-machine"),
  environment: {
    STATE_MACHINE_ARN: stateMachine.stateMachineArn,
  },
});
stateMachine.grantStartExecution(func);

const subscription = new sns_subscriptions.LambdaSubscription(func);
for (const topic of props.topics) {
  topic.addSubscription(subscription);
}
...
```

ProcessorStackの実装(抜粋)

aws-cdk-lib.assertions module

- CloudFormationテンプレートにフォーカスした、CDKアプリケーションに対するアサーションテストを書くためのモジュール
- 各テストではまず Template.fromStack() を使ってテンプレートを生成する
  - Template クラスには CloudFormationテンプレートに対してアサーションを記述するための一連のメソッドが含まれている

```
import { Template } from "aws-cdk-lib/assertions";
…
describe("ProcessorStack", () => {
  test("Synthesize ProcessorStack", () => {
  const app = new cdk.App();

  // クロススタック参照するスタックのインスタンスを作成
  const topicsStack = new TopicsStack(app, 'TopicsStack')

  // テスト対象の ProcessorStack のインスタンスを作成
  const processorStack = new ProcessorStack(app, 'ProcessorStack', {
    topics: topicsStack.topics,
  })

  // CloudFormation テンプレートを生成
  const template = Template.fromStack(processorStack);
}
```

ProcessorStack の CloudFormation テンプレートを生成するコード

Snapshot Test：テストの記述と実行

- `npm test` でテストを実行すると
  package.json の定義によってスナップショットが保存される

test/processor-stack-snapshot.test.ts

```
describe('ProcessorStack', () => {
  test('Snapshot test', () => {

    (中略)

    // テンプレートの生成
    const template = Template.fromStack(processorStack)

    // テストの実行
    expect(template.toJSON()).toMatchSnapshot()
  })
})
```

Snapshot Test の記述

Console

```
> npm test
> jest

PASS test/cdk-test-sample.test.ts
ProcessorStack
✓ synthesizes the way we expect (47 ms)

› 1 snapshot written.
Snapshot Summary
› 1 snapshot written from 1 test suite.

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 1 written, 1 total
Time: 2.237 s, estimated 3 s
Ran all test suites.
```

テストの実行

Snapshot Test：スナップショットのアップデート

- テンプレートに変更があるとテストが失敗する
- 意図した変更であることを確認してからスナップショットをアップデートする

Console

```
% npm test

Snapshot name: `ProcessorStack synthesizes the
way we expect 1`

- Snapshot - 1
+ Received + 1

@@ -166,11 +166,11 @@
    "Fn::GetAtt": [
      "LambdaFunctionServiceRoleC555A460",
      "Arn",
    ],
  },
  - "Runtime": "nodejs18.x",
  + "Runtime": "nodejs20.x",
  },  …
```

スナップショットを
アップデート

Console

```
% npm test -- -u

> cdk-test-sample@0.1.0 test
> jest -u

PASS test/shapshot-test.test.ts
ProcessorStack
✓ synthesizes the way we expect (48 ms)

› 1 snapshot updated.
Snapshot Summary
```

Fine-grained assertions Test：テストの記述

- assertionsモジュールを使ってテストを書いていく

test/processor-stack.fine-grained.test.ts

```
describe('ProcessorStack', () => {
  test('Fine-grained test', () => {

  (中略)

  // テンプレートの生成
  const template = Template.fromStack(processorStack)

  // Functionが正しいプロパティで生成されているか確認
  template.hasResourceProperties("AWS::Lambda::Function", {
    Handler: "handler",
    Runtime: "nodejs18.x",
  })

  // サブスクリプションが1つ生成されているか確認
  template.resourceCountIs("AWS::SNS::Subscription", 1);
  })
})
```

Lambda関数と Amazon SNSサブスクリプションが作成されたことを確認する例

Fine-grained assertions Test : Matcher (1/2)

- hasResourceProperties()のデフォルトの挙動は部分一致
  - 第２引数にオブジェクトを渡されると Match.objectLike() で比較される
- Matcherを使うとMatchの挙動を変えることができる
  - Lenient(Match.anyValue)
  - Strict(Match.objectEquals)

  など様々なMatcherがある

```
import { Match, Template } from "aws-cdk-lib/assertions";
…
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"],
              ],
            },
```

IAMロールを完全一致でアサートしつつ、
リージョンが変わっても対応できるようにする例

Fine-grained assertions Test : Matcher (2/2)

- CloudFormation のリソースには JSON オブジェクトが含まれていることがある
- Match.serializedJson() を使うと JSON 内のプロパティを照合できる

```
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
  // Match.objectEquals()は暗黙的に使用されるが、ここでは明示的に使用している
  Match.objectEquals({
      StartAt: "StartState",
      States: {
        StartState: {
          Type: "Pass",
          End: true,
          // Nextというフィールドがないことを確認する
          Next: Match.absent(),
        },
...
```

Step Functions ステートマシンの定義 (JSONベース)をテストする例

Fine-grained assertions Test : Capture

- Capture クラスを使用するとテスト中の値を取り出せる
- プロパティが特定の形式に従っているか、
  または他のプロパティと同じ値を持っているかをテストできる

```
import { Capture, Template } from "aws-cdk-lib/assertions";
…
const startAtCapture = new Capture()
const statesCapture = new Capture()
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    Match.objectLike({
      StartAt: startAtCapture,
      States: statesCapture,
    })
  ),
})

// 開始状態が "Start "で始まることを確認
expect(startAtCapture.asString()).toEqual(expect.stringMatching(/^Start/))

// statesオブジェクトに、開始ステートが存在することを確認
expect(statesCapture.asObject()).toHaveProperty(startAtCapture.asString())
```

← 値をキャプチャ

Capture を使ってステートマシンの状態をテストする例

Validation Test

- Construct のコンストラクタや addValidation() で
  意図しない入力で例外を送出することをテストする

lib/processor-stack.ts

```
export class ProcessorStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: ProcessorStackProps) {
    if (props.topics.length === 0) {
      throw new Error('At least one topic is required.’)
    }
    super(scope, id, props)
    …
```

Topicsが1つ以上指定されないとエラーになるConstruct

エラーになることを確かめるテスト

test/processor-stack.validation.test.ts

```
describe('ProcessorStack', () => {
  test('at least one topics must be specified', () => {
    const app = new cdk.App()
    expect(() => {
      new ProcessorStack(app, 'ProcessorStack', {
        topics: []
      })
    }).toThrowError(/At least one topic is required\./);
  })
})
```

Policy Validation : cdk-nag (1/2)

- CDK アプリケーションがルール群に準拠しているかどうかを検証する
- 定義済みルールが豊富、セキュリティとコンプライアンスのルールの自動チェックに

Console
```
% npm install cdk-nag
```
インストール

bin/cdk-test-sample.ts
```
const app = new cdk.App()
cdk.Aspects.of(app).add(new AwsSolutionsChecks({ verbose: true }))

const topicsStack = new TopicsStack(app, 'TopicsStack')
new ProcessorStack(app, 'ProcessorStack', {
  topics: topicsStack.topics,
})
```
Aspects の追加

実行

Console
```
% npx cdk synth
[Error at /TopicsStack/Topic1/Resource]
AwsSolutions-SNS2: The SNS Topic does not have
server-side encryption enabled. Server side
encryption adds additional protection of sensitive
data delivered as messages to subscribers.

...
Found errors
```

Policy Validation : cdk-nag (2/2)

- エラーに合わせて修正するか、理由があって修正できないものは抑制する

エラーメッセージ

```
[Error at /TopicsStack/Topic1/Resource]
AwsSolutions-SNS2: The SNS Topic does not have
server-side encryption enabled. Server side
encryption adds additional protection of sensitive
data delivered as messages to subscribers.
```

修正して対処

```
new sns.Topic(this, 'Topic1', {
  kmsMasterKeyId: snsKey.keyArn
})
```

理由を記入して Supress

```
import { NagSuppressions } from 'cdk-nag'

const topic = new sns.Topic(this, 'Topic1')

NagSuppressions.addResourceSuppressions(topic, [
  {id: 'AwsSolutions-SNS2', reason:'Encryption is not needed for topics
which is used for triggering state machine.'}
])
```

Policy Validation : CloudFormation Guard

- cdk-validator-cfnguard を使ってCDKと連携可能 (2023年12月現在experimental)
- 開発や運用までいずれの工程でも
  システム環境がポリシーに適合していることをコードで検証できる
- S3 Bucket がバージョン管理を有効にしていなければならないといった単純な検証や、
  サブネットに配置された Redshift クラスタのパブリックネットワークへの到達性を
  防ぐといった複雑な検証を組み合わせて表現することができる

Console

```
// Install (CDK CFN Guard Validator Plugin)
% npm install @cdklabs/cdk-validator-cfnguard
```

インストール

追加後 synth を実行

bin/cdk-test-sample.ts

```
import { CfnGuardValidator } from "@cdklabs/cdk-validator-cfnguard"

const app = new cdk.App({
  policyValidationBeta1: [new CfnGuardValidator()],
})
```

Integration Test

- integ-runner CLI と組み合わせて統合テストを実行する
  - cdk synth を実行して Snapshot Test をする
  - cdk deploy を実行して有効な CloudFormation テンプレートであることを確かめる
    - デプロイされたリソースはテスト後に削除される
  - デプロイされたリソースに対して追加のテストを記述することもできる

Console

```
% npm install @aws-cdk/integ-runner
% npm install @aws-cdk/integ-tests-alpha
```

必要なモジュールのインストール

test/integ.processor-stack.test.ts

```
const app = new cdk.App();
const topicsStack = new TopicsStack(app, 'TopicsStack')

new IntegTest(app, 'ProcessorStackIntegTest', {
  testCases: [
    new ProcessorStack(app, 'ProcessorStack', {
      topics: topicsStack.topics,
    })
  ],
})
```

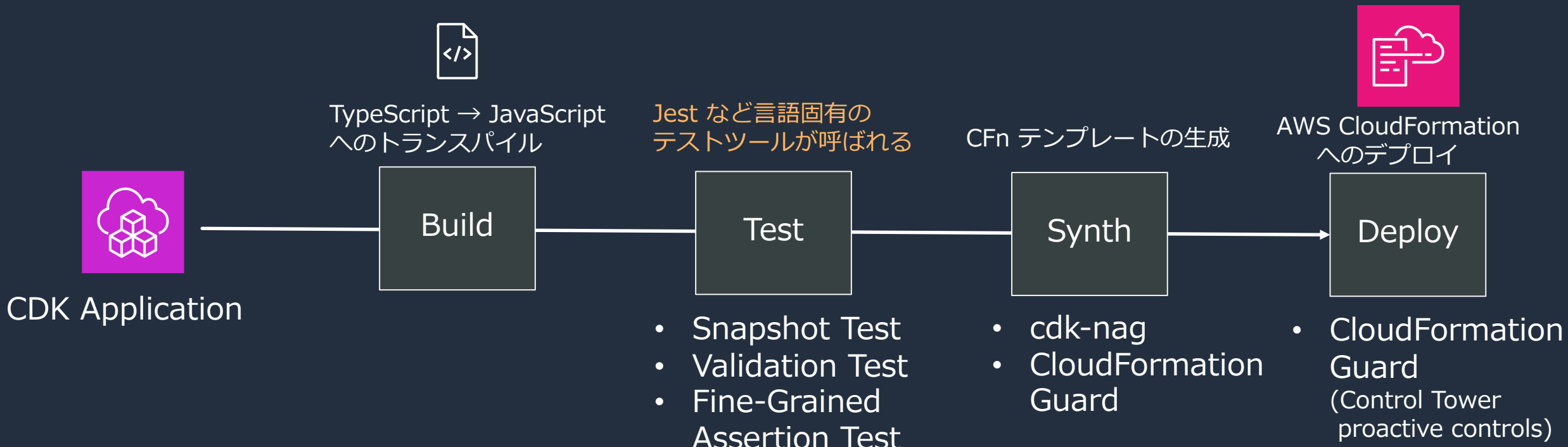
テストの記述

Console

```
% npx integ-runner
```

テストの実行

テストが実行されるタイミング



他のタイミングで呼ばれるもの
- Integration Test (integ-runner 実行時)
- Linter・Formatter など

テストの使い分けの例

- CDK アプリケーション開発
  - Snapshot Test
    - 導入が簡単で、意図しないデグレを防ぐために有用
  - Fine-Grained Assertion Test
    - CDKアプリのコードをシンプルに保っていると自明なテストになりがち
      必要なところに必要なだけ
  - Policy Validation
    - あらかじめ用意されたポリシーを使って手軽にチェックを始めたい → cdk-nag
    - 独自のポリシーを作成したい、組織全体のガバナンスを取りたい → cfn-guard
- CDK Construct開発
  - Validation Test、Integration Test などを上記に加えて導入

開発にまつわる Tips

汎用的なセキュリティチェックツール

- 様々なツールを使用することでデプロイ前に脆弱性をチェックできる
- 例
  - git-secrets コミット前
    - https://github.com/awslabs/git-secrets
    - パスワードやその他の機密情報を g コミット前、CI コミットすることを防ぐ
  - コードレビューツール
    - Semgrep(TypeScript)、Bandit(Python)、Amazon CodeGuru など
  - コンテナを使用している場合は Trivy、 Amazon Inspector など

Linter と Formatter

- Linter
  - コードに問題点がないかを確認する静的解析ツール
  - ESLint (TypeScript/JavaScript) など
- Formatter
  - コードのスタイルをチェックするツール
  - 書き方が統一されるので、多くの人が読みやすい書き方でコードを書くことできる
  - Prettier (TypeScript/JavaScript) など

```
constructor(scope: Construct, id: string, props?: cdk.StackProps) {
  super(scope, id, props);
  this.topics = [
    new sns.Topic(this, 'Topic1', {});
  ]
}
```

Formatterを実行すると
改行やセミコロンの有無などが統一される

```
constructor(scope: Construct, id: string, props?: cdk.StackProps) {
  super(scope, id, props)
  this.topics = [new sns.Topic(this, 'Topic1', {})]
}
```

projen

- https://github.com/projen/projen
- package.json、tsconfig.json、.gitignore、GitHub Workflows、eslint、jest などのようなプロジェクト設定ファイルを生成するツール

Console

```
$ mkdir my-project
$ cd my-project
$ npx projen new awscdk-app-ts
🤖 Synthesizing project...
...
```

ファイルが生成される

Console

```
$ ls
LICENSE  README.md  cdk.json  node_modules  package.json  src
test  tsconfig.dev.json  tsconfig.json  yarn.lock
```

よく使用するドキュメント

- AWS CDK Reference Documentation
  https://docs.aws.amazon.com/cdk/api/v2/
  - CDKを利用する際に一番基本となるドキュメント
- AWS CloudFormation User Guide
  https://docs.aws.amazon.com/en_us/cloudformation/
  - L1 Construct を利用する際のお供に
- AWS CDK GitHub Issues
  https://github.com/aws/aws-cdk/issues
  - 不具合報告や既出の質問がないか調べる
- AWS CDK RFCs (RFC = Request for Comments)
  https://github.com/aws/aws-cdk-rfcs
  - AWS CDK や jsii などの関連プロジェクトに対する主要な変更について学ぶことができる

AWS Black Belt Online Seminar とは

- 「サービス別」「ソリューション別」「業種別」などのテーマに分け、アマゾン ウェブ サービス ジャパン合同会社が提供するオンラインセミナーシリーズです
- AWS の技術担当者が、 AWS の各サービスやソリューションについてテーマごとに動画を公開します
- 以下の URL より、過去のセミナー含めた資料などをダウンロードすることができます
  - https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/
  - https://www.youtube.com/playlist?list=PLzWGOASvSx6FIwIC2X1nObr1KcMCBBlqY

ご感想は X (Twitter) へ！ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では資料作成時点のサービス内容および価格についてご説明しています。AWS のサービスは常にアップデートを続けているため、最新の情報は AWS 公式ウェブサイト (https://aws.amazon.com/) にてご確認ください
- 資料作成には十分注意しておりますが、資料内の価格と AWS 公式ウェブサイト記載の価格に相違があった場合、AWS 公式ウェブサイトの価格を優先とさせていただきます
- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます
- 技術的な内容に関しましては、有料の AWS サポート窓口へお問い合わせください
- 料金面でのお問い合わせに関しましては、カスタマーサポート窓口へお問い合わせください (マネジメントコンソールへのログインが必要です)

Thank you!