aws DEV DAY
ONLINE JAPAN

20-22.10.2020

DEV DAY

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

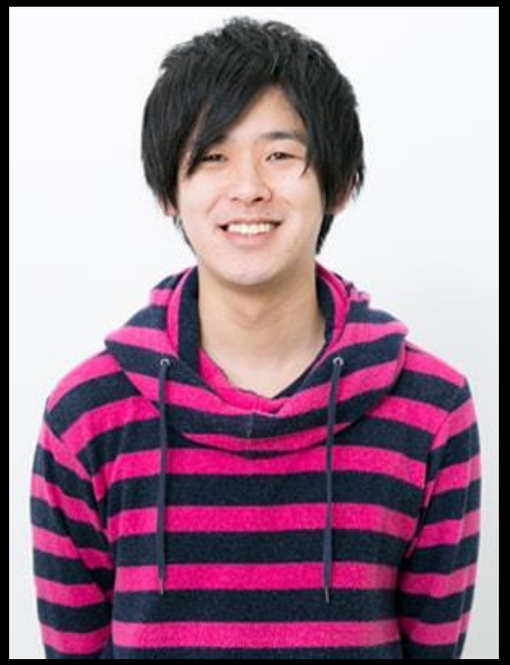
F - 9

CDK
我流ベストプラクティスの紹介

放地 宏佳 @hixi-hyi

In Partnership with intel

自己紹介



放地宏佳 @hixi-hyi

2013年株式会社ディー・エヌ・エー新卒入社
オープンプラットフォームのサーバサイドエンジニア
いくつかの協業案件にてアプリ・クラウドアーキテクトを経験
現在社内横断エンジニア部門に所属

副業制度を利用し、技術コンサル・アーキテクト・インフラなどの副業活動中。

Cloudformation を利用していたが、 CDK のリリースを機に CDK に移行

本セッションで話す内容

- 自分がやってみてよかったと思う構成の話をします
- 中級者セッションとしてますので、用語の詳細について説明を省きます
- セッション内容を適用したサンプルについては最後にリンクを張ります

CDK 我流ベストプラクティス

- id の構造化
- ディレクトリ構成と役割・責務

aws DEV DAY
ONLINE JAPAN

id の構造化

In Partnership with intel

cdk init で作られるサンプルコード

```
cdk init app --language=typescript aws-cdk-bestpractice
```


In Partnership with intel

cdk init で作られるサンプルコード

```
cdk init app --language=typescript aws-cdk-bestpractice
```

bin/aws-cdk-bestpractice.ts

```
const app = new cdk.App();
new AwsCdkBestpracticeStack(app, 'AwsCdkBestpracticeStack’);
```

cdk init で作られるサンプルコード

```
cdk init app --language=typescript aws-cdk-bestpractice
```

bin/aws-cdk-bestpractice.ts

```
const app = new cdk.App();
new AwsCdkBestpracticeStack(app, 'AwsCdkBestpracticeStack’);
```

lib/aws-cdk-bestpractice.ts

```
export class AwsCdkBestpracticeStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, …) {
    super(scope, id, props);
  }
}
```


In Partnership with intel

cdk init で作られるサンプルコード

```
cdk init app --language=typescript aws-cdk-bestpractice
```

bin/aws-cdk-bestpractice.ts

```
const app = new cdk.App();
new AwsCdkBestpracticeStack(app, 'AwsCdkBestpracticeStack’);
```

lib/aws-cdk-bestpractice.ts

```
export class AwsCdkBestpracticeStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, …) {
    super(scope, id, props);
  }
}
```


In Partnership with intel

やりたいこと

- 開発と本番をほぼ共通なリソースセットを定義したい
  ただし開発と本番でパラメータを使いたい部分がある

- リソース定義の際、ある程度 human readable な名前をつけたい
  リソースに渡す文字列を決め打ちとしてしまうと、 construct の再利用性が損なわれる
  毎回毎回文字列を外部から渡せるようにするのは煩雑
  リソースによっては文字制約が存在するが、それを使うたびに判断するのは簡単ではない

id の構造化 - 定義

https://github.com/hixi-hyi/aws-cdk-identifier

```
const id = new cdkid.Identifier({
  section: resolveEnvironmentFromAwsAccountId(),
});

const app = new cdk.App();
id.child({legion: "product"}).scope((id: cdkid.Identifier) => {
  id.child({cohort: "service"}).scope((id: cdkid.Identifier) => {
    new stack.ProductServiceApi(app, id.child({family: 'api'}));
  });
});


export class ProductServiceApi extends cdk.Stack {
  constructor(app: cdk.App, id: cdkid.Identifier, props: ProductServiceApiProps) {
  ...
  }
}
```

id の構造化 - パラメータ設定

https://github.com/hixi-hyi/aws-cdk-identifier

```
export class ProductServiceApi extends cdk.Stack {
  constructor(app: cdk.App, id: cdkid.Identifier, props: ProductServiceApiProps) {
    super(app, id.stackName, props);
    const fargate = new construct.Fargate(this, id.child("Aurora"), {
      maxCapacity: cdkid.Variables.resolve(id, cdkid.RankLoc.Section, {
        dev: 1,
        prod: 10,
      }),
    });
}
```

パラメータ設定を柔軟に

id の構造化 - 名前付け

https://github.com/hixi-hyi/aws-cdk-identifier

```
export class Fargate extends cdk.Construct {
  constructor(app: cdk.Construct, id: cdkid.Identifier, props: FargateProps) {
    super(app, id.constructName);
    const cluster = new ecs.Cluster(this, 'Cluster', {
      clusterName: id.camelName,
      ...
    });
    ...
}
```

Construct の再利用性を高める

id の構造化 - +α システムとの連携

https://github.com/hixi-hyi/aws-cdk-identifier

```
const patterns = new ecsPatterns.ApplicationLoadBalancedFargateService(this, “Service”,
{
  ...
  taskImageOptions: {
    environment: {
      ‘IAC_CORE_SECTION’: id.rank.section, // dev or prod
    },
  },
});
```

開発側にも環境変数で渡している

id の構造化

- id を String のままにしておくのはもったいない
- 第二引数では構造化された id が渡るのを共通ルールとして利用しましょう

aws DEV DAY
ONLINE JAPAN

ディレクトリ構成と役割・責務

In Partnership with intel

ディレクトリを分ける意味

- 責務の明確化を行い保守性を高める
- 変数スコープを用いて依存を明確化する

ディレクトリを分ける意味

- 責務の明確化を行い保守性を高める
- 変数スコープを用いて依存を明確化する

プログラミング言語だからこそ

- やらなければいけない
- できて嬉しい

ディレクトリ構成

```
├── bin
│   └── infra.ts
└── lib
    ├── @aws-cdk/
    ├── construct/
    ├── helper/
    └── stack/
```

ディレクトリ構成

```
├── bin
│   └── infra.ts （lib/stack を呼び出して stack の依存を定義する）
└── lib
    ├── @aws-cdk （cdk に変更を加えるときに利用する）
    ├── construct （自プロダクト用 L2/patterns construct を定義する）
    ├── helper （プロダクトや環境毎に異なる変数の定義する）
    └── stack （lib/construct を呼び出してリソースの依存を定義する）
```

aws DEV DAY
ONLINE JAPAN

bin/infra.ts の役割・責務

In Partnership with intel

bin/infra.ts の役割・責務

- スタック間の依存の定義

  具体的なパラメータの定義はしない

- StackName の論理的階層構造の定義

  自分の場合以下のようなルールを作っています

  - 実サービスの動作に影響があるものは Service 以下にまとめる
  - 実サービスの動作に影響がないものは Support 以下にまとめる

```
id = cdkid.Identifier({ section: resolveEnvironmentFromAwsAccountId() });
const common = new stack.ProductCommon(app, id);
new stack.ProductServiceApi(app, id.child({cohort: “service”, family: 'api'}), { common });
```

aws DEV DAY
ONLINE JAPAN

lib/stack/* の役割・責務

In Partnership with intel

lib/stack/* の役割・責務

- lib/construct を用いてプロダクトのスタックを定義する
- idの構造化で述べたパラメータの解決は基本的にここで行う

```
export class ProductServiceApi extends cdk.Stack {
  constructor(app: cdk.App, id: cdkid.Identifier, props: ProductServiceApiProps) {
    super(app, id.stackName, props);

}
```

lib/stack/* の役割・責務

- lib/construct を用いてプロダクトのスタックを定義する
- idの構造化で述べたパラメータの解決は基本的にここで行う

```
export class ProductServiceApi extends cdk.Stack {
  constructor(app: cdk.App, id: cdkid.Identifier, props: ProductServiceApiProps) {
    super(app, id.stackName, props);
    const ecr = new construct.Ecr(this, id.child('Repository'));

}
```

lib/stack/* の役割・責務

- lib/construct を用いてプロダクトのスタックを定義する
- idの構造化で述べたパラメータの解決は基本的にここで行う

```
export class ProductServiceApi extends cdk.Stack {
  constructor(app: cdk.App, id: cdkid.Identifier, props: ProductServiceApiProps) {
    super(app, id.stackName, props);
    const ecr = new construct.Ecr(this, id.child('Repository'));
    const fargate = new construct.FargateL7Webapp(this, id.child('Webapp'), {
      ...
      image: ecs.ContainerImage.fromEcrRepository(ecr.repository),
      maxCapacity: cdkid.Variables.resolve(id, cdkid.RankLoc.Section, {
        dev : 1,
        prod: 10,
      }),
    });
    ...
}
```

lib/stack/* の役割・責務

- lib/construct を用いてプロダクトのスタックを定義する
- idの構造化で述べたパラメータの解決は基本的にここで行う

```
export class ProductServiceApi extends cdk.Stack {
  constructor(app: cdk.App, id: cdkid.Identifier, props: ProductServiceApiProps) {
    super(app, id.stackName, props);
    const ecr = new construct.Ecr(this, id.child('Repository'));
    const fargate = new construct.FargateL7Webapp(this, id.child('Webapp'), {
      ...
      image: ecs.ContainerImage.fromEcrRepository(ecr.repository),
      maxCapacity: cdkid.Variables.resolve(id, cdkid.RankLoc.Section, {
        dev : 1,
        prod: 10,
      }),
    });
    ...
}
```

aws DEV DAY
ONLINE JAPAN

lib/construct/* の役割・責務

In Partnership with intel

lib/construct/* の役割・責務

- プロダクトで汎用的に使える L2/patterns construct の定義を行う

  aws-cdk で用意されている L2/patterns はあくまで汎用的なもの

  プロダクトで使うには更にまとめたものが必要になってくる

  - fargate を用いた webapp は ecs-patterns だけでは構築できない
  - ecs.Cluster, logs.LogGroup, ecr.Repository などの定義が必要

  プロダクトで使うには共通の設定が一定必要になる

  - セキュリティ・監査のために KMS を確実に有効化する
  - 標準的な監視を On にする, 等々

- Stack 毎に L2/patterns を使うと再利用性を損なう・構築漏れが発生

aws DEV DAY
ONLINE JAPAN

lib/@aws-cdk/* の役割・責務

In Partnership with intel

lib/@aws-cdk/* の役割・責務

- L2/patterns construct の patch をあてたものを定義する

  cdk では construct で使いたい機能が提供されていないものがある

  これら機能が提供されるまでプロダクト構築が止まるのは論外

  参考: https://github.com/aws/aws-cdk/issues/9926 https://github.com/aws/aws-cdk/pull/10324

  - 8/24: issued
  - 9/13: PR open
  - 9/25: PR merged
  - 9/30: cdk release

aws DEV DAY
ONLINE JAPAN

lib/helper/* の役割・責務

In Partnership with intel

lib/helper/* の責務

- [EXPERIMENTAL] 自分でも試行錯誤中

  プロダクトや環境、状況によって異なる変数の定義

  construct に特有の変数を持ち込まないための処置

    RemovalPolicy, LogRetentionDays, S3Bucket の suffix などを定義している

    初期構築時にはすべての RemovalPolicy を Delete にして反復実行できるようにしています

  construct の再利用性を高めるためにこのようなレイヤーを作ることが好ましいと考えています

```
export function LogGroupRetentionDay(id: cdkid.Identifier): logs.RetentionDays {
  return cdkid.Variables.resolve<logs.RetentionDays>(id, cdkid.RankLoc.Section, {
    dev: logs.RetentionDays.TWO_WEEKS,
    prod: logs.RetentionDays.INFINITE,
  })
}
```

ディレクトリ構成と役割・責務

```
├── bin
│   └── infra.ts （lib/stack を呼び出して stack の依存を定義する）
└── lib
    ├── @aws-cdk （cdk に変更を加えるときに利用する）
    ├── construct （自プロダクト用 L2/patterns construct を定義する）
    ├── helper （プロダクトや環境毎に異なる変数の定義する）
    └── stack （lib/construct を呼び出してリソースの依存を定義する）
```

aws DEV DAY
ONLINE JAPAN

まとめ

In Partnership with intel

まとめ

- id の構造化
- ディレクトリ構成と役割・責務

本日の構成を用いて作ったサンプルは以下から見ることができます。
https://github.com/hixi-hyi/aws-cdk-bestpractice

aws DEV DAY
ONLINE JAPAN

CDK よくありそうな疑問

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

Q1. L1 を使うべき？L2 を使うべき？

In Partnership with intel

Q1. L1 を使うべき？ L2 を使うべき？

L2 を使いましょう

In Partnership with intel

Q1. L1 を使うべき？ L2 を使うべき？

多くのエンジニアの集合知に乗れるため積極的に採用すべき
軽微の未実装の場合は lib/@aws-cdk で補完しつつ PR をしていく

aws DEV DAY
ONLINE JAPAN

Q2. テストを書いてますか？

In Partnership with intel

Q2. テストを書いていますか？

書いてません

Q1. テストを書いていますか？書いてませんか？

今の所必要性がない
cdk diff だけで十分テストになっている

aws DEV DAY
ONLINE JAPAN

Q3. IaC で気をつけていることは？

In Partnership with intel

Q3. IaC で気をつけていることは？

開発者の手を止めないための手段を考える

アプリとインフラの境界線を常に意識する

- 開発環境では開発者が自由にいじってもいい IAM Policy を用意しておく
- SSM Parameters の /service/xxx/app/* 以下は開発者が自由に触る場所
- インフラと開発をつなぐのは Dockerfile のみ

aws DEV DAY
ONLINE JAPAN

Thank you!

CDK我流ベストプラクティスの紹介
- id の構造化
- ディレクトリ構成と役割・責務

放地 宏佳

https://github.com/hixi-hyi/aws-cdk-bestpractice

In Partnership with intel