TOKYO

DEV DAY

2019.10.03-04

DEV DAY
TOKYO

2019.10.03-04

B - 3

ソフトウェア開発者のための AWS環境構築フレームワーク AWS Cloud Development Kit (CDK)

大村 幸敬
ソリューションアーキテクト
アマゾンウエブサービスジャパン

aws

大村 幸敬 (おおむら ゆきたか)

ソリューションアーキテクト

- これからクラウドを使いはじめる
  エンタープライズ企業をサポート
- Management Tools & DevOps 系サービス

好きなAWSのサービス：**AWS CLI, AWS CDK**

Agenda

1. AWSの環境を構築するには
2. AWS CDK (Cloud Development Kit) の紹介
3. How to use AWS CDK
4. Demo
5. CDK コンセプト
6. DiveDeep
7. 情報源

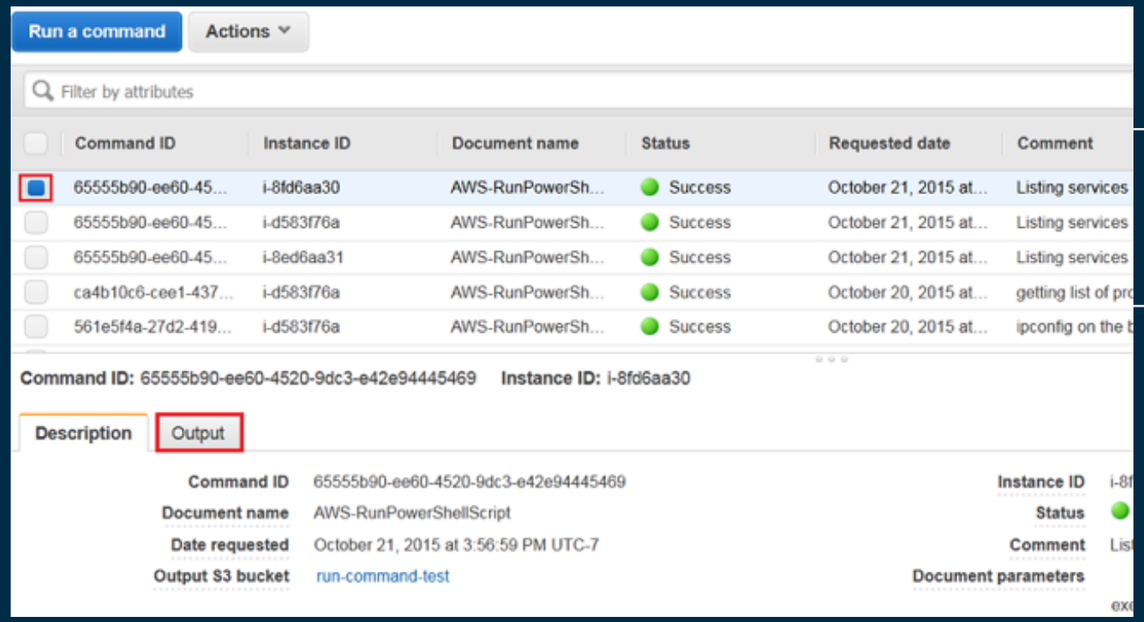
※このセッションは TypeScript メインで話しますが、
Pythonの解説を見たい方は AWS Innovate（オンラインカンファレンス）をご覧ください

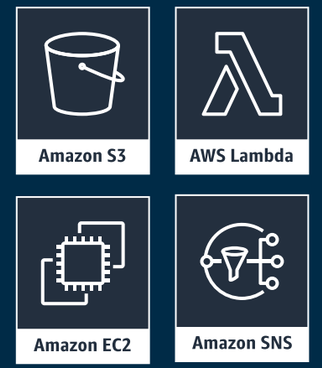
DEV DAY

AWS の環境を構築するには

手動操作（マネジメントコンソール）

操作手順書が別途必要



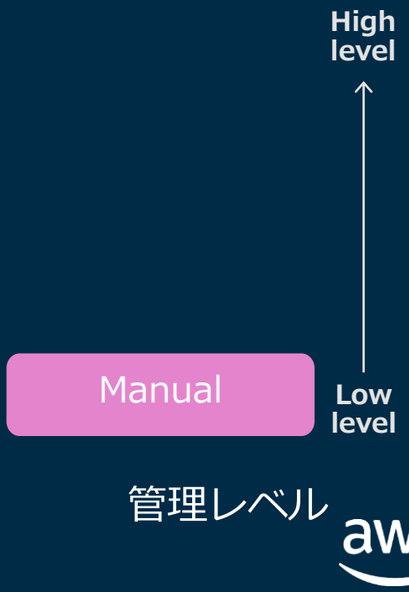


👍始めるのは簡単

🤔繰り返し可能ではない

🤔エラーが起きやすい

🤔時間がかかる

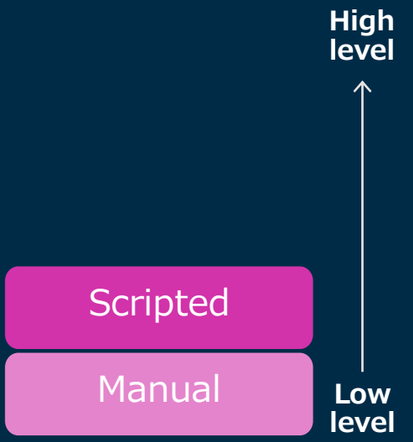


スクリプト (SDK, CLI)

操作手順の定義が可能

```ruby
require 'aws-sdk-ec2'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

instance = ec2.create_instances({
  image_id: 'IMAGE_ID',
  min_count: 1,
  max_count: 1,
  key_name: 'MyGroovyKeyPair',
  security_group_ids: ['SECURITY_GROUP_ID'],
  instance_type: 't2.micro',
  placement: {
    availability_zone: 'us-west-2a'
  },
  subnet_id: 'SUBNET_ID',
  iam_instance_profile: {
    arn: 'arn:aws:iam::' + 'ACCOUNT_ID' + ':instance-profile/aws-opsworks-ec2-role'
  }
})
```

- 🤔APIコールが失敗したら何が起こる？
- 🤔どうやってアップデートする？
- 🤔リソースが準備完了なのはどうやって知る？
- 🤔どうやってロールバックする？



プロビジョニング ツール (CloudFormationなど)

あるべき状態の定義が可能

👍自動化が容易
👍再生成可能
😩ツール固有の記述方式
😩抽象化なし、詳細な記述

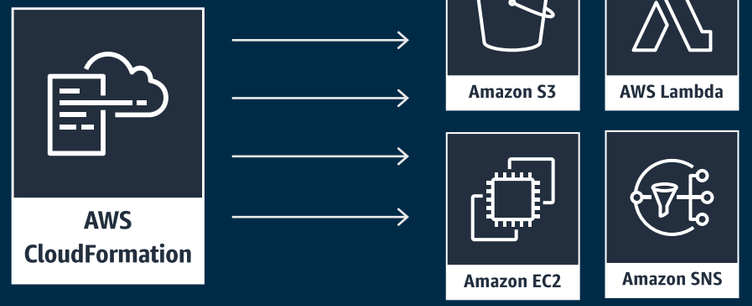
Document Object Models (DOMs)

あるべき状態の定義がコードで可能

```python
from troposphere import Template
from troposphere.ec2 import VPC, Subnet, InternetGateway

t = Template()

VPC = t.add_resource(
    VPC(
        'VPC',
        CidrBlock='10.0.0.0/16',
        Tags=Tags(
            Application=ref_stack_id)))

subnet = t.add_resource(
    Subnet(
        'Subnet',
        CidrBlock='10.0.0.0/24',
        VpcId=Ref(VPC),
        Tags=Tags(
            Application=ref_stack_id)))
```

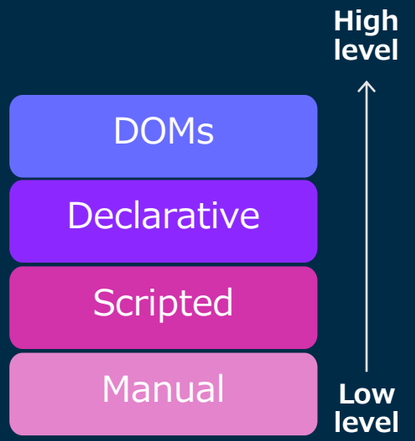
Troposphere *Python*
SparkleFormation *Ruby*
GoFormation *Go*
…

CFn
テンプレート
の生成



CFnで
デプロイ

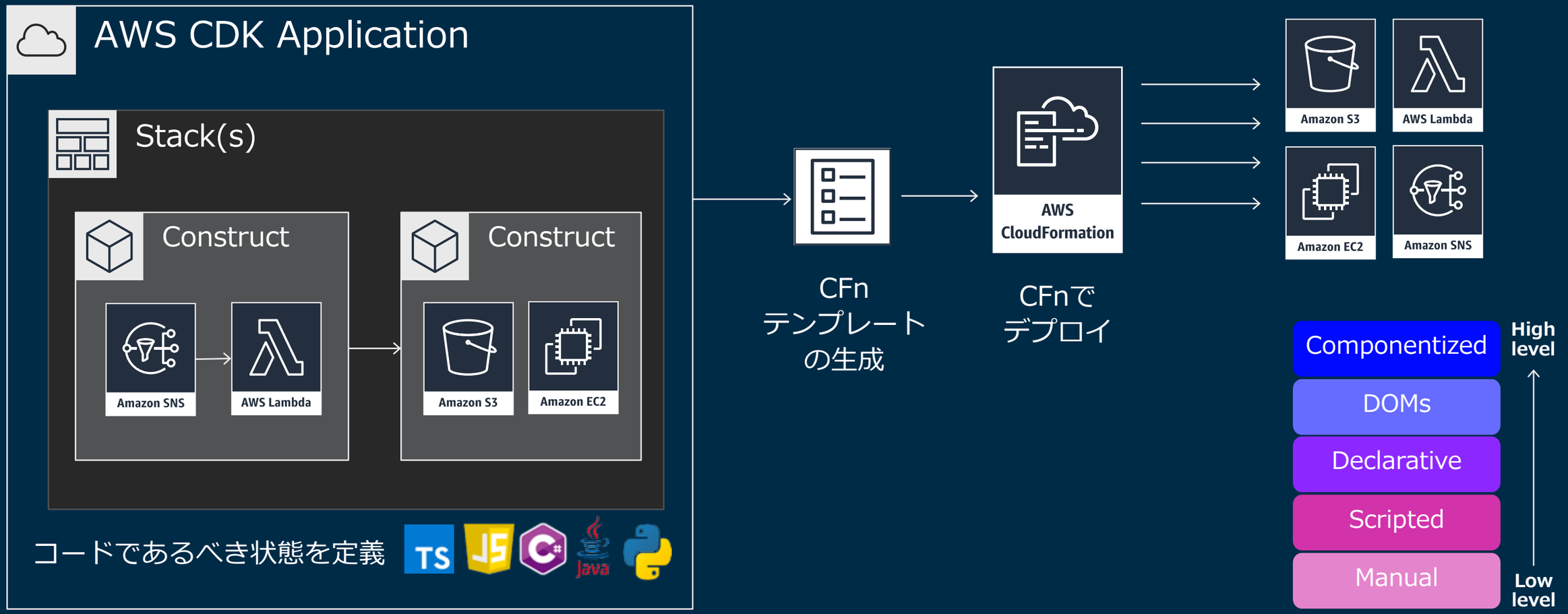
👍リアルコード ♥ *if*文、*for* ループ、*IDE*利用可能
👍あるべき状態の定義 例えば、*Troposphere*で*VPC*を作成するには*128*行必要
😢抽象化は組み込まれていない



管理レベル

AWS CDK

あるべき状態の定義がコードで可能+抽象化



DEV DAY

AWS CDK (Cloud Development Kit)

AWS CDK

AWSの環境を 一般のプログラミング言語で記述できるツールキット

- ソースコードからCloudFormationテンプレートを生成
- AWSのベストプラクティスが定義されたライブラリ (Construct) によって少ないコードで記述可能
- オープンソースで開発されておりユーザが拡張可能
- 2018年8月 から Developer Preview
  2019年7月11日 一般利用可能（Generally Available）

AWS CDK 環境要件

CDK
- Node.js (>= 8.11.x)

開発言語
- TypeScript >= 2. 7
- Python >= 3.6
- Java 8 and Maven 3.5.4
- C# (.NET Core >= 2.0,
  .NET Framework >= 4.6.1,
  Mono >= 5.4)

一般利用可能:
TypeScript, Python

Developer Preview:
Java, C# (.NET)

aws

参考: jsii

JavaScript で書かれたクラスに 他の言語からアクセスするためのライブラリ

- https://github.com/aws/jsii
- "Use javascript modules naturally from every programming language"

- AWSが CDK 実装のために開発しオープンソース化 (Apache License 2.0)
- CDK は TypeScript で実装され、jsii によって
  Python など他の言語でもタイムリーに反映されるようにしている
- jsii は開発やビルドツールでの使用を想定しておりパフォーマンスは限定的

CDKの特徴（CloudFormation に比較して）

CDKのメリット

- 一般のプログラミング言語が使える
  - 制御構文のほか、クラスや継承など抽象化の概念が利用できる
  - エディタによる型チェック、サジェスト、API仕様の参照が可能
- 一般にコード量が少なくなる
- テストコードが記述できる
- 複数スタック間の依存関係が記述できる
- バックエンドがCFnである（豊富な実績とツールの安定性）

CDKから見た CloudFormation (CFn) の位置付け

- CDKを使った場合も最終的なデプロイはCFnで行われる
- 詳細な設定の記述には CFn と同様の記述が必要
- 従来の資産やSAM(Serverless Application Model) の活用はCFn

aws

DEV DAY

How to use AWS CDK

CDK 実行環境の準備

1. 事前に必要な環境
   - AWS CLI
   - Node.js 8.0 以上の環境を用意
     - Windows: MSIインストーラ / Mac: homebrew & nodebrew
2. cdkの導入
   - "npm install -g aws-cdk"
3. 各言語の開発環境を用意（例: TypeScript）
   - お好きなエディタ （例：VisualStudio Code）
4. 初期コードを用意
   - "cdk init app --language=typescript"
   - サンプルコード付き → "cdk init sample-app --language=typescript"
5. Let's coding!

CDK コードのデプロイ

1. CDKデプロイ管理用の環境（S3バケット）を作成（初回のみ）
   - "cdk bootstrap"

2. ビルドとCFnテンプレートの生成
   - "npm run build"
   - "cdk synth mystack"

3. デプロイ
   - "cdk deploy mystack"

DEV DAY

Demo

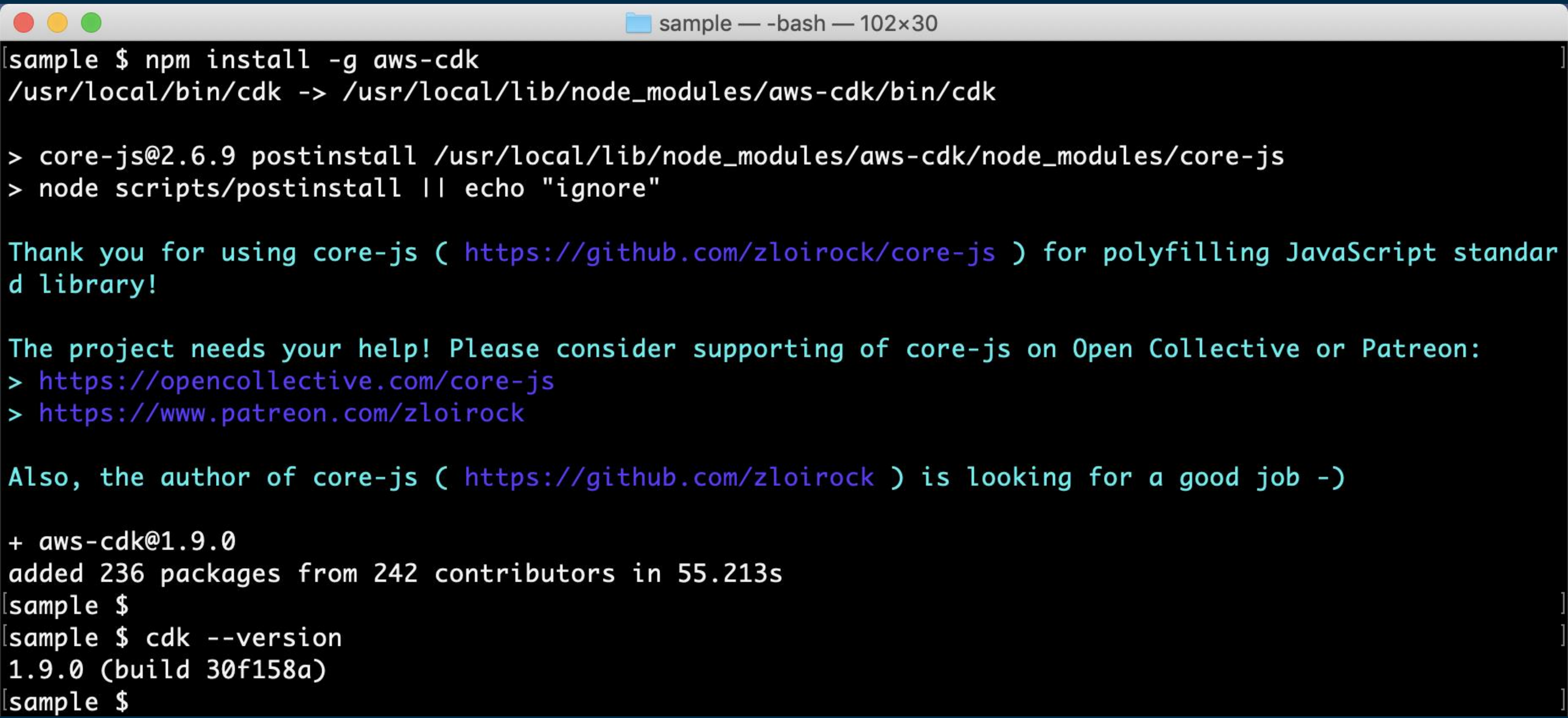
デモの流れ

1. 環境の準備
   - Node.js 導入済み
   1. cdk インストール
   2. cdk bootstrap
   3. cdk init
   4. コードの確認

2. 最初のデプロイ
   1. npm build
   2. cdk synth
   3. cdk deploy

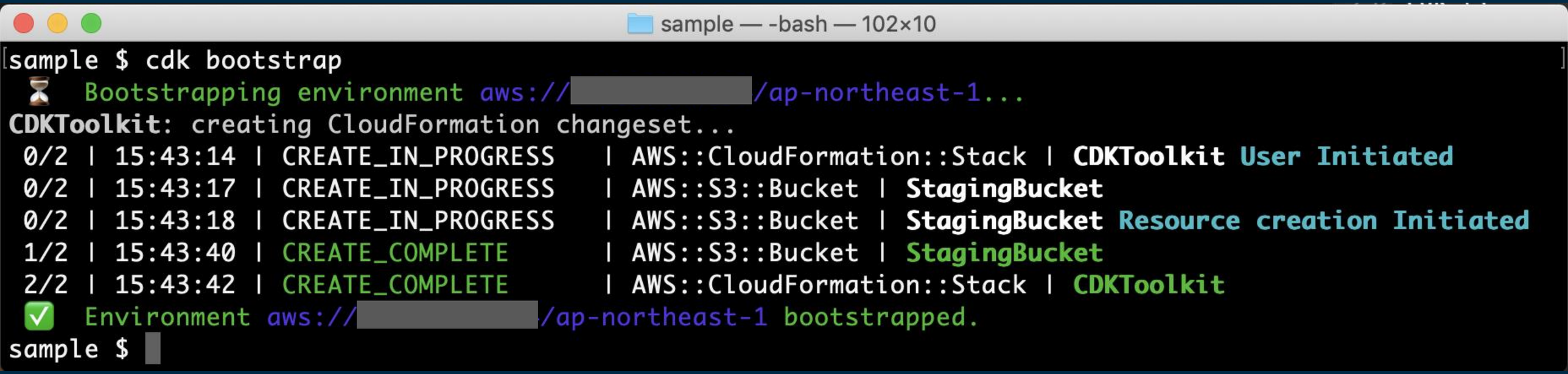
3. コード変更とデプロイ
   1. テストの実施(OK)
   2. テストコード変更(NG)
   3. コード修正と再テスト(OK)
   4. cdk diff
   5. cdk deploy

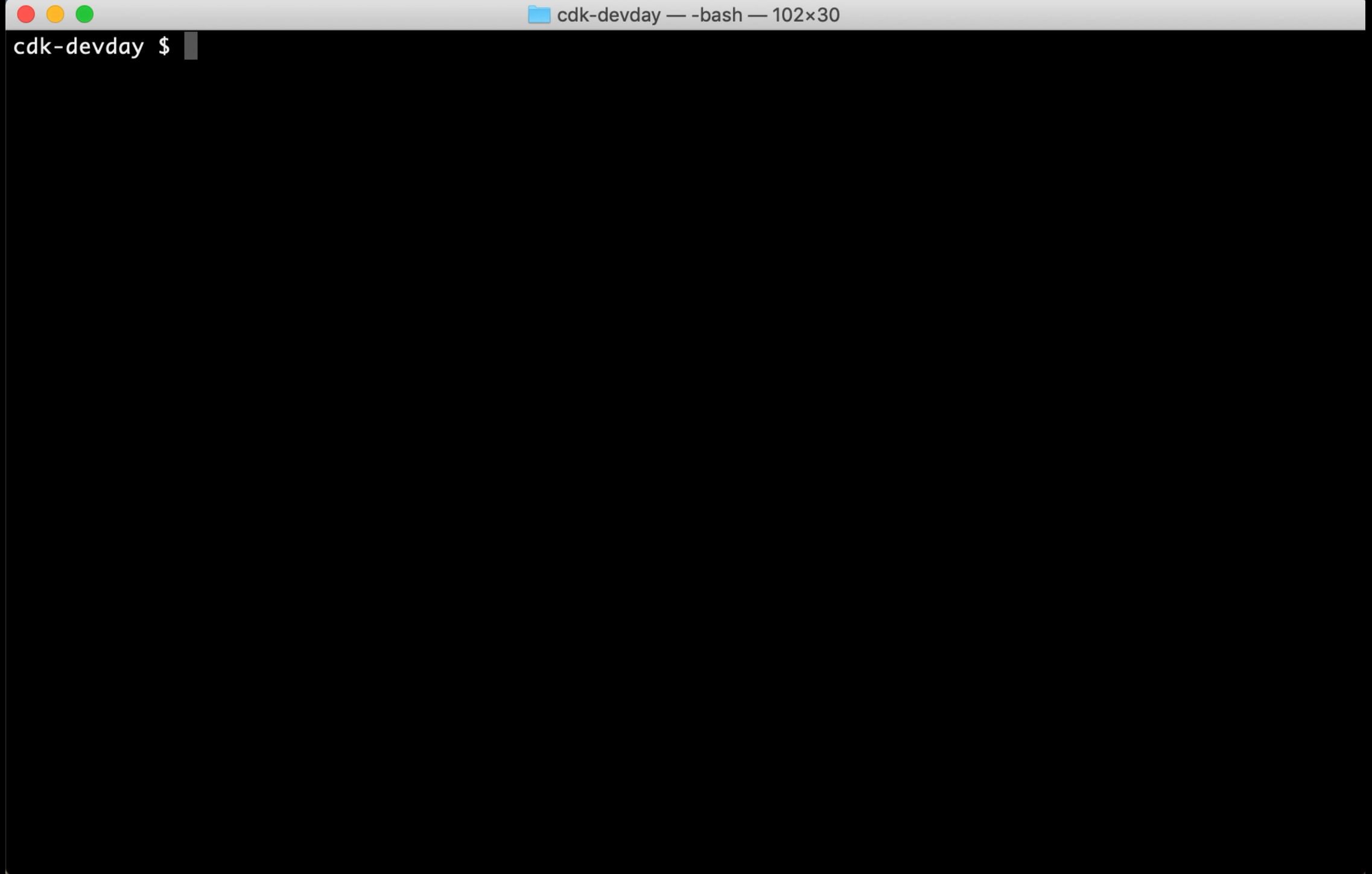
CDKインストール


sample — -bash — 102×30
sample $ npm install -g aws-cdk
/usr/local/bin/cdk -> /usr/local/lib/node_modules/aws-cdk/bin/cdk
> core-js@2.6.9 postinstall /usr/local/lib/node_modules/aws-cdk/node_modules/core-js
> node scripts/postinstall || echo "ignore"
Thank you for using core-js (https://github.com/zloirock/core-js) for polyfilling JavaScript standar
d library!
The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock
Also, the author of core-js (https://github.com/zloirock) is looking for a good job -)
+ aws-cdk@1.9.0
added 236 packages from 242 contributors in 55.213s
sample $
sample $ cdk --version
1.9.0 (build 30f158a)
sample $

cdk bootstrap

- 対象アカウント、リージョンにつき1回だけ実施
- CDKがデプロイのために使用するS3バケットを作成



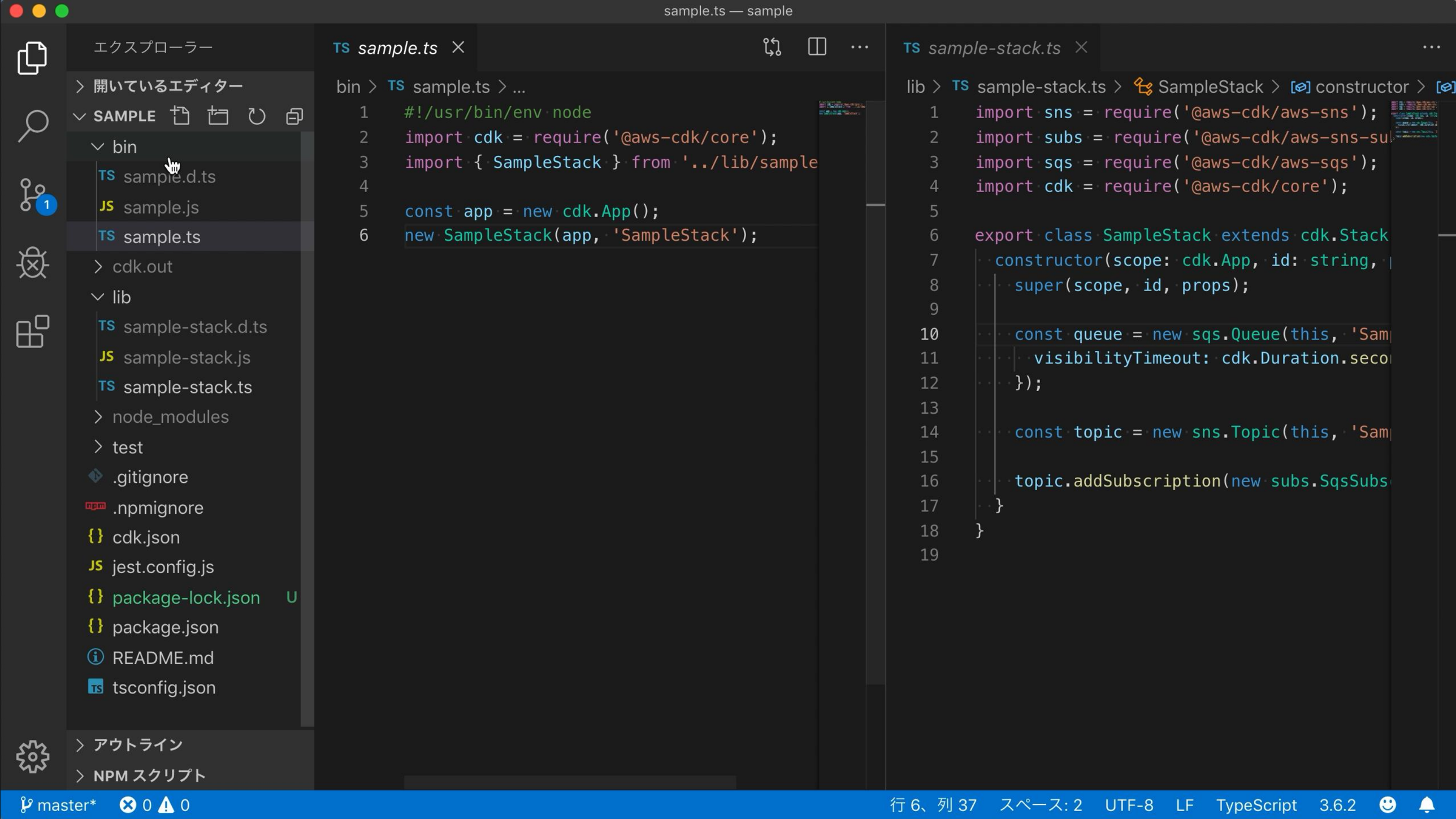

cdk-devday — -bash — 102×30
cdk-devday $

デモの流れ

1. 環境の準備
   - Node.js 導入済み
   1. cdk インストール
   2. cdk bootstrap
   3. cdk init
   4. コードの確認

2. 最初のデプロイ
   1. npm build
   2. cdk synth
   3. cdk deploy

3. コード変更とデプロイ
   1. テストの実施(OK)
   2. テストコード変更(NG)
   3. コード修正と再テスト(OK)
   4. cdk diff
   5. cdk deploy

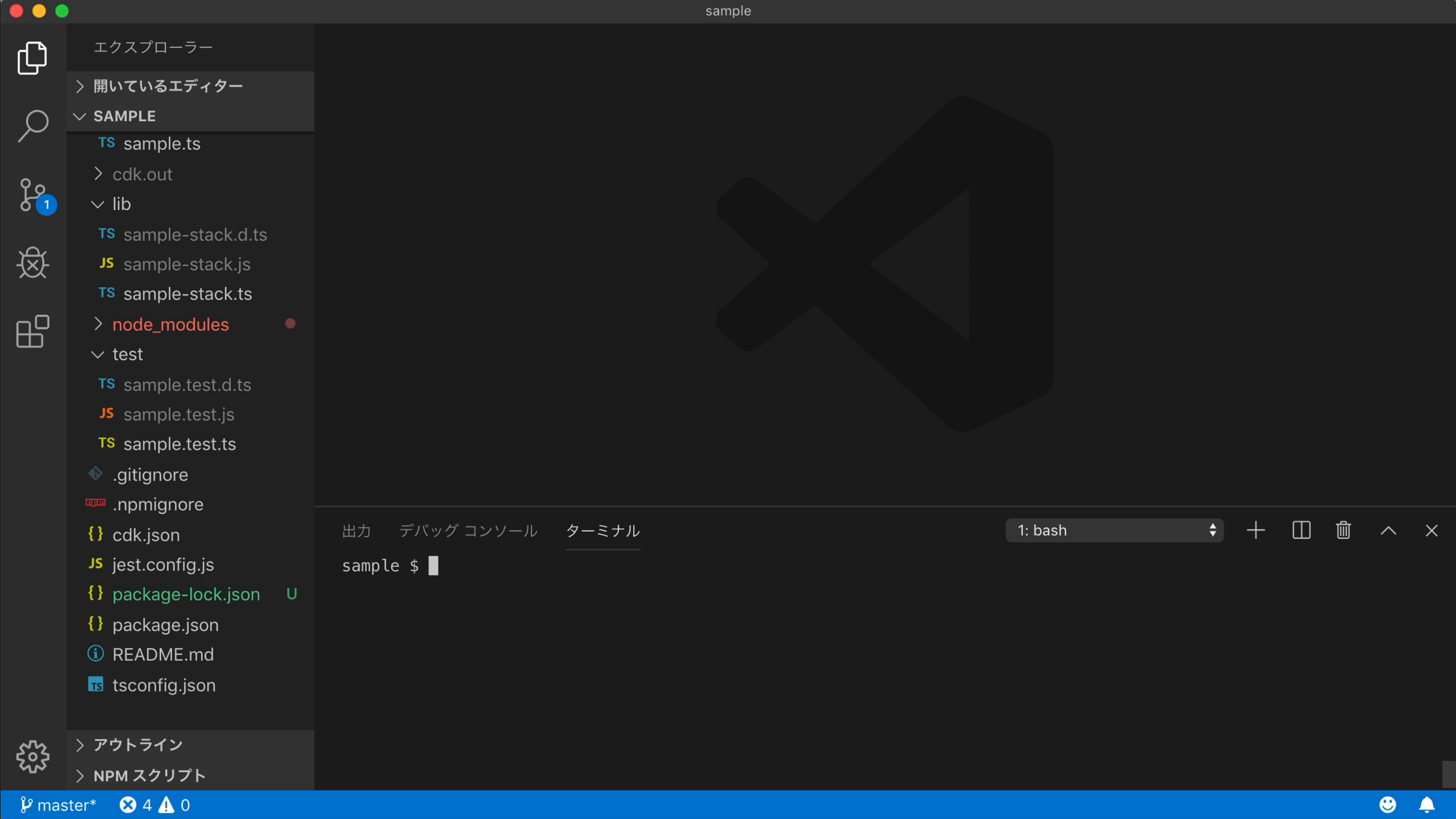

sample.ts — sample
エクスプローラー
開いているエディター
SAMPLE
bin
sample.d.ts
sample.js
sample.ts
cdk.out
lib
sample-stack.d.ts
sample-stack.js
sample-stack.ts
node_modules
test
.gitignore
.npmignore
cdk.json
jest.config.js
package-lock.json
package.json
README.md
tsconfig.json
アウトライン
NPM スクリプト
sample.ts
bin > sample.ts > ...
#!/usr/bin/env node
import cdk = require('@aws-cdk/core');
import { SampleStack } from '../lib/sample
const app = new cdk.App();
new SampleStack(app, 'SampleStack');
sample-stack.ts
lib > sample-stack.ts > SampleStack > constructor
import sns = require('@aws-cdk/aws-sns');
import subs = require('@aws-cdk/aws-sns-su
import sqs = require('@aws-cdk/aws-sqs');
import cdk = require('@aws-cdk/core');
export class SampleStack extends cdk.Stack
constructor(scope: cdk.App, id: string,
super(scope, id, props);
const queue = new sqs.Queue(this, 'Sam
visibilityTimeout: cdk.Duration.seco
});
const topic = new sns.Topic(this, 'Sam
topic.addSubscription(new subs.SqsSubs
}
}
master*
0 0
行 6、列 37
スペース: 2
UTF-8
LF
TypeScript
3.6.2

デモの流れ

1. 環境の準備
   - Node.js 導入済み
   1. cdk インストール
   2. cdk bootstrap
   3. cdk init
   4. コードの確認

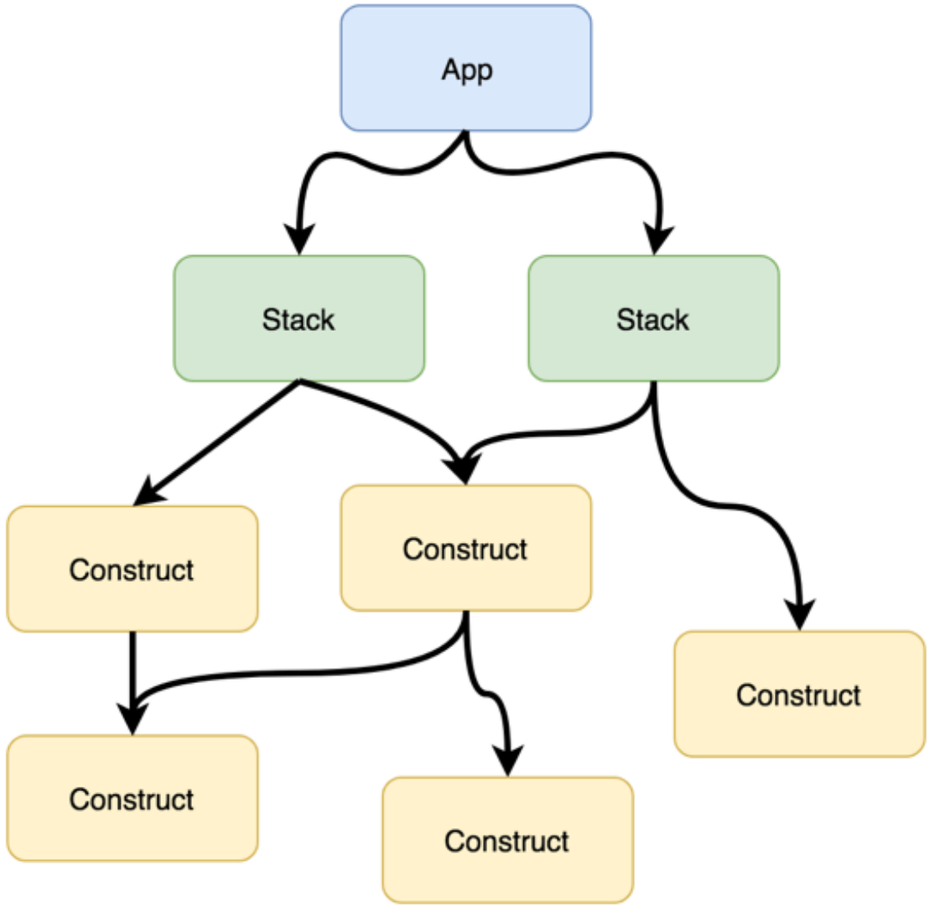
2. 最初のデプロイ
   1. npm build
   2. cdk synth
   3. cdk deploy

3. コード変更とデプロイ
   1. テストの実施(OK)
   2. テストコード変更(NG)
   3. コード修正と再テスト(OK)
   4. cdk diff
   5. cdk deploy


sample
エクスプローラー
開いているエディター
SAMPLE
sample.ts
cdk.out
lib
sample-stack.d.ts
sample-stack.js
sample-stack.ts
node_modules
test
sample.test.d.ts
sample.test.js
sample.test.ts
.gitignore
.npmignore
cdk.json
jest.config.js
package-lock.json
U
package.json
README.md
tsconfig.json
アウトライン
NPM スクリプト
出力
デバッグ コンソール
ターミナル
1: bash
sample $
master*
4
0

DEV DAY

CDK コンセプト



CDK アプリケーションの構成

App

- CloudFormationテンプレートの生成とデプロイに利用する最上位要素
- 複数のStackとその依存関係を定義

Stack

- CloudFormation Stack に該当しデプロイ可能な最小単位
- リージョンとアカウントを保持

Construct

- Stackに作成されるAWSリソース
- 標準でAWS Construct Libraryを提供
- 独自に定義したり配布することが可能

Construct

AWS CDK アプリの基本ビルディングブロック

- Cloudコンポーネントを表し、CloudFormationがリソースを作成するために必要なすべてをカプセル化

- 単一のリソースを表現したり（Amazon S3 Bucketなど）、複数のAWS CDKリソースをまたがるハイレベル コンポーネントを表現することも可能

- 標準で提供する AWS Constructs Libraryのほか、既存Constructを継承して独自のConstructを定義したり作成したConstructをパッケージとして配布することが可能

AWS Constructs Library

AWS CDK が標準で提供する Construct のライブラリ

- High-level constructs
  - デフォルト値や便利なメソッドを定義したAWSリソースを表すクラス
  - 例）クラス s3 は メソッド s3.Bucket.addLifeCycleRule() を持つ

- Low-level constructs
  - CloudFormationリソースおよびプロパティと1:1で対応（自動生成される）
  - CfnXXXという名前 （例：s3.CfnBucket は AWS::S3::Bucket を意味）
  - すべてのプロパティを明示的に設定する必要がある

- Patterns
  - 複数のリソースを含む一般的な構成パターンを事前に定義したもの
  - aws-esc-patterns.LoadBalancedFargateServiceなど

DEV DAY

CDK DiveDeep

- テストコード
- 複数スタック
- アクセス許可
- パラメータ

aws

CDK テストコード

"Testing infrastructure with the AWS Cloud Development Kit (CDK)"

https://aws.amazon.com/jp/blogs/developer/testing-infrastructure-with-the-aws-cloud-development-kit-cdk/

- Snapshot tests (golden master tests)
  - あるべきCFnテンプレート全体を用意し、CDKで作成したテンプレートが一致することを確認する
  - CDKの開発では「Integration test」として使用
- Fine-grained assersions
  - CDKで作成したテンプレートの一部をチェックし、指定したリソースが特定のプロパティを持つことを確認する
  - expect(stack).toHaveResource('AWS::SQS::Queue', {プロパティ}) の形で検証
- Validation tests
  - Construct に与えられたパラメータが正しいことを検証するなど一般的なユニットテスト
  - expect(...).toThrowError() など

(注) @aws-cdk/assert ライブラリはデベロッパープレビューの状態であり、実装は今後変更される可能性があります

CDK テストコードの例

Snapshot test code

```
import { SynthUtils } from '@aws-cdk/assert';
import cdk = require('@aws-cdk/core');
import vpcstack = require('../lib/vpc-stack');

test('VPC Stack template integ', () => {
    const app = new cdk.App();
    const stack = new vpcstack.VPCStack(app, 'MyTestStack');
    expect(SynthUtils.toCloudFormation(stack)).toMatchSnapshot();
});
```

Fine-grained test code

```
import { expect as expectCDK, haveResource } from '@aws-cdk/assert';
import cdk = require('@aws-cdk/core');
import Sample = require('../lib/sample-stack');

test('SQS Queue Created', () => {
    const app = new cdk.App();
    const stack = new Sample.SampleStack(app, 'MyTestStack');
    expectCDK(stack).to(haveResource("AWS::SQS::Queue",{
        VisibilityTimeout: 300
    }));
});
```

Snapshot test 実行例

```
basestack $ npm run test

> basestack@0.1.0 test /Users/ohmurayu/work/cdk-devday/basestack
> jest

 PASS  test/vpc-stack.test.ts
  ✓ VPC Stack template integ (212ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   1 passed, 1 total
Time:        2.743s
Ran all test suites.
```

Fine-grained test 実行例

```
sample $ npm run test

> sample@0.1.0 test /Users/ohmurayu/work/cdk-devday/sample
> jest

 PASS  test/sample.test.ts
  ✓ SQS Queue Created (78ms)
  ✓ SQS Queue Created - Short (41ms)
  ✓ SNS Topic Created (36ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        3.767s
Ran all test suites.
```

（"npm run test -- -u" で現在のスナップショットを保存）

CDK 複数スタックの管理

- スタック間でConstructを参照すればOK
- スタックの依存順序が定義可

```
const app = new cdk.App();
const vpc_stack = new VPCStack(app, 'VPCStack');
const alb_stack = new ALBStack(app, 'ALBStack', vpc_stack.vpc);
alb_stack.addDependency(vpc_stack);
```

依存関係のあるスタックの実装例↑

デプロイの例（ALBStack を作ると VPCStack も作られる）→

```
basestack $ cdk deploy ALBStack
Including dependency stacks: VPCStack
VPCStack
VPCStack: deploying...
VPCStack: creating CloudFormation changeset...
  0/25 | 17:27:47 | CREATE_IN_PROGRESS   | AWS::
 24/25 | 17:30:21 | CREATE_COMPLETE      | AWS::

 ✅ VPCStack

Stack ARN:
arn:aws:cloudformation:ap-northeast-1:340935377
ALBStack
ALBStack: deploying...
ALBStack: creating CloudFormation changeset...
 0/2 | 17:30:32 | CREATE_IN_PROGRESS   | AWS::C
 2/4 | 17:43:28 | CREATE_IN_PROGRESS   | AWS::E
2/4 Currently in progress: ALBStack, alb8A8B13C2

 ✅ ALBStack
```

自動生成されたエクスポート値 →

エクスポート名	エクスポート値	スタックの名前
VPCStack:ExportsOutputRefTheVPC92636AB00B2A4A70	vpc-0b9bbbcda361a209f	VPCStack

CDK アクセス許可の与え方

- S3 や DynamoDB などアクセス対象となる Construct は アクセス許可を与えるため grant から始まるメソッドを持つ
  - 例） grantRead() , grantWrite(), grantReadWrite()

- IGrantable インタフェースを実装した オブジェクトを渡すと適切な IAM Role/Policy が設定される
  - 例) s3bucket.grantRead(lambdaFunction);

CDK パラメータ

- Environment
  - CDKコマンドを実行するシェルの環境変数
  - 既存環境の情報を AWS CLI を使って参照する場合などに必要
  - process.env.CDK_DEFAULT_ACCOUNT のようにアクセス
- Context
  - CDKコードを実行する際の周囲の環境（Dev/Prod, VPC IDなど）を定義
  - cdk.json ファイルや cdk --context key=value として指定
  - CDK が既存環境を参照した場合は cdk.context.json ファイルに情報を保存 (AZ構成など)
  - this.node.tryGetContext('key') のようにアクセス
- SSM ParameterStore
  - AWS上で構成情報を保持するパラメータストア
  - CDKコードとは別のライフサイクルでメンテナンスする情報を保持
  - SecureString が渡せる場所は CFn の仕様に依存するので注意

パラメータの例: 既存VPCに DB接続情報を持つ Fargateを立てる

←App

```ts
const app = new cdk.App();
const fargate_stack = new FargateStack(app, 'FargateStack', {
  env: {
    account: process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEFAULT_REGION
  }});
```

↓Stack

```ts
export class FargateStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const vpc = ec2.Vpc.fromLookup(this, 'defVPC', {
      vpcName: this.node.tryGetContext('vpcName')
    });
    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });
    const user = ssm.StringParameter.fromStringParameterAttributes(this, 'SSMUser', {
      parameterName: '/MySystem/Dev/DB/User'
    });
    const password = ssm.StringParameter.fromSecureStringParameterAttributes(this, 'SSMPassword', {
      parameterName: '/MySystem/Prod/DB/Password',
      version: 1
    });
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService", {
      cluster: cluster,
      image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
      environment: { 'user': user.stringValue },
      secrets: { 'password': ecs.Secret.fromSsmParameter(password) }
    });
  }
}
```

↓cdk.json

```json
{
  "app": "npx ts-node bin/basestack.ts",
  "context": {
    "vpcName": "myvpc"
  }
}
```

パラメータの例:
既存VPCに
DB接続情報を持つ
Fargateを立てる

←App

```ts
const app = new cdk.App();
const fargate_stack = new FargateStack(app, 'FargateStack', {
  env: {
    account: process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEFAULT_REGION
  }});
```

Environment:
デフォルト環境の情報取得

↓Stack

```ts
export class FargateStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const vpc = ec2.Vpc.fromLookup(this, 'defVPC', {
      vpcName: this.node.tryGetContext('vpcName')
    });
    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });
    const user = ssm.StringParameter.fromStringParameterAttributes(this, 'SSMUser', {
      parameterName: '/MySystem/Dev/DB/User'
    });
    const password = ssm.StringParameter.fromSecureStringParameterAttributes(this, 'SSMPassword', {
      parameterName: '/MySystem/Prod/DB/Password'
      version: 1
    });
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService", {
      cluster: cluster,
      image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
      environment: { 'user': user.stringValue },
      secrets: { 'password': ecs.Secret.fromSsmParameter(password) }
    });
  }
}
```

Environment+Context で
既存環境からVPCを取得

SSMの平文パラメータは
CFn パラメータ化（デプロイ時に確定）

SSMのSecureStringは
CFn DynamicReference化（Task実行時に確定）

SSMパラメータをTaskDefinitionの
environment と secret に渡す

↓cdk.json

```json
{
  "app": "npx ts-node bin/basestack.ts",
  "context": {
    "vpcName": "myvpc"
  }
}
```

Context:
VPCの名前を指定

DEV DAY

AWS CDK の情報源

ドキュメントと参考資料

GitHub
https://github.com/aws/aws-cdk

ドキュメント（Developer Guide）
https://docs.aws.amazon.com/cdk/latest/guide/home.html

APIリファレンス
https://docs.aws.amazon.com/cdk/api/latest/docs/aws-construct-library.html

CDK meetup (7/19開催) の発表資料 https://awsclouddevelopmentkitcdkmeetu.splashthat.com/

AWSのインフラはプログラミングコードで構築！AWS Cloud Development Kit 入門（AWS 福井）
https://www.slideshare.net/AmazonWebServicesJapan/awsaws-cloud-development-kit

CDKを用いたモダンなECSクラスタの構築と運用（株式会社はてな 原田陽太様）
https://speakerdeck.com/cohalz/aws-cloud-development-kit-cdk-meetup

はてなブログ タグとCDK（株式会社はてな 中澤亮太様）
https://speakerdeck.com/aereal/the-epic-of-aws-cdk-and-hatena-blog-tag

Ansible + CloudFormation を AWS CDK に移行する方法 （ディー・エヌ・エー 佐藤学様）
https://speakerdeck.com/manab/ansible-plus-cloudformation-wo-aws-cdk-niyi-xing-surufang-fa

ハンズオン

- サンプル
  - https://github.com/aws-samples/aws-cdk-examples
- ワークショップ（おすすめ）
  - https://cdkworkshop.com
  - 日本語版: http://bit.ly/cdkworkshopjp

DEV DAY

Thank you!

大村幸敬
ohmurayu@amazon.co.jp
Twitter: @yktko

aws

DEV DAY

Please complete the session survey in the mobile app.

aws