AWS Black Belt Online Seminar

AWS Cloud Development Kit (CDK)

Basic #1

概要

高野 賢司

Solutions Architect
2023/07

こうの　けんじ
高野 賢司

ソリューションアーキテクト @名古屋
アマゾンウェブサービスジャパン合同会社

Baseline Environment on AWS (BLEA) 開発者
https://github.com/aws-samples/baseline-environment-on-aws

好きな AWS サービス： AWS CDK

アジェンダ

1. AWS Cloud Development Kit (CDK) とは
2. AWS CDK のコンセプト
3. AWS CDK と他のサービスの連携
4. TypeScript での開発の流れ
5. 各言語におけるプロジェクト構成
6. デモ（TypeScript でのプロジェクト作成 ～ Amazon VPC をデプロイ）
7. AWS CDK の学習リソース

AWS Cloud Development Kit (CDK) とは

Infrastructure as Code (IaC) とは？

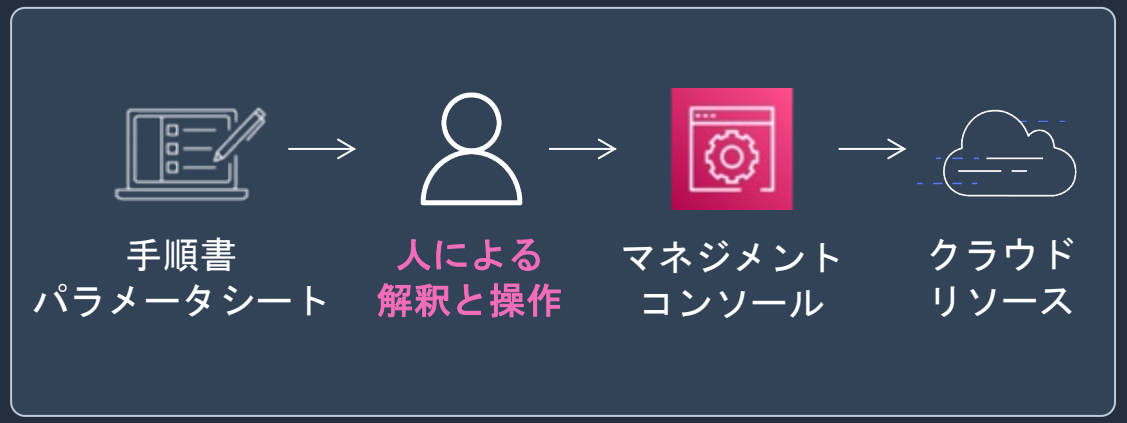
手動ではなく、コードによって
インフラストラクチャの管理やプロビジョニングを行うプロセス

ソフトウェア開発のプラクティスをインフラ構築の自動化に活かす
継続的デリバリーに必須の技術のひとつ

https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/infrastructure-as-code.html

なぜ Infrastructure as Code (IaC) が必要なのか？

手動操作（マネジメントコンソール）



スクリプト（CLI, SDK）

手動操作と手続き型スクリプトの課題

- 現在の状態がわからずリリースしづらい
- 人による解釈違いや操作ミスのリスクあり
- 何度も同じ構成を作るのが大変
- 手順書やスクリプトの作成に時間がかかり継続的な更新やテストが困難（陳腐化）
- リソースの状態による判断やエラー処理、ロールバックを網羅しづらい

Infrastructure as Code (IaC) のメリット

コスト削減

- 手順書の作成、メンテナンス、引継ぎコストを削減
- 手順書と実環境の乖離によるブラックボックス化を防止
- デプロイ作業時間を削減
- 必要なときにリソースを作成、まとめて破棄

スピードアップ

- CI/CD で自動テスト、デプロイ
- 変更を予測可能にして頻繁にデプロイ
- 同じ構成を何度でもデプロイ
- 構成パターンとベストプラクティスの共有

リスク低減

- 人的ミスの排除
- バージョン管理による変更の追跡と承認プロセス
- 信頼できる唯一の情報源としての Git リポジトリ
- 必要に応じて前のバージョンにロールバック

AWS Cloud Development Kit (CDK)

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

AWS CDK

使い慣れたプログラミング言語で
クラウドリソースを定義できる
OSS のフレームワーク

Meeting developers where they are

開発者体験を改善

- アプリと同じ言語で記述でき
  ドメイン固有言語の習得が不要
- ソフトウェア開発の技法と
  プラクティス、ツールを活用
- 型付けとバリデーションで
  素早くフィードバックを得る
- リソースとスタックの依存関係を
  自動的に解決

アプリ全体をコードで定義

- クラウドリソースだけでなく
  AWS Lambda 関数のコードや
  コンテナイメージなど
  アプリ全体をまとめて管理
- CI/CD パイプラインを自動構築
- 複数の AWS アカウントに
  またがる環境を管理

高レベルの抽象化

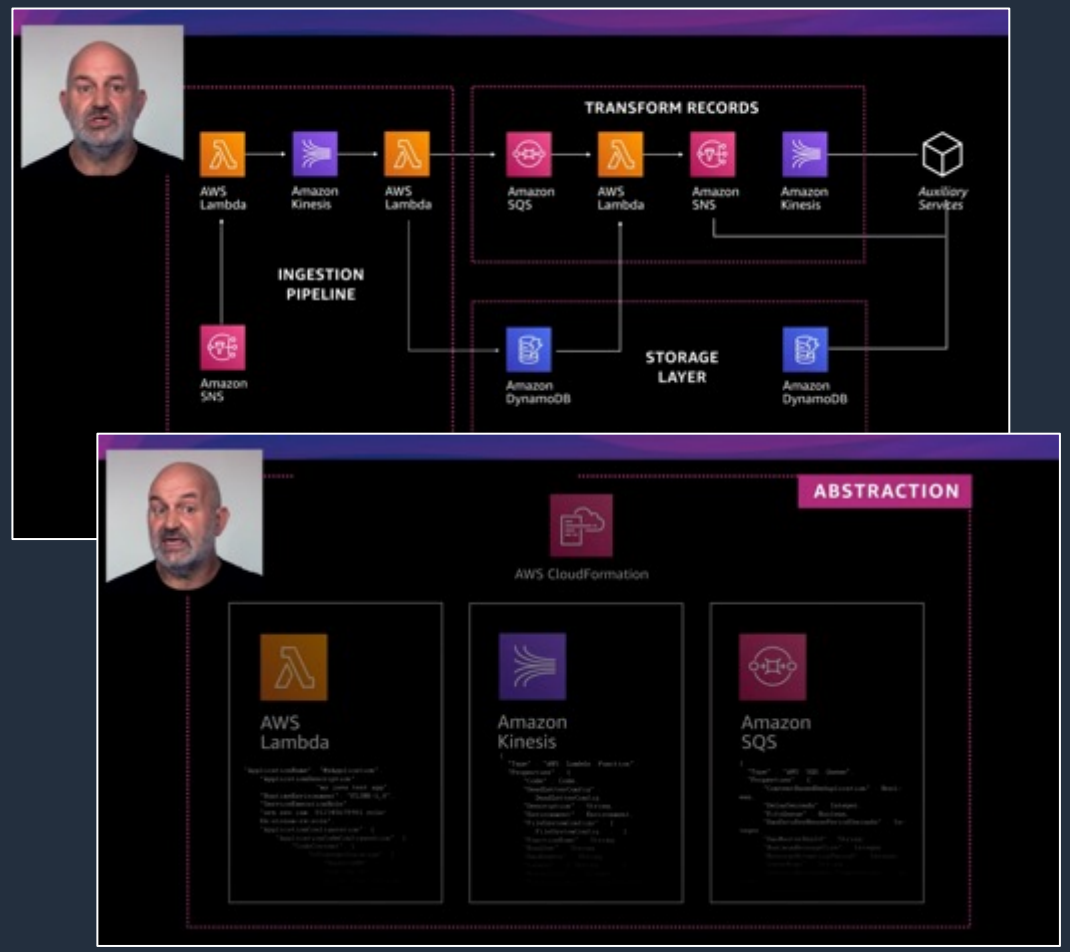
- 複雑な設定を抽象化し
  コード量と学習コストを削減
- いつでも抽象化から抜け出して
  個別の設定にアクセス可能
- ベストプラクティスに基づく
  デフォルト値と設定を提供
- チームの構成パターンや
  ガードレールを共有・再利用

6つのプログラミング言語をサポート

TypeScript で書かれた単一のコードベースを
jsii によって各言語のコードにコンパイルして提供

JSii https://github.com/aws/jsii

なぜ Amazon は AWS CDK を作ったか？



https://www.youtube.com/watch?v=AYYTrDaEwLs

Amazon.com の検索システムを再構築し
トレンド商品をリアルタイムに特定したい。

AWS CloudFormation で構築していたが・・・

- 複数のチームが独立して開発・デプロイ可能にするためにモジュール化したい
- 繰り返し複雑なものを作成することを避けるために抽象化したい
- JSON や YAML のような設定言語よりも自分たちのアイデアを表現できる言語を使いたい

これらの課題を解決するために作成された
オブジェクト指向ライブラリが AWS CDK の起源。

AWS の社内で CDK はデファクトスタンダードになった

AWS CDK のしくみ

テンプレートと状態を**スタック**として管理

開発者

プログラミング言語で
インフラ構成を定義

TypeScript JavaScript Python

Java .NET Go

$ cdk deploy

AWS CDK Toolkit (CLI)

Synthesize

CloudFormation テンプレートを**合成**
アセットをバンドル、デプロイ

Node.js

Node.js のコマンドラインツール

各言語の**コンストラクトライブラリ**を利用

テンプレートを実行

AWS CloudFormation

テンプレートに定義された
状態になるようリソースを操作

- S3 バケットがなかったら作成
- 状態が違っていたら更新
- 状態が同じなら何もしない etc.

API 呼び出し

クラウドリソース

目的の状態に収束

アセットをデプロイ

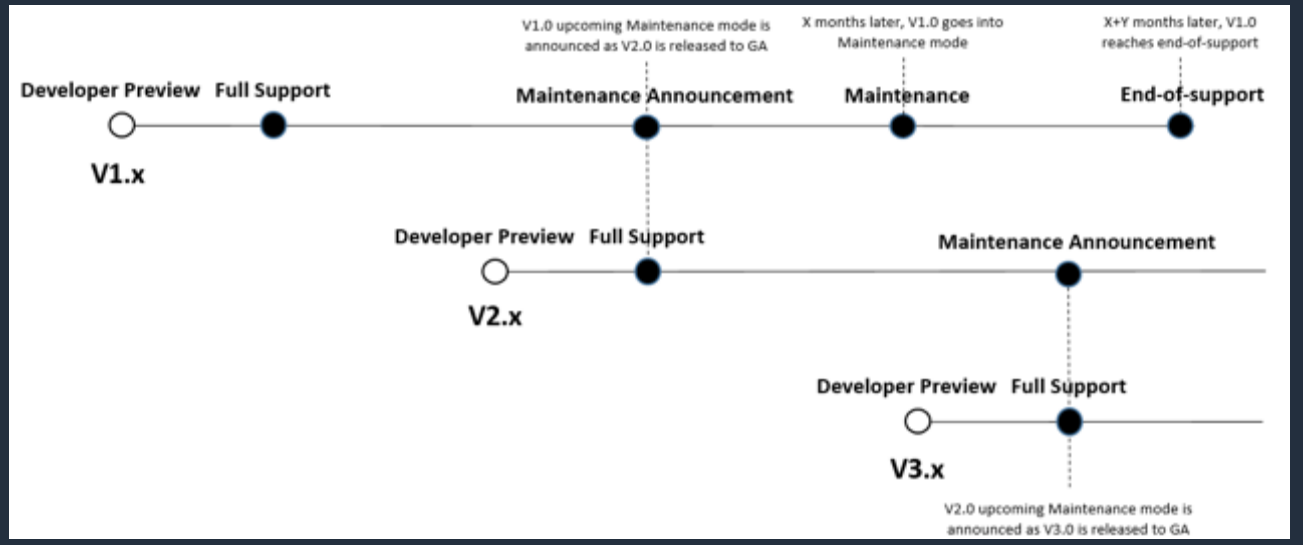
- Lambda関数のコード
- コンテナイメージ
- ファイル など

AWS CDK v2　2021年12月リリース

- コンストラクトライブラリを
  1つのモノリシックなパッケージに統合。
  依存関係の管理が容易に
- Semantic Versioning に準拠し
  安全にアップデート可能に
- v2 へのマイグレーションガイドを提供
  https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/migrating-v2.html

experimental / deprecated な API を使っていなければ パッケージのインポートを 修正するだけ

https://mplay-assets.s3.amazonaws.com/sites/awsreinv21/_uploads/assets/otmqsererizzcufy_awsreinv21.pdf



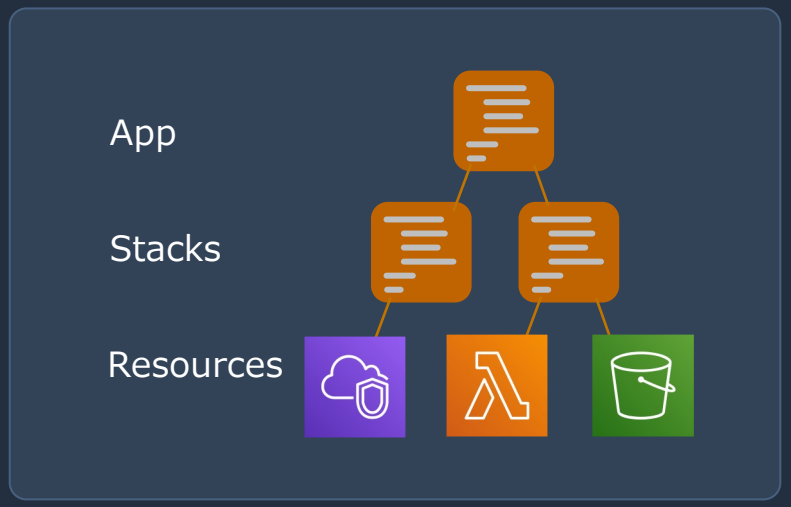
AWS CDK のメンテナンスポリシー

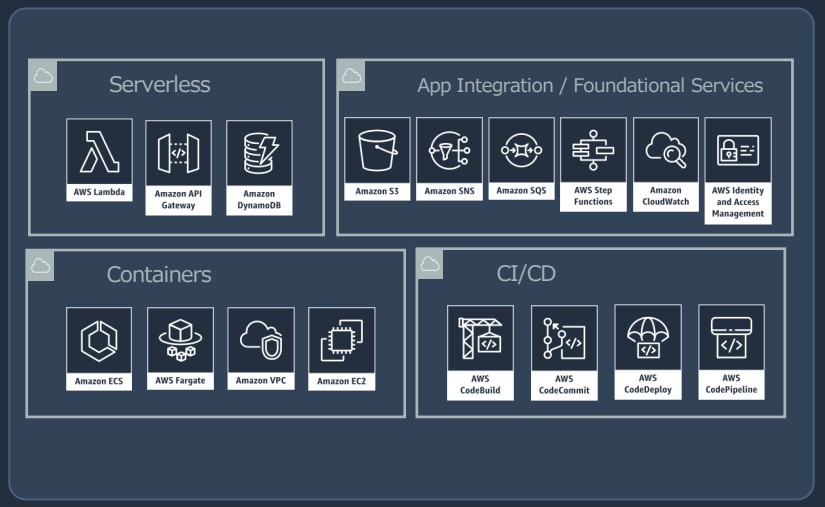
https://github.com/aws/aws-cdk-rfcs/blob/master/text/0079-cdk-2.0.md#aws-cdk-maintenance-policy

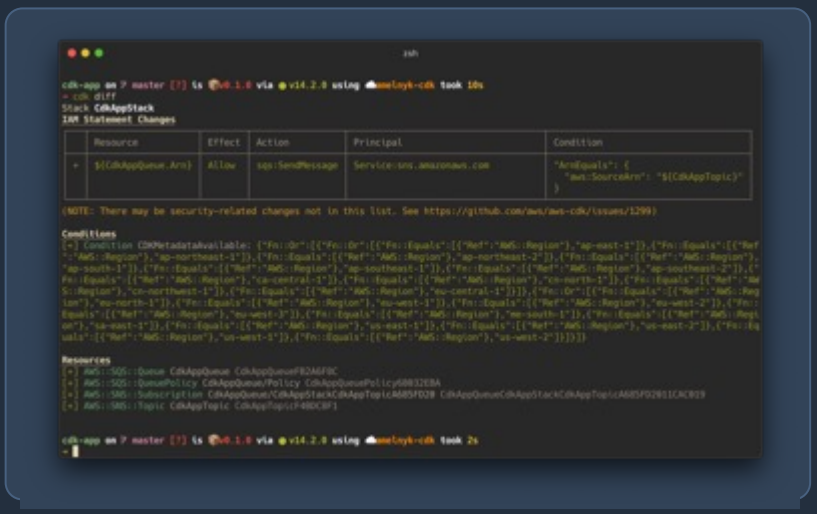
- メンテナンスフェーズに入る6ヶ月前に告知
- メンテナンスフェーズは12ヶ月間、
  重大なバグ修正とセキュリティパッチのみ提供
- AWS CDK v1 のサポートは 2023年6月に終了済み

AWS CDK のコンセプト

AWS CDK の構成要素







コアフレームワーク

npm package: constructs

コンストラクトライブラリ

npm package: aws-cdk-lib

AWS CDK Toolkit (CLI)

npm package: aws-cdk

プログラミング言語ごとに**専用**のパッケージ
※ サードパーティ製または自作のコンストラクトライブラリも利用可能

プログラミング言語によらず**共通**
※ Node.js で動作

AWS CDK の概念

https://docs.aws.amazon.com/cdk/v2/guide/home.html

App

- アプリケーション全体
- 複数のAWSアカウント、リージョンにまたがることが可能

Stack

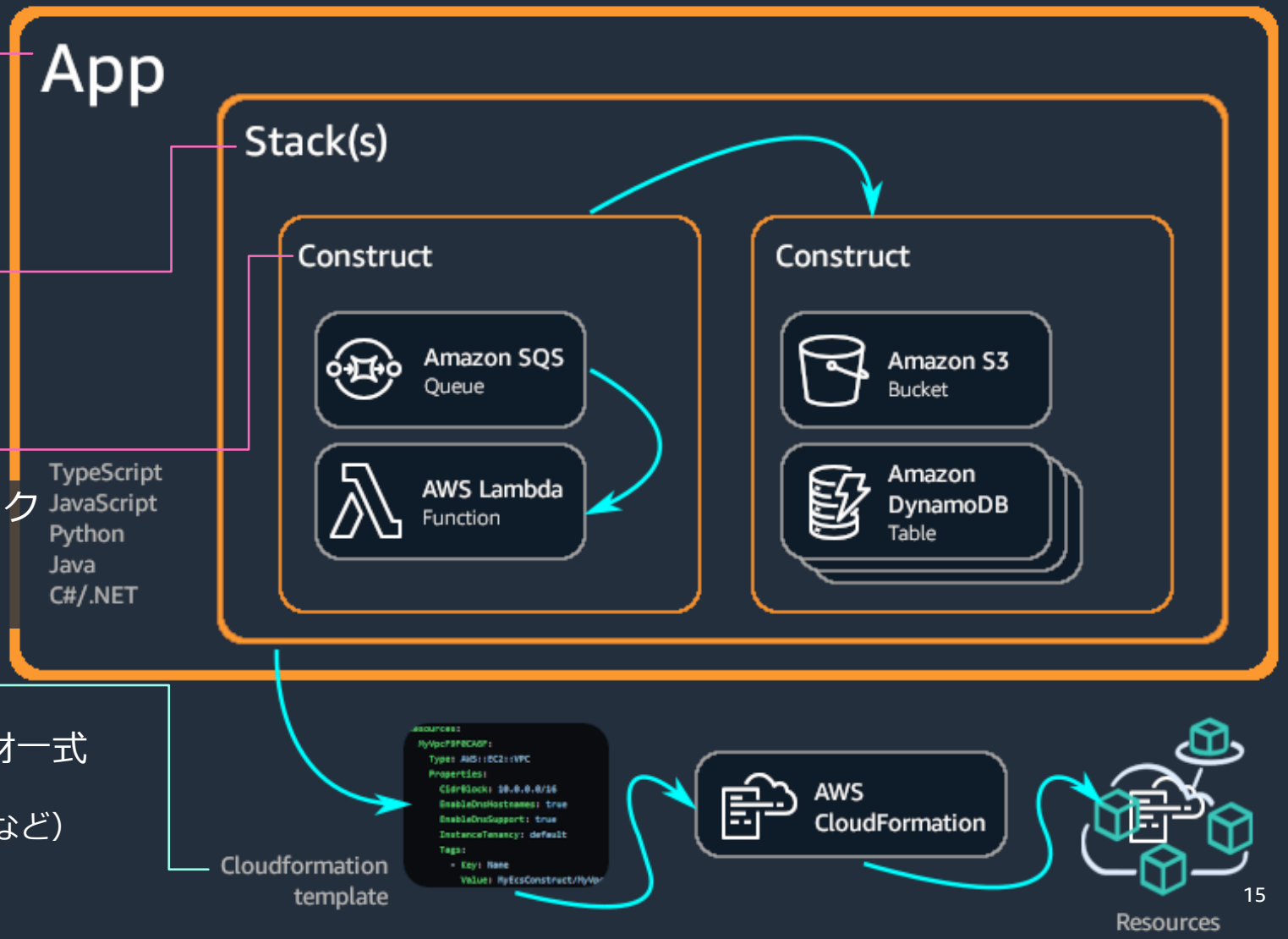
- CloudFormation スタックに対応
- デプロイ可能な最小単位

Construct

- CDKの最も基本的なビルディングブロック
- 1つまたは複数のAWSリソースを表現
- ユーザーにより定義・配布が可能

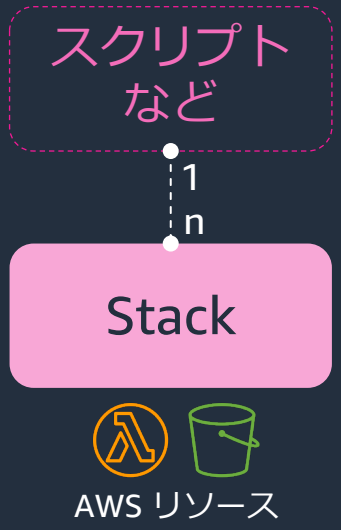
Cloud Assembly

- CDK App の出力。デプロイに必要な資材一式
  - CloudFormation テンプレート
  - アセット（ファイル、Docker イメージなど）



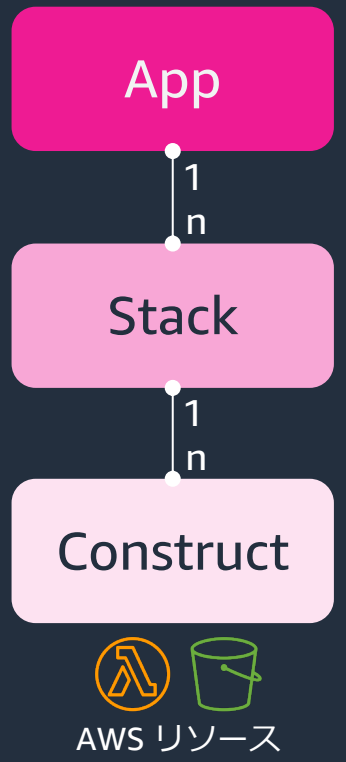
AWS CDK の概念 - CloudFormation との比較

AWS CloudFormation



- スタックを1つずつデプロイ
  $ aws cloudformation deploy
  --stack-name HogeStack
  --template-file hoge.yaml
- 依存関係を考慮して
  人がデプロイ順序を決める
- 複数のスタックを一括デプロイ
  するためにはスクリプトや
  Makefile, CodePipeline 等が必要
- CloudFormation Module による
  カプセル化と再利用

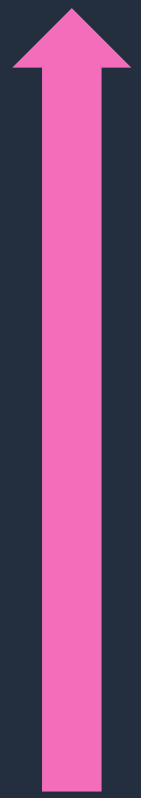
AWS CDK



- 複数のスタックをまとめて
  1つのアプリとしてデプロイ
  (スタックを指定してデプロイも可能)
  $ cdk deploy --all
- スタック間の依存関係を CDK が
  解決してデプロイ順序を決定
- Construct による抽象化と再利用
- 複数の Stack をまとめる Stage
  を追加することで環境の複製も
  容易 (Dev, Staging, Prod ...)

AWS Constructs Library

AWS CDK が標準で提供する CONSTRUCT のライブラリ



Patterns（L3）

- 複数のリソースを含む一般的な構成パターンを抽象化
  - `aws-ecs-patterns.LoadBalancedFargateService` など

High-level constructs (L2)

- デフォルト値や便利なメソッドを定義した 単一の AWS リソースを表すクラス
  - `s3.Bucket` クラスのインスタンスは `addLifeCycleRule()` メソッドを実装
- より特定のシナリオに合わせて単一リソースを抽象化した **L2.5 constructs** も存在
  - `aws-lambda-nodejs.NodeJsFunction`, `eks.FargateCluster` など

Low-level constructs (L1)

- 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

CloudFormation テンプレートの例

例: IAM User からの
読取専用アクセスを許可する S3 Bucket を作成

```
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
  MyUser:
    Type: AWS::IAM::User
  MyUserPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
            Effect: Allow
            Resource:
              - Fn::GetAtt: [ MyBucket, Arn ]
              - Fn::Sub: "${MyBucket.Arn}/*"
        Version: "2012-10-17"
      PolicyName: MyUserPolicy
      Users:
        - Ref: MyUser
```

AWS CloudFormation
template language

L1 Constructs を使用した例

CloudFormation テンプレートとほぼ1:1対応
型チェックや補完、ループなどは使用可能

AWS CloudFormation resources

AWS CDK

"L1"

CloudFormation から自動的に生成

AWS CloudFormation template language

```
const bucket = new CfnBucket(this, 'MyBucket');
const user = new CfnUser(this, 'MyUser');
new CfnPolicy(this, 'MyUserPolicy', {
  policyName: 'MyUserPolicy',
  policyDocument: new PolicyDocument({
    statements: [new PolicyStatement({
      actions: [
        's3:GetObject*',
        's3:GetBucket*',
        's3:List*'],
      resources: [
        bucket.bucketArn,
        `${bucket.bucketArn}/*`]
    })]
  }),
  users: [user.userName],
});
```

L2 Constructs を使用した例

コントリビューターが作成

AWS Construct Library

AWS CDK

"L2"

AWS CloudFormation resources

AWS CDK

"L1"

CloudFormation から自動的に生成

AWS CloudFormation template language

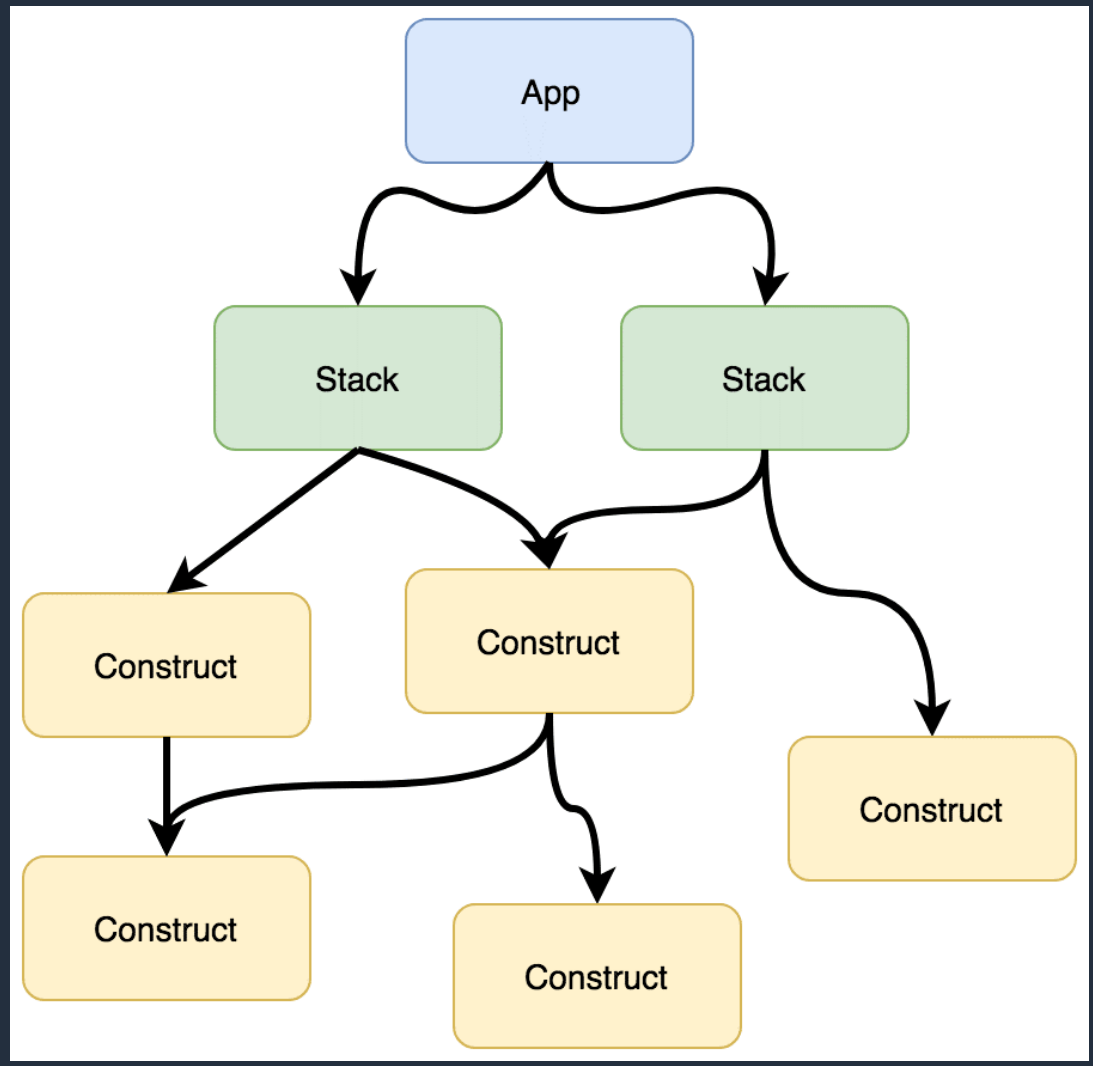
```
const bucket = new s3.Bucket(this, 'MyBucket');
const user = new iam.User(this, 'MyUser');

bucket.grantRead(user);
```

grant() メソッドにより IAM Policy を自動生成
コードから意図が明確に

Construct ツリー

- App をルートとして
  自由に Construct を構造化できる
- すべての Construct で
  明示的に scope (親) を指定して初期化
- AWS リソースを作成する Construct は
  Stack の子孫でなければならない
  - Stack クラスも Construct の一つ
- Node クラスで Construct ツリーにアクセス
- Aspect で各 Node への操作を実装可能 *

* Aspect の詳細は今後の Blackbelt または AWS CDK 開発者ガイドを参照



Construct ツリー の可視化

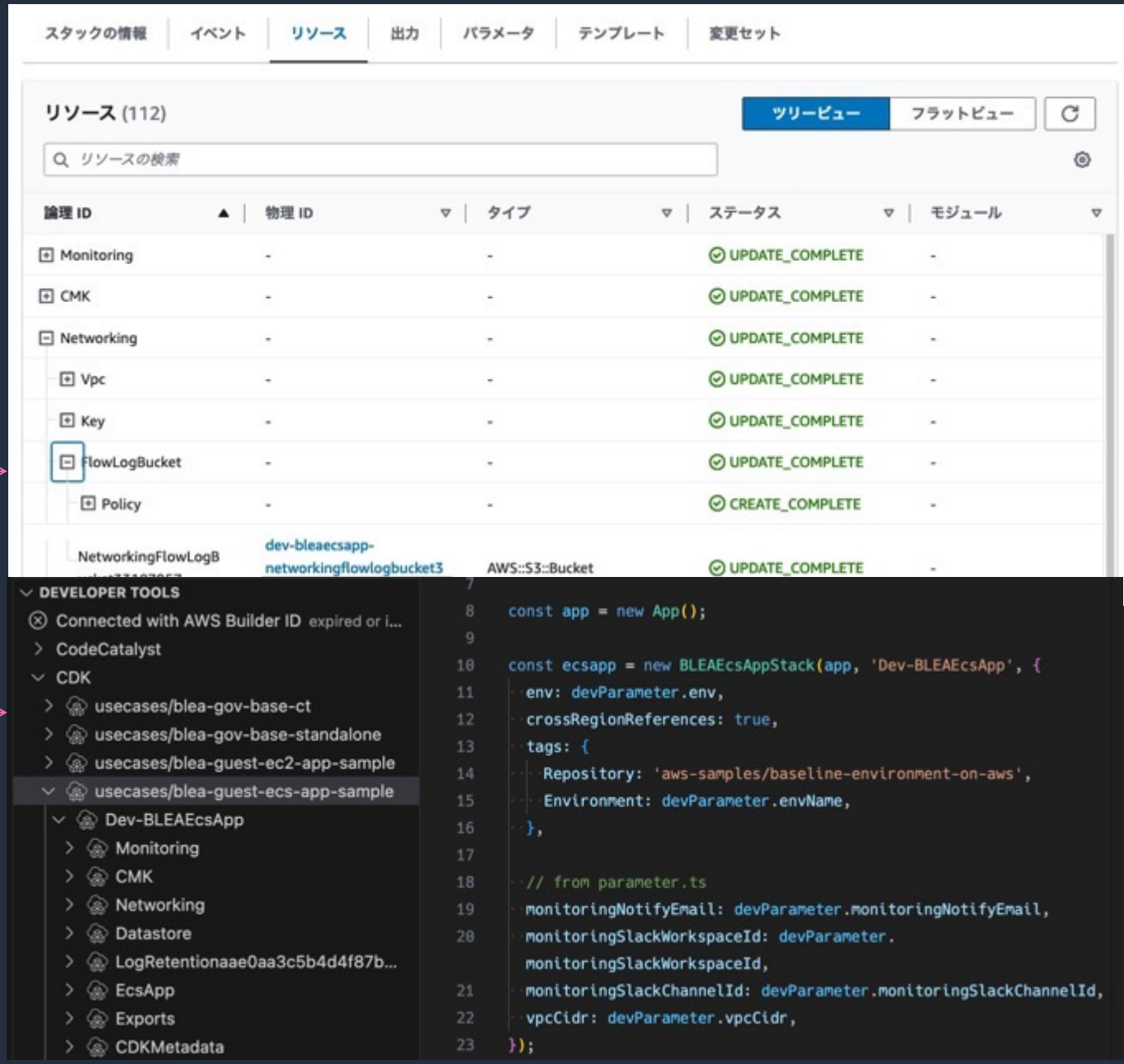
Construct は自由に構造化できる（ツリー構造）
構造化によってコードとリソースの可読性が向上

AWS CloudFormation コンソール

https://aws.amazon.com/jp/about-aws/whats-new/2022/09/aws-cloud-development-kit-cdk-announces-cdk-construct-tree-view-cloudformation-console/

AWS Toolkit for Visual Studio Code

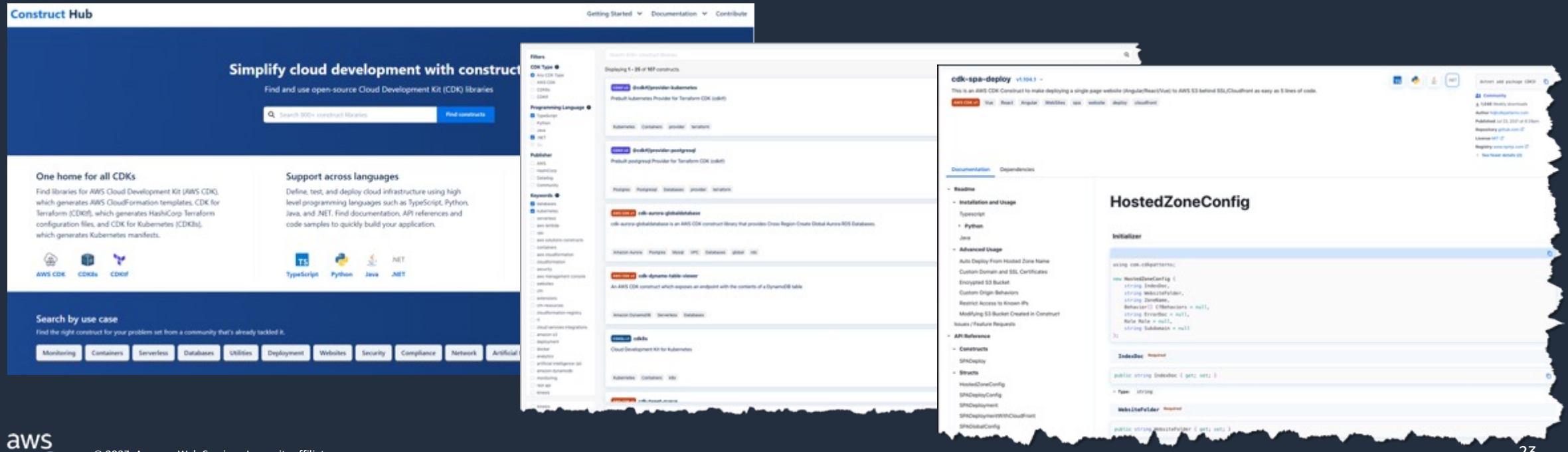
https://docs.aws.amazon.com/toolkit-for-vscode/latest/userguide/cdk-explorer.html



Construct Hub

https://constructs.dev/

- 1,000 以上のオープンソースのコンストラクトを公開
  - 公開されたコンストラクトを組み合わせることで、
    目的に合わせたアプリケーション環境をさらに迅速に構成できる
  - AWS CDK, CDK8s, CDKtf などのタイプやバージョン、言語、パブリッシャーなどで検索可能



エスケープハッチ (代替手段)

L2コンストラクトがない場合 (CloudFormation で対応しているリソースの場合)

- CfnBucket や CfnRole など、Cfn で始まる L1 コンストラクトを使う
- L1 コンストラクトもない場合、`cdk.CfnResource` を使う（CFnテンプレートとほぼ同等の記述）

L2 コンストラクトでプロパティが設定できない場合

- `construct.node.defaultChild` で L1 コンストラクトを取得し、プロパティを変更する

CloudFormation で対応していない機能の場合

- **Provider Framework** を使用してカスタムリソースを作成する
  https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.custom_resources-readme.html#provider-framework
- AWS の API を呼び出すシンプルなカスタムリソースは `AwsCustomResource` コンストラクトを使用することで、Lambda 関数のコードを記述することなく簡単に作成可能
  https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.custom_resources-readme.html#custom-resources-for-aws-apis

AWS CDK と
他のサービスの連携

AWS SAM でローカルデバッグを実行

AWS CDK で作成したサーバーレスアプリケーションを
AWS SAM CLI を使用してローカルでデバッグ可能

`sam local invoke`, `start-api`, `start-lambda` に対応。デプロイは CDK で行う

https://docs.aws.amazon.com/ja_jp/serverless-application-model/latest/developerguide/serverless-cdk-getting-started.html

AWS Amplify と CDK の連携

`$ amplify override` **... バックエンドリソースをCDKで上書き**

- AWS Amplify が自動生成したリソースを CDK でカスタマイズ
  IAM ロール、Cognito による認証機構、S3 バケット、DynamoDB テーブルに対応
  https://docs.amplify.aws/cli/restapi/override/

`$ amplify add custom` **... CDK でカスタムバックエンドを追加**

- AWS Amplify が生成できるバックエンドリソースに加え、CDK で任意のリソースを定義
  https://docs.amplify.aws/cli/custom/cdk/

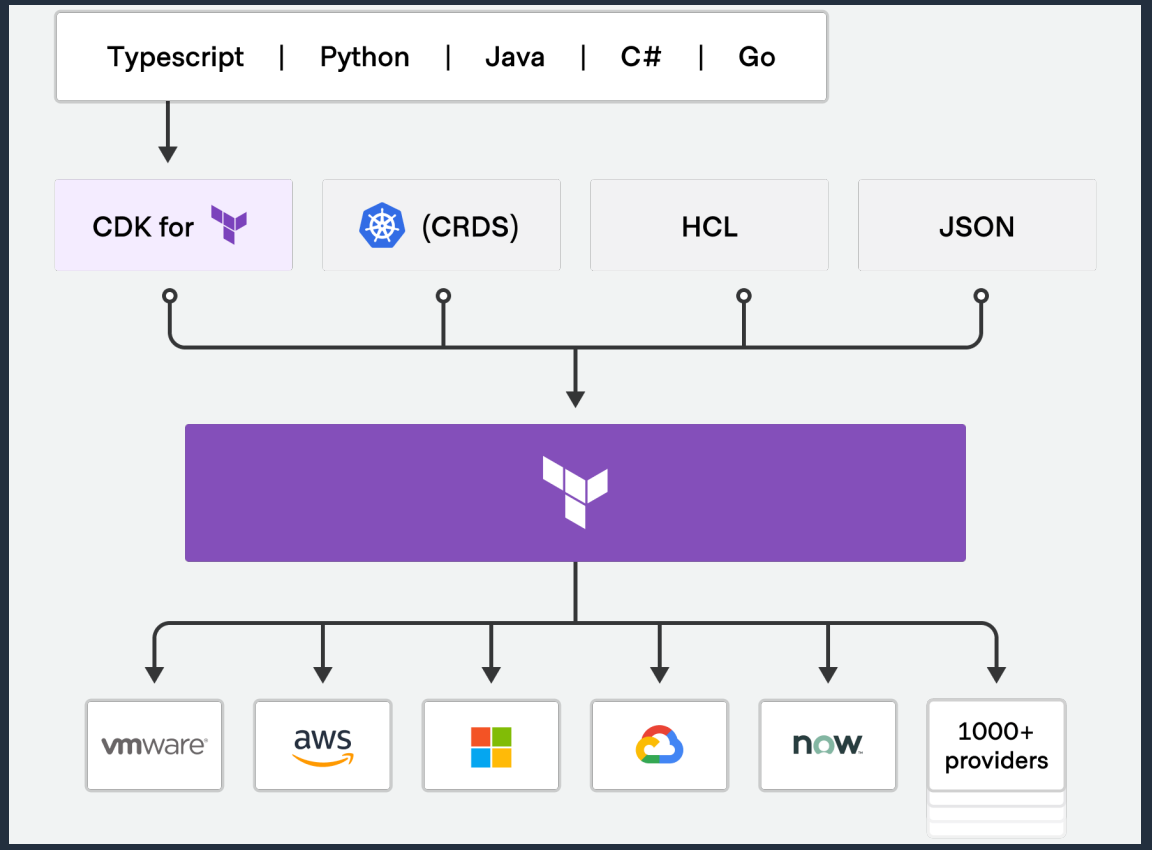
`$ amplify export` **... バックエンドリソースをCDKでエクスポート**

- AWS Amplify で生成したバックエンドをエクスポートして AWS CDK からデプロイ。
  CDK Pipelines や Amazon CodeCatalyst の CDK deploy action が利用可能
  https://docs.amplify.aws/cli/usage/export-to-cdk/

CDK for Terraform = CDKTF

https://www.terraform.io/cdktf

- HashiCorp 社と AWS CDK チームが共同開発
- AWS CDK がコードから CloudFormation テンプレートを生成するのと同様に、Terraform の JSON 構成を生成
- cdktf-cli により、初期化や合成の他、Terraform レジストリのプロバイダーをプロジェクトにインポート可能



CDK for Terraform on AWS 一般提供 (GA) のお知らせ
https://aws.amazon.com/jp/blogs/news/cdk-for-terraform-on-aws-jp/

CDK for Kubernetes = CDK8s

https://cdk8s.io/

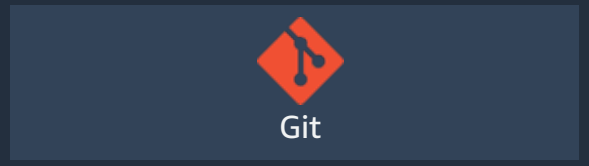
プログラミング言語で KUBERNETES のマニフェストファイルを生成できるツールキット

- ソースコードから Kubernetes のマニフェスト YAML を生成
  - 制御構文やクラス・継承などの概念で効率的に書くことが可能
  - エディタによる型チェック、サジェスト、API 仕様の参照
- オープンソースとして開発
  - ベストプラクティスを定義した Construct として拡張・共有
- 任意の Kubernetes クラスターで利用可能
- 言語サポート
  - TypeScript/JavaScript、Python、Java
- 任意の Kubernetes API バージョンとカスタムリソースを使用可能
- CDK8s+ で高レベルなコンストラクトを提供
  https://aws.amazon.com/jp/blogs/news/announcing-general-availability-of-cdk8s-plus-and-support-for-manifest-validation/

TypeScript での開発の流れ

AWS CDK の開発に必要なもの

Git



- **Git** https://git-scm.com/
  - IaC の原則としてバージョン管理は必須　※ AWS CDK の直接的な依存関係ではない

AWS CLI と言語ランタイム

- **AWS CLI v2** https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html
- **Node.js** https://nodejs.org/en/ ※ アクティブな LTS 版を推奨

JavaScript / TypeScript 以外の言語で記述する場合は
その言語のランタイムやコンパイラも必要

テキストエディタ / IDE

- **VSCode** https://code.visualstudio.com/ ※ またはお好きなテキストエディタ
  - 各言語のコード補完やリファクタリング機能を持つツールの利用を強く推奨

AWS CLI の認証情報を設定

IAM ユーザーのアクセスキーを使用する場合

Console
```
$ aws configure --profile <your-profile-name>
```

または環境変数 AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION を指定
https://docs.aws.amazon.com/cdk/v2/guide/getting_started.html#getting_started_prerequisites

AWS IAM Identity Center (SSO) を使用する場合

v2.18.0 ~

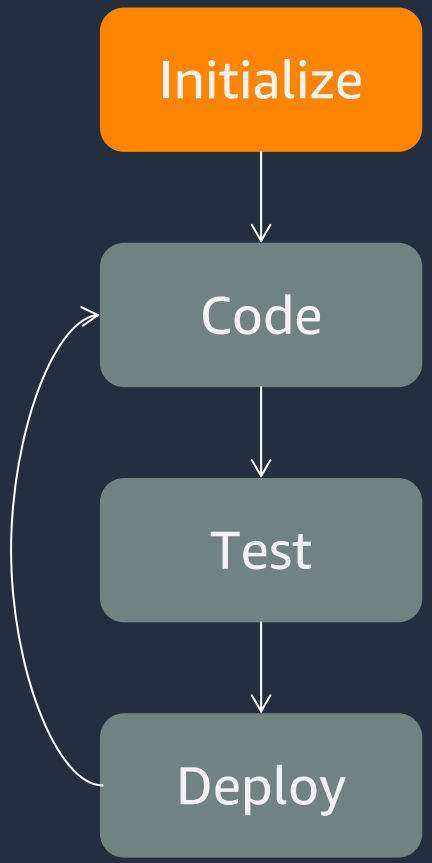
https://docs.aws.amazon.com/ja_jp/cli/latest/userguide/cli-configure-sso.html

Console
```
$ aws configure sso
```

一時的な認証情報を使用して
セキュリティベストプラクティスに準拠

AWS CDK (TypeScript) での開発フロー 1/4 Initialize

cdk init でプロジェクトを作成



Console

```
# ディレクトリを作成
$ mkdir cdk-sample
$ cd cdk-sample

# CDK プロジェクトを作成
$ npx aws-cdk init app --language=typescript

# CDK Bootstrapping
$ npx aws-cdk bootstrap
```

cdk init はディレクトリ名を使用してファイル名や Stack 名を生成

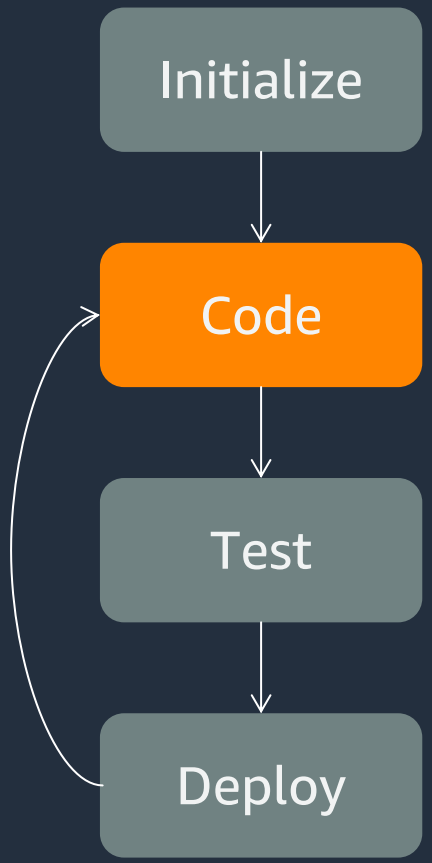
空のテンプレートを生成。 sample-app を指定するとサンプルが追加

CDK が使用する IAM ロール、S3 バケット、ECR リポジトリなどを作成
対象アカウント、リージョンにつき1回だけ実施

Tips: npx について

cdk コマンドをグローバルインストールした状態 (npm i -g aws-cdk) を前提に cdk init と記載されることも。
cdk init 時（npm プロジェクト外）は最新の aws-cdk を使用すること、
cdk synth や deploy 時（npm プロジェクト内）はローカルインストールの aws-cdk を使用することを目的として
本資料ではコマンドを npx aws-cdk に統一する。なお npx cdk としてもよい（一部状況下では少し挙動が異なる）

AWS CDK (TypeScript) での開発フロー 2/4 Code



IDE やテキストエディタを使用して CDK アプリをコーディング

cdk init で生成されるファイルの例

File Structure

```
.
├── README.md
├── bin
│   └── cdk-sample.ts
├── cdk.json
├── jest.config.js
├── lib
│   └── cdk-sample-stack.ts
├── node_modules
├── package-lock.json
├── package.json
├── test
│   └── cdk-sample.test.ts
└── tsconfig.json
```

コード例 (引数など一部省略)

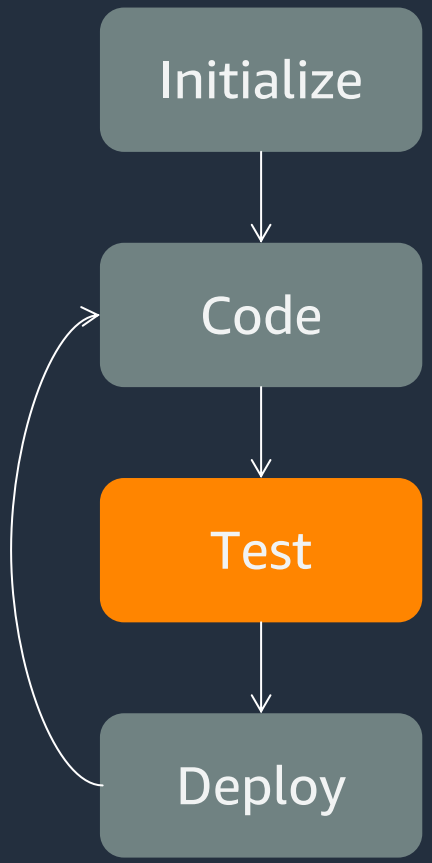
bin/cdk-sample.ts

```
// App を作成
const app = new cdk.App();
// App に Stack を追加
new CdkSampleStack(app, "MyApp");
```

lib/cdk-sample-stack.ts

```
// Stack を定義
export class CdkSampleStack extends Stack {
  constructor() {
    // コンストラクタ内で Construct を追加
    new s3.Bucket(this, "HogeBucket");
    new sqs.Queue(this, "HogeQueue");
  }
}
```

AWS CDK (TypeScript) での開発フロー 3/4 Test



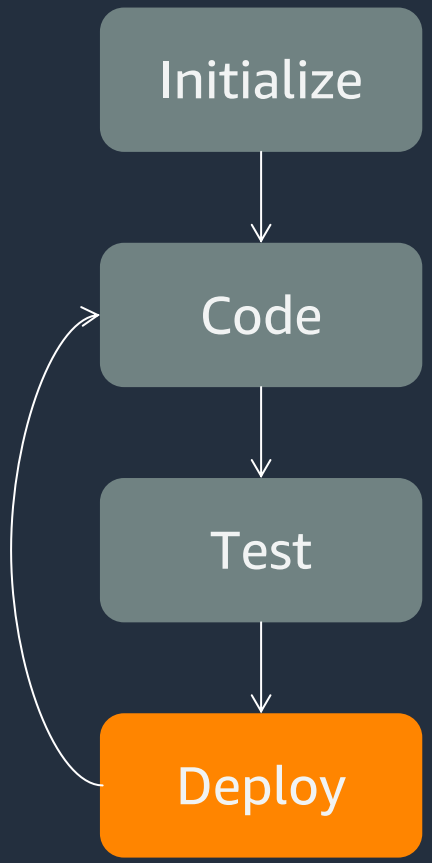
CDK アプリをテスト

Console

```
# CDK から CloudFormaiton template などを含む Cloud Assembly を合成
$ npx aws-cdk synth

# テスト
$ npm run test
```

Jest などでテストの実装が必要。実装の詳細は今後の Blackbelt で解説予定

AWS CDK (TypeScript) での開発フロー 4/4 Deploy



CDK アプリを AWS にデプロイ

```
Console
# デプロイ済みの CloudFormation Template との差分を確認
$ npx aws-cdk diff

# AWS に CDK アプリをデプロイ
$ npx aws-cdk deploy --all
```

* cdk deploy の代表的なオプション

```
Console
$ npx aws-cdk deploy SampleStack # スタックを指定してデプロイ
$ npx aws-cdk deploy --all --require-approval=never # デプロイ時の確認を行わない
$ npx aws-cdk deploy --all --hotswap # Lambda関数などの開発時に変更を高速に反映する
$ npx aws-cdk deploy --all --no-rollback # スタックの更新失敗時に自動ロールバックしない
$ npx aws-cdk deploy --all -c key=value # Context を指定
```

* 最新のコマンドリファレンスは GitHub を参照
https://github.com/aws/aws-cdk/blob/main/packages/aws-cdk/README.md

各言語における プロジェクト構成

AWS CDK in Python

前提条件や言語固有のイディオムなど、詳細はこちら
https://docs.aws.amazon.com/cdk/v2/guide/work-with-cdk-python.html

File Structure

```
.
├── .gitignore
├── .venv
├── README.md
├── app.py
├── cdk.json
├── cdk_sample
│   ├── __init__.py
│   └── cdk_sample_stack.py
├── node_modules
├── package-lock.json
├── package.json
├── requirements-dev.txt
├── requirements.txt
├── source.bat
└── tests
    ├── __init__.py
    └── unit
        ├── __init__.py
        └── test_cdk_sample_stack.py
```

Entry point

cdk.json

```
"app": "python3 app.py"
```

Stack

Console

```
# CDK プロジェクトを作成
$ npx aws-cdk init app --language=python
$ source .venv/bin/activate
$ pip install -r requirements.txt
# CDK Toolkitをローカルインストールしてバージョンを固定（推奨）
$ npm init -y
$ npm install -D aws-cdk
$ echo node_modules >> .gitignore
# CDK Bootstrapping
$ npx aws-cdk bootstrap
# デプロイ
$ npx aws-cdk deploy --all
```

Console

```
# CDK Toolkit のアップデート
$ npx npm-check-updates -u
$ npm install
# コンストラクトライブラリのアップデート
$ pip list -o | sed -e '1,2d' | cut -f1 -d' ' | xargs pip
install -U
```

(参考) Recommended AWS CDK project structure for Python applications | AWS Developer Tools Blog
https://aws.amazon.com/jp/blogs/developer/recommended-aws-cdk-project-structure-for-python-applications/

AWS CDK in Java

前提条件や言語固有のイディオムなど、詳細はこちら
https://docs.aws.amazon.com/cdk/v2/guide/work-with-cdk-java.html

File Structure

```
.
├── .gitignore
├── README.md
├── cdk.json
├── node_modules
├── package-lock.json
├── package.json
├── pom.xml
└── src
    ├── main/java/com/myorg
    │   ├── CdkSampleApp.java
    │   └── CdkSampleStack.java
    └── test/java/com/myorg
        └── CdkSampleTest.java
```

Stack

Entry point

cdk.json

```
"app": "mvn -e -q compile exec:java"
```

Console

```
# CDK プロジェクトを作成
$ npx aws-cdk init app --language=java
# CDK Toolkitをローカルインストールしてバージョンを固定（推奨）
$ npm init -y
$ npm install -D aws-cdk
$ echo node_modules >> .gitignore
# CDK Bootstrapping
$ npx aws-cdk bootstrap
# デプロイ
$ npx aws-cdk deploy --all
# （参考）コンパイルとテストを実行
$ mvn package
```

Console

```
# CDK Toolkit のアップデート
$ npx npm-check-updates -u
$ npm install
# コンストラクトライブラリのアップデート
$ mvn versions:use-latest-versions
```

AWS CDK in C#

前提条件や言語固有のイディオムなど、詳細はこちら
https://docs.aws.amazon.com/cdk/v2/guide/work-with-cdk-csharp.html

File Structure

```
.
├── .gitignore
├── README.md
├── cdk.json
├── node_modules
├── package-lock.json
├── package.json
└── src
    ├── CdkSample
    │   ├── CdkSample.csproj
    │   ├── CdkSampleStack.cs
    │   ├── GlobalSuppressions.cs
    │   └── Program.cs
    └── CdkSample.sln
```

Stack

Entry point

Console

```
# CDK プロジェクトを作成
$ npx aws-cdk init app --language=csharp
# CDK Toolkitをローカルインストールしてバージョンを固定（推奨）
$ npm init -y
$ npm install -D aws-cdk
$ echo node_modules >> .gitignore
# CDK Bootstrapping
$ npx aws-cdk bootstrap
# デプロイ
$ npx aws-cdk deploy --all
# （参考）コンパイルとテストを実行
$ dotnet build src
```

Console

```
# CDK Toolkit のアップデート
$ npx npm-check-updates -u
$ npm install
```

cdk.json

```
"app": "dotnet run --project src/CdkSample/CdkSample.csproj"
```

コンストラクトライブラリのアップデート方法はこちらを参照
https://docs.aws.amazon.com/cdk/v2/guide/work-with-cdk-csharp.html#csharp-managemodules

AWS CDK in Go

前提条件や言語固有のイディオムなど、詳細はこちら
https://docs.aws.amazon.com/cdk/v2/guide/work-with-cdk-go.html

File Structure

```
.
├── .gitignore
├── README.md
├── cdk-sample.go
├── cdk-sample_test.go
├── cdk.json
├── go.mod
├── go.sum
├── node_modules
├── package-lock.json
└── package.json
```

App & Stack

Entry point

cdk.json

```
"app": "go mod download && go run cdk-sample.go"
```

Console

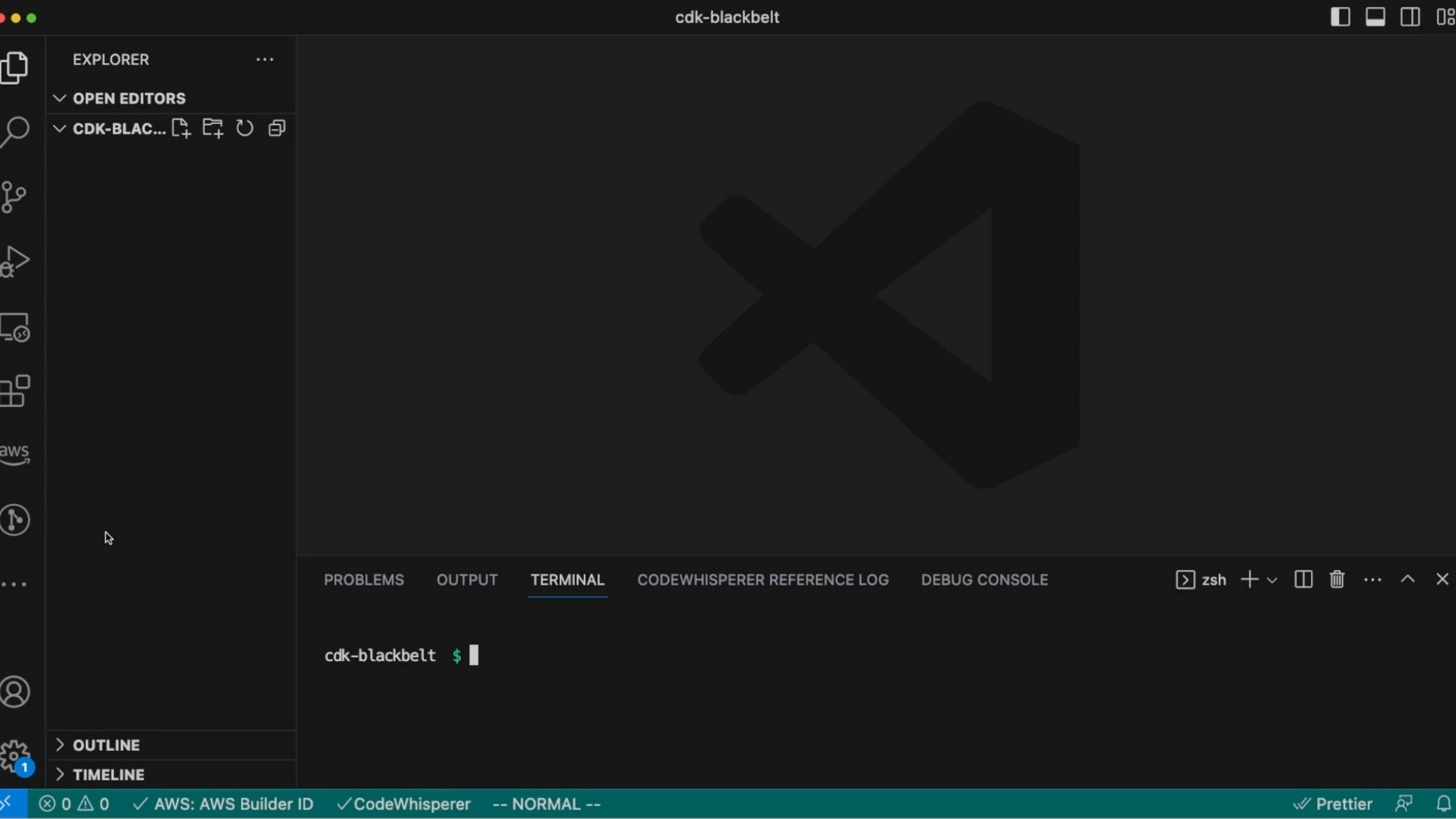
```
# CDK プロジェクトを作成
$ npx aws-cdk init app --language=go
# CDK Toolkitをローカルインストールしてバージョンを固定（推奨）
$ npm init -y
$ npm install -D aws-cdk
$ echo node_modules >> .gitignore
# CDK Bootstrapping
$ npx aws-cdk bootstrap
# パッケージのインストール
$ go get
# デプロイ
$ npx aws-cdk deploy --all
# （参考）コンパイルとテストを実行
$ go test
```

Console

```
# CDK Toolkit のアップデート
$ npx npm-check-updates -u
$ npm install
# コンストラクトライブラリのアップデート
$ go get -u
$ go mod tidy
```

デモ

TypeScript での AWS CDK プロジェクト作成 ～
Amazon VPC をデプロイ


cdk-blackbelt
EXPLORER
OPEN EDITORS
CDK-BLAC...
PROBLEMS
OUTPUT
TERMINAL
CODEWHISPERER REFERENCE LOG
DEBUG CONSOLE
zsh
cdk-blackbelt $
OUTLINE
TIMELINE
0 0
AWS: AWS Builder ID
CodeWhisperer
-- NORMAL --
Prettier

AWS CDK の学習リソース

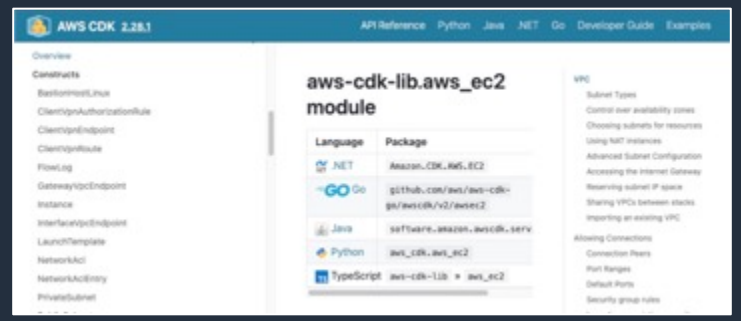
AWS CDK のドキュメント

AWS CDK Developer Guide

https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/home.html

AWS CDK のコンセプトや
実践的なベストプラクティスなど
開発に役立つ情報を記載

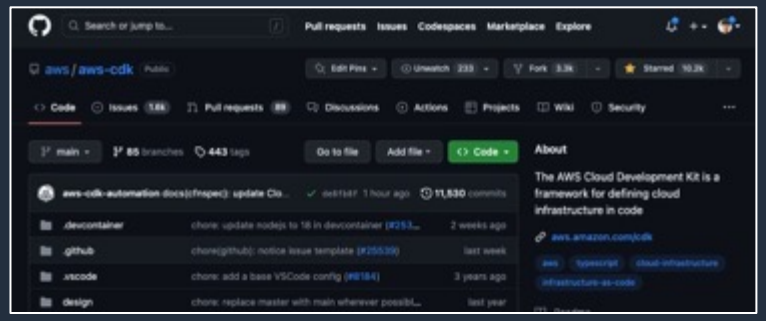
API Reference



https://docs.aws.amazon.com/cdk/api/v2/docs/aws-construct-library.html

API の仕様はこちらで確認

GitHub repository



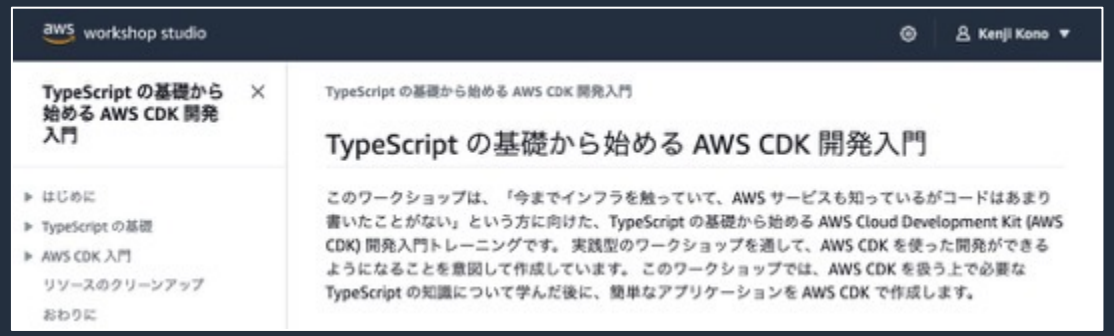
https://github.com/aws/aws-cdk

AWS CDK の開発リポジトリ

最新の開発状況や Design Doc
などを確認できる

AWS CDK のワークショップ (日本語)

TypeScript の基礎から始める AWS CDK 開発入門



https://catalog.workshops.aws/typescript-and-cdk-for-beginner/

あまりコードを書いたことがない方向けに
TypeScript の基礎から
CDK を学べるワークショップ

CDK Workshop

https://cdkworkshop.com/

実際にコードを書きながら
CDK を学べるワークショップ

TypeScript, Python, C#/.NET, Java, Go に対応

Thank you!

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 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 2023 年 7 月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)