AWS Black Belt Online Seminar

AWS Cloud Development Kit (CDK)

Basic #2

基本的なコンポーネントと機能

高野 賢司

Solutions Architect
2023/08

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. CDK App のライフサイクル
2. CDK App の構成要素
3. リソースとパラメータの参照
4. リソースのインポート

AWS CDK v2.92.0 対応

最新の情報は
AWS 公式ウェブサイト または
aws-cdk リポジトリにて
ご確認ください

CDK App のライフサイクル

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

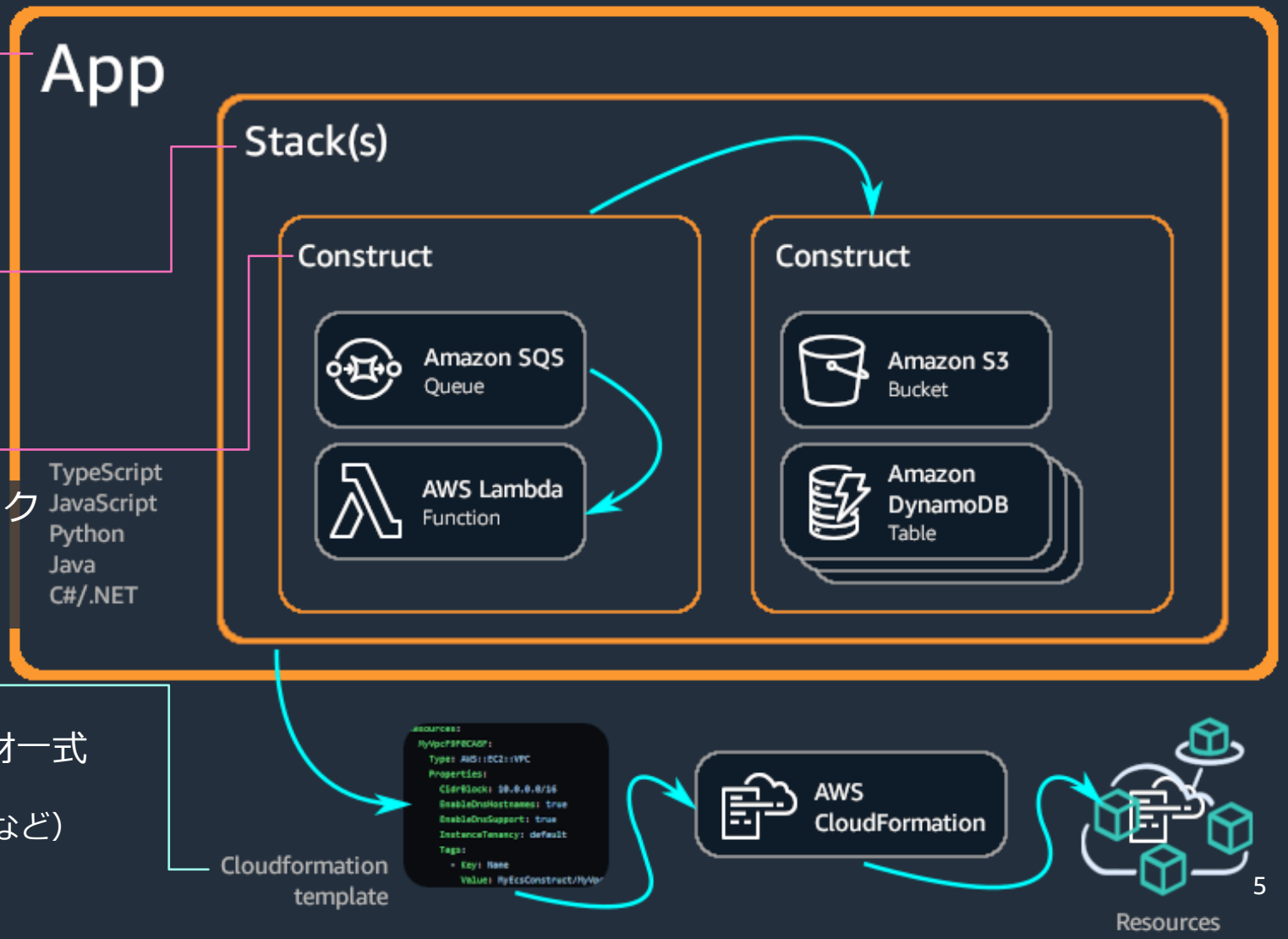
- CloudFormation スタックに対応
- デプロイ可能な最小単位

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

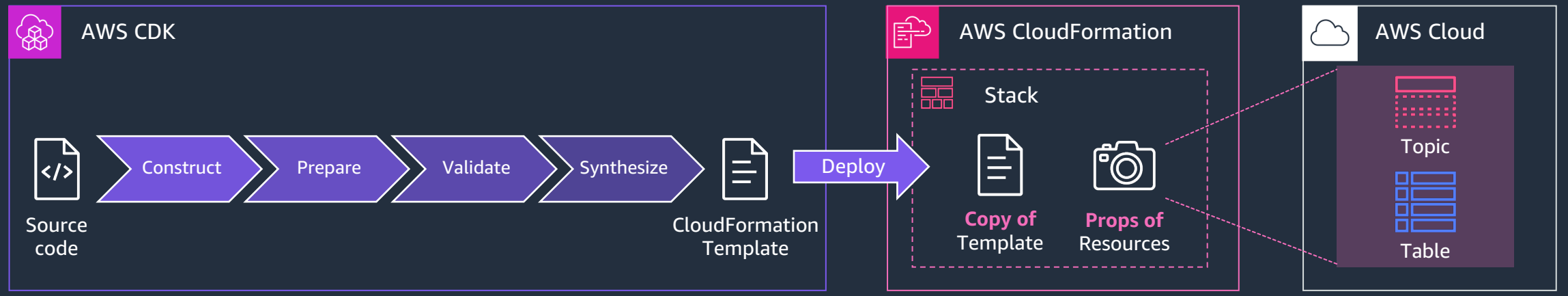
Cloud Assembly

- 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 と AWS CloudFormation の関わり

* Asset はここでは図示していません



プログラミングの世界

- リソースのあるべき状態を宣言する
- CloudFormation テンプレートに
  ロジックを含めることはできないため *
  未確定な値は CDK または別のプログラムで解決するか
  CloudFormation の機能で渡す必要がある

* ここでは任意の処理を組み込めるわけではないことを指す。
Fn::If や Fn::FindInMap などで単純なロジックは記述可能。

IaC サービスの世界

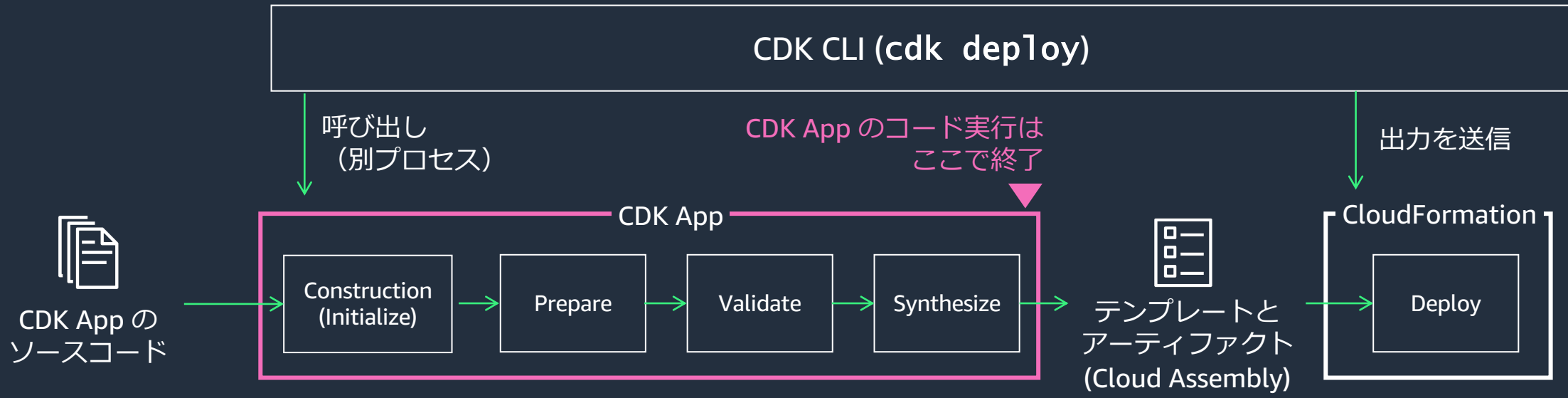
- ユーザーの指示を
  テンプレートとして
  受け取って API を呼び出し
- **スタックに永続化**
  - テンプレートのコピー
  - 管理しているリソースの設定
  - Export / Import された値

現実世界

- さまざまな手段で
  変更されうる
- CloudFormation で
  管理されていない
  ものもある

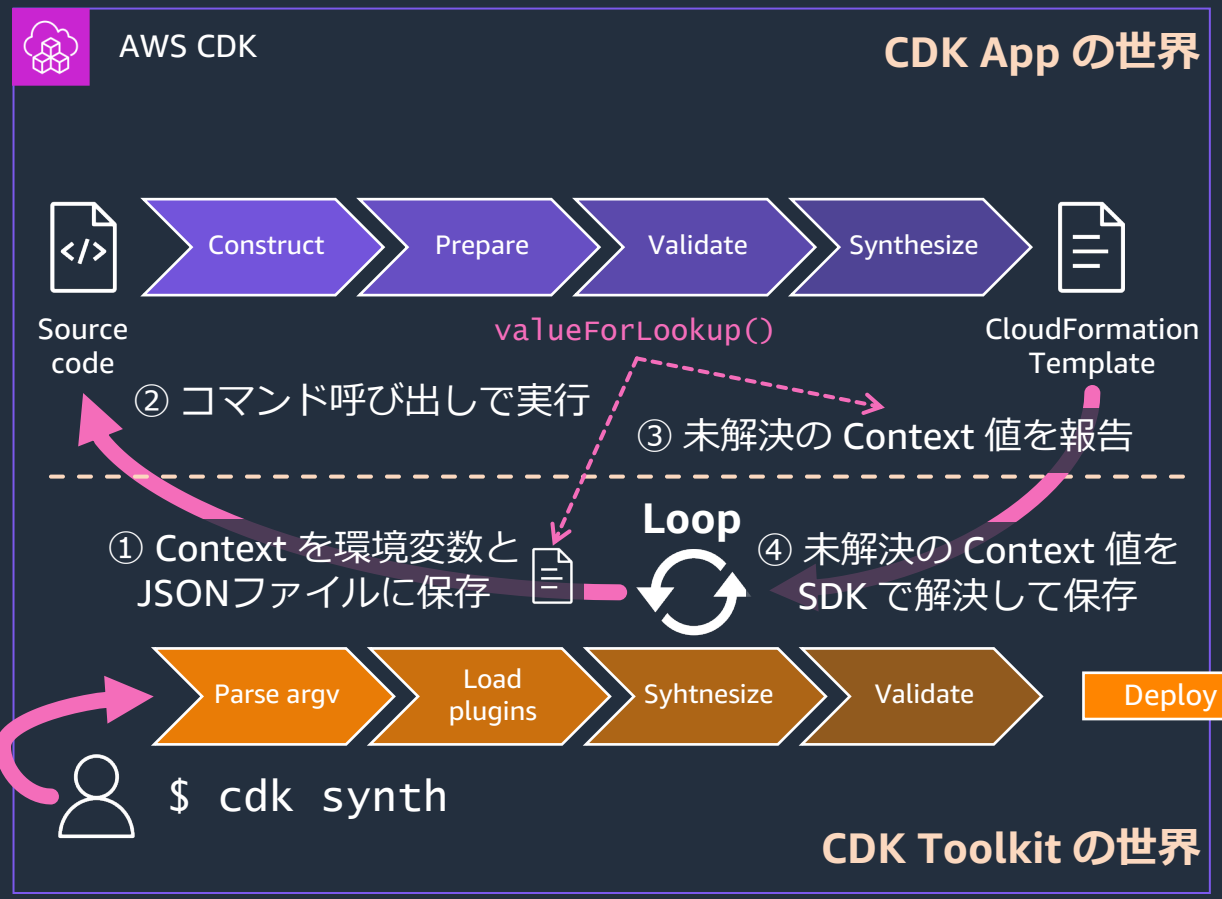
CDK App のライフサイクル

- CDK App は CDK CLI によって実行され、さらに実行は 4 つの内部フェーズに分かれている
- CDK App のコードは基本的に Construction フェーズで実行される
  - 後続フェーズで Validate される例：RestApi にメソッドが一つも含まれない場合にエラーにする
- CDK App の**コード実行が完了してから** CloudFormation や Asset のデプロイが行われる
  - コード上でデプロイ時に決定される値を受け取ったり、デプロイ中に処理を実行したりすることはできない
    （未決定の値を扱う Token や、Custom Resource, Trigger などの機能でカバー可能）



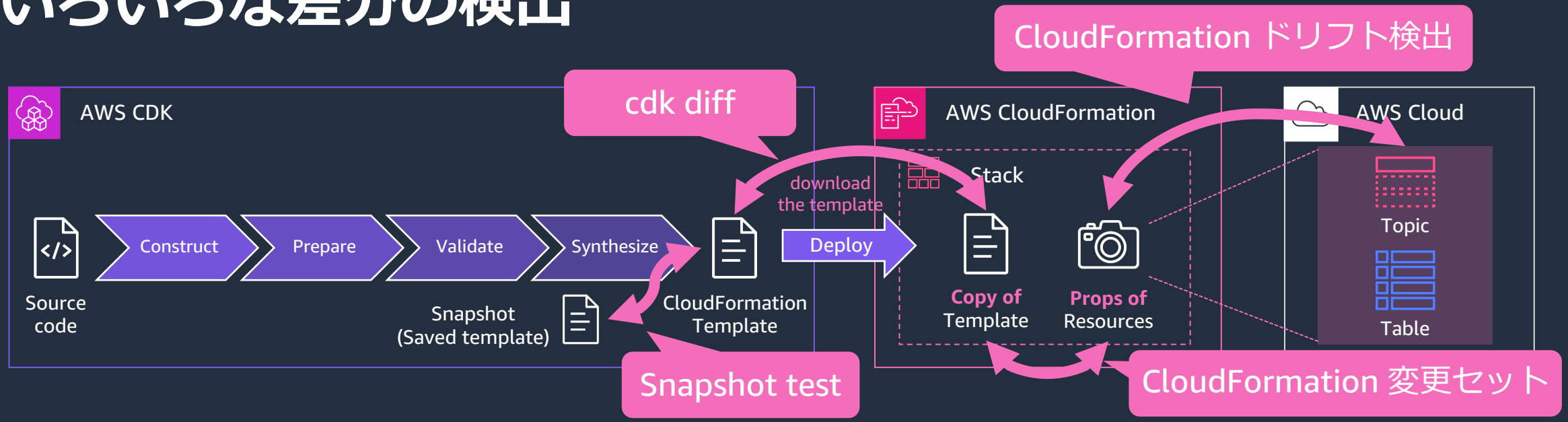
https://docs.aws.amazon.com/cdk/v2/guide/apps.html

Dive Deep: CDK App のライフサイクル



- CDK App の役割は CloudAssembly を作成すること
- CDK Toolkit の役割は CDK App のために Context などの周辺環境を準備したり diff の生成やデプロイなどの処理を行うこと
- **CDK App と CDK Toolkit は直接統合しない**
  - CDK App をコマンドで実行（別プロセス）
  - Context は環境変数や JSON で渡す（言語非依存）
- `valueFromLookup` などによって未解決の Context 値が参照されると **CDK App が複数回実行されることがある**
  - CDK App で SDK 呼び出しや副作用のある処理を書く場合パフォーマンス劣化や冪等性、権限などに注意

いろいろな差分の検出



cdk diff コマンド

- 合成したテンプレートと
  すでにデプロイ済みのテンプレートを比較
- リソースの置換が起こるかどうかも仕様から確認

CDK のスナップショットテスト

- 合成したテンプレートと
  以前のテンプレート（スナップショット）を比較

CloudFormation 変更セット

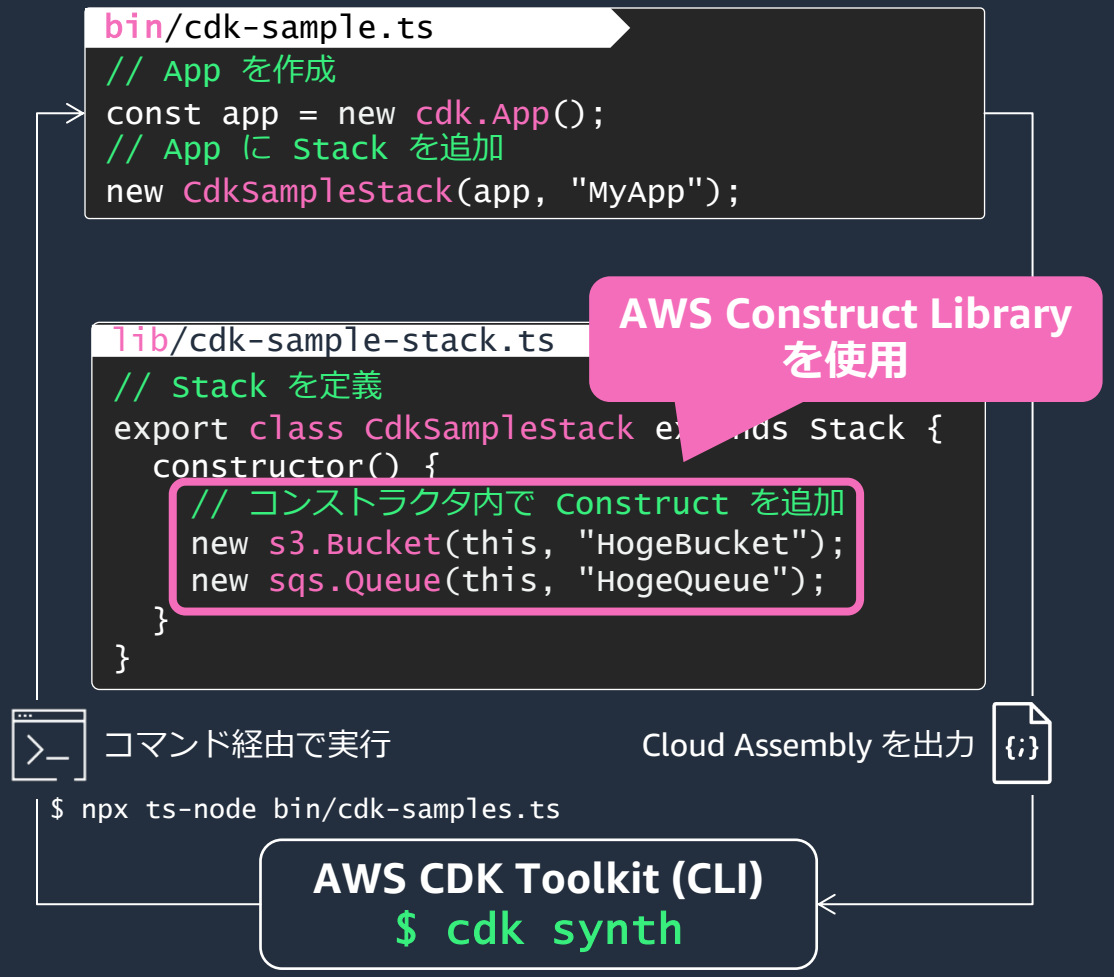
- 新しいテンプレートをアップロードし
  保存されているスタックリソースの状態と比較
- UpdateStack が呼ばれると以前の変更セットは削除

CloudFormation ドリフト検出

- 保存されているスタックリソースの状態と
  現在のリソースの状態を比較

CDK App の構成要素

Construct



- CDK におけるもっとも基本的な構成要素で クラウドコンポーネントを表現する
  - 単一のリソース (Amazon S3 Bucket など) だけでなく 複数のリソースを含むハイレベルな Construct を定義可能
  - App や Stack などのクラスも Construct を実装している
- AWS Construct Library が標準で提供されるほか ユーザーが独自の Construct を定義・配布可能
- Construct Programming Model (CPM) の一部
  - クラウドアプリケーションなどの望ましい状態を定義して 抽象化するためのプログラミングモデル
  - CDK v2 から Construct は独立したライブラリとなり CDKTF, CDK8s, Projen などの他のツールでも使用される

Construct のシグネチャ

lib/cdk-sample-stack.ts

```
new Table(this, 'MyTable', {
 partitionKey: { name: 'pk', type: AttributeType.STRING },
 billingMode: BillingMode.PAY_PER_REQUEST,
});
```

Construct の初期化時は
共通して 3つの引数を取る

位置	仮引数	型	説明
1	scope	Construct	Construct ツリーの親を指定。 通常は現在のスコープを表す this になる。 任意の Construct を渡す場合は ID の重複に注意。
2	id	string	**Construct ID** と呼ばれる。 scope 内で一意となる識別子を指定。 リソース名や論理 ID の一部になる。
3	props	(Construct ごとに固有の型)	Construct の初期状態を定義するプロパティ。 適切なデフォルト値が用意されており、 すべてのプロパティがオプションのことも。

複数の Construct をまとめる（コンポジション）

lib/construct/frontend.ts

Construct を継承

```
export class Frontend extends Construct {
 constructor(scope: Construct, id: string, props: FrontendProps) {
  super(scope, id);

  const webAcl = new wafv2.CfnWebACL(this, 'WebAcl');
  const webContentBucket = new s3.Bucket(this, 'WebContentBucket');
  const distro = new cf.Distribution(this, 'Distro', {
   defaultBehavior: {
    origin: new origins.S3Origin(webContentBucket),
   },
   webAclId: webAcl.attrArn,
  });
 }
}
```

※ 簡素化のため一部の必須プロパティを省略

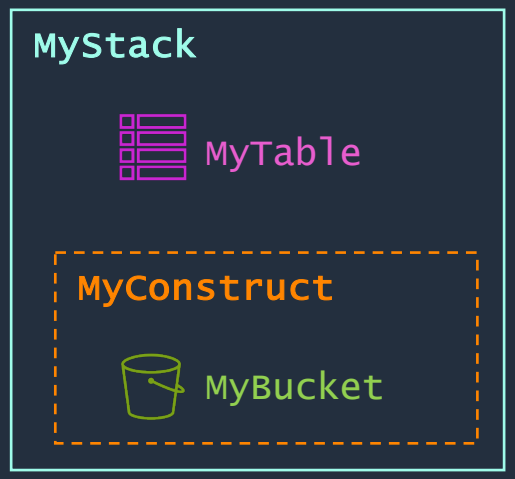
※ L2 Construct を新たに作成して AWS CDK にコントリビュートする場合は作成方法が異なる
https://github.com/aws/aws-cdk/blob/main/CONTRIBUTING.md
https://github.com/aws/aws-cdk/blob/main/docs/DESIGN_GUIDELINES.md

- Construct を継承したクラスを定義して複数の Construct をまとめる
- CDK でコンポーネントを構造化するもっとも一般的な方法
  - CloudFormation では Stack とテンプレートファイルが 1:1 対応だが、CDK では Stack を複数の Construct から構成可能。ファイルの分割も自由に行える
- 作成したハイレベル Construct をライブラリとして共有する場合は Projen の利用を推奨
  https://github.com/projen/projen/blob/main/docs/awscdk-construct.md

Construct ID と物理名

Construct ID … コンストラクトを初期化するときに第２引数に指定する文字列のこと

```
new Table(stack1, 'MyTable1', {
  partitionKey: { name: 'pk', type: AttributeType.STRING, },
  removalPolicy: RemovalPolicy.DESTROY,
});
```



CDK CLI 上の表示 （Stack の Construct ID）

```
$ npx aws-cdk ls
MyStack
```

AWS マネジメントコンソール上の表示 （物理名）

- MyStack
- MyStack-MyTableXXXXXXXX-XXXXXXXX
- mystack-myconstructmybucketxxxxxxxx-xxxxxxxx

- スタック名には Construct ID がそのまま使われる
- 物理名には <Stack>- が前置され、リソースまでの Construct のパスが結合して使われる

※ App クラスには Construct ID がない
※ リソースごとの名前の制約に合わせて変換されることも（小文字化、区切り文字の変更など）

- Stage で Stack をまとめるとさらに <Stage>- が前置される

Construct の使い分けとエスケープハッチ

Raw Resource

L1 が存在しないときにのみ使用。
プロパティに型がないため
指定には注意が必要

L1 Construct

L2 にないリソースやプロパティ、
Override, Depencency の指定が
必要な場合に使用

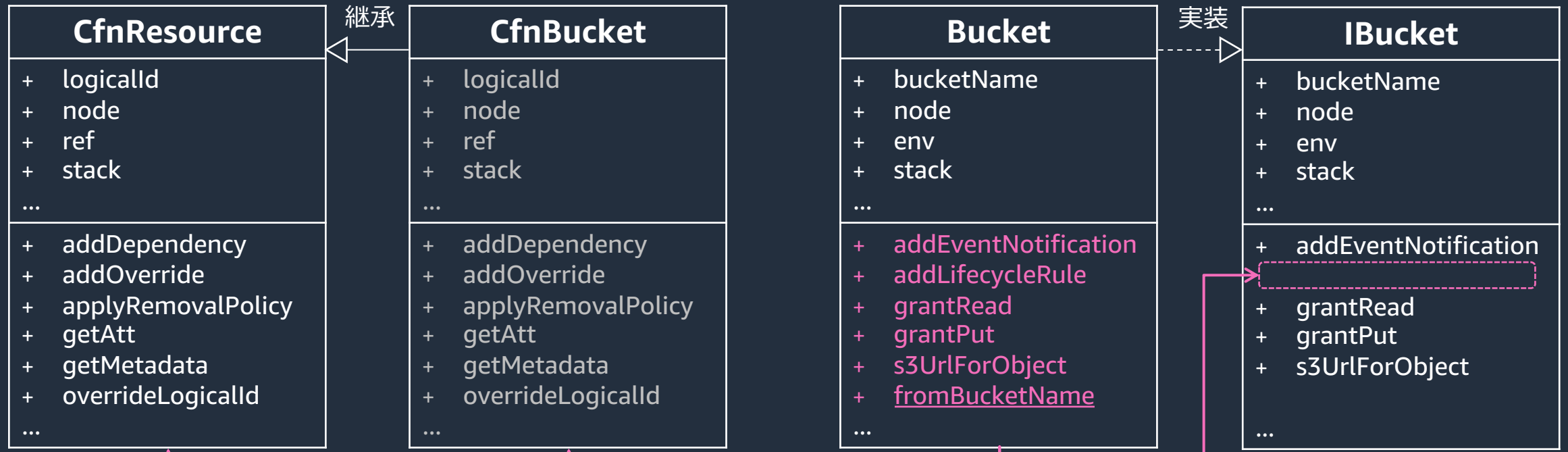
L2 Construct

1st Choice

優先的に使用。
便利なメソッドや
デフォルト値をもつ

Interface

fromXXX メソッドで取得できる
外部リソースや
読み取り専用のリソースを表現



node.defaultChild で取得可能 (エスケープハッチ)

リソースを変更するメソッドは存在しない

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

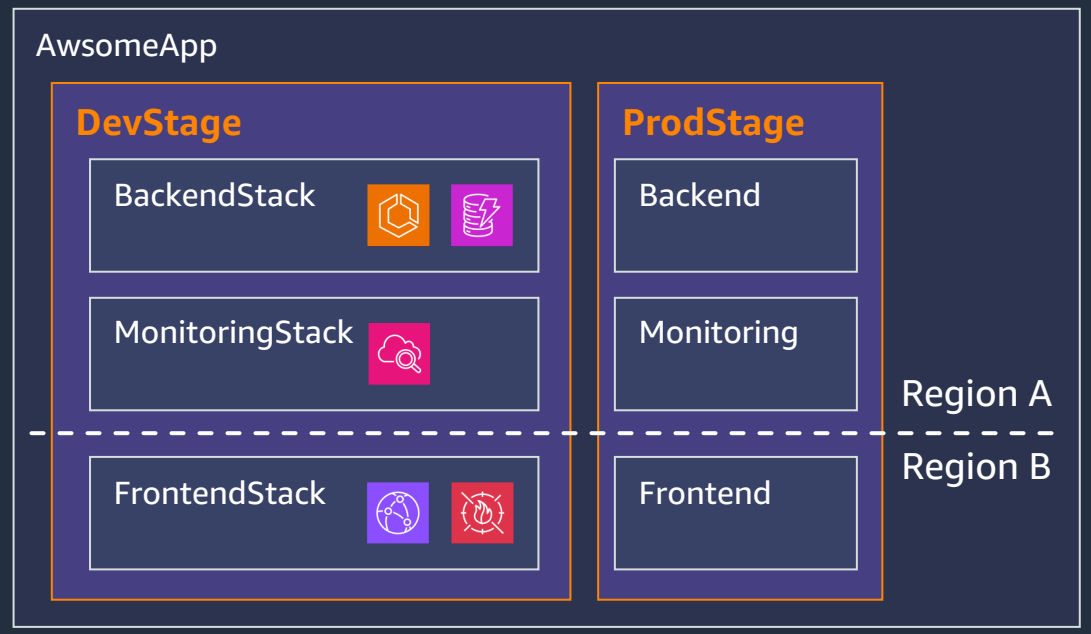
App

- アプリケーション全体を表現する Construct ツリーの根
- 複数のAWSアカウント、リージョンをまたいで Stack を一元管理できる
- Stack の依存関係からデプロイ順序を自動で管理
- App は CDK CLI からコマンド経由で実行され Cloud Assembly を作成する
  - cdk.json の app プロパティまたは CDK CLI の --app オプションで指定（上書き）
- App の終了後に CDK CLI によって Cloud Assembly のデプロイが行われる
  - デプロイ中のプロセスには直接干渉できない

Stack

- CloudFormation スタックに対応し、CloudFormation の制約を受ける
  https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-limits.html
- CDK は CloudFormation を使用して AWS リソースをプロビジョニングするため AWS リソースは Stack に含まれる必要がある
- Stack ごとに AWSアカウント、リージョンを指定可能
- tags プロパティを指定すると Stack 自身と配下のタグ付け可能なリソースすべてにタグを付けられる

Stage



- 複数の Stack をまとめて論理的なアプリ (サービス) のかたまりを表現
- 1つの Stage を定義して環境ごとにインスタンス化することで複製が可能
- CDK Pipelines 使用時には必須

Console

```
# deploy --all では Stage 配下は対象にならないため ** を指定
$ cdk deploy '**'
# 特定の Stage に絞ってデプロイ
$ cdk deploy 'Dev/*'
```

bin/cdk-sample.ts

```
// App を作成
const app = new cdk.App();
// App に Stage を追加
new AwsomeAppStage(app, "Dev");
new AwsomeAppStage(app, "Prod");
```

lib/stage.ts

```
export class AwsomeAppStage extends Stage {
 constructor(scope: Construct, id: string, props: AppParameter) {
  super(scope, id, props);

  const backend = new BackendStack(this, 'Backend');
  const frontend = new FrontendStack(this, 'Frontend');
  const monitoring = new MonitoringStack(this, 'Monitoring');
```

Environment

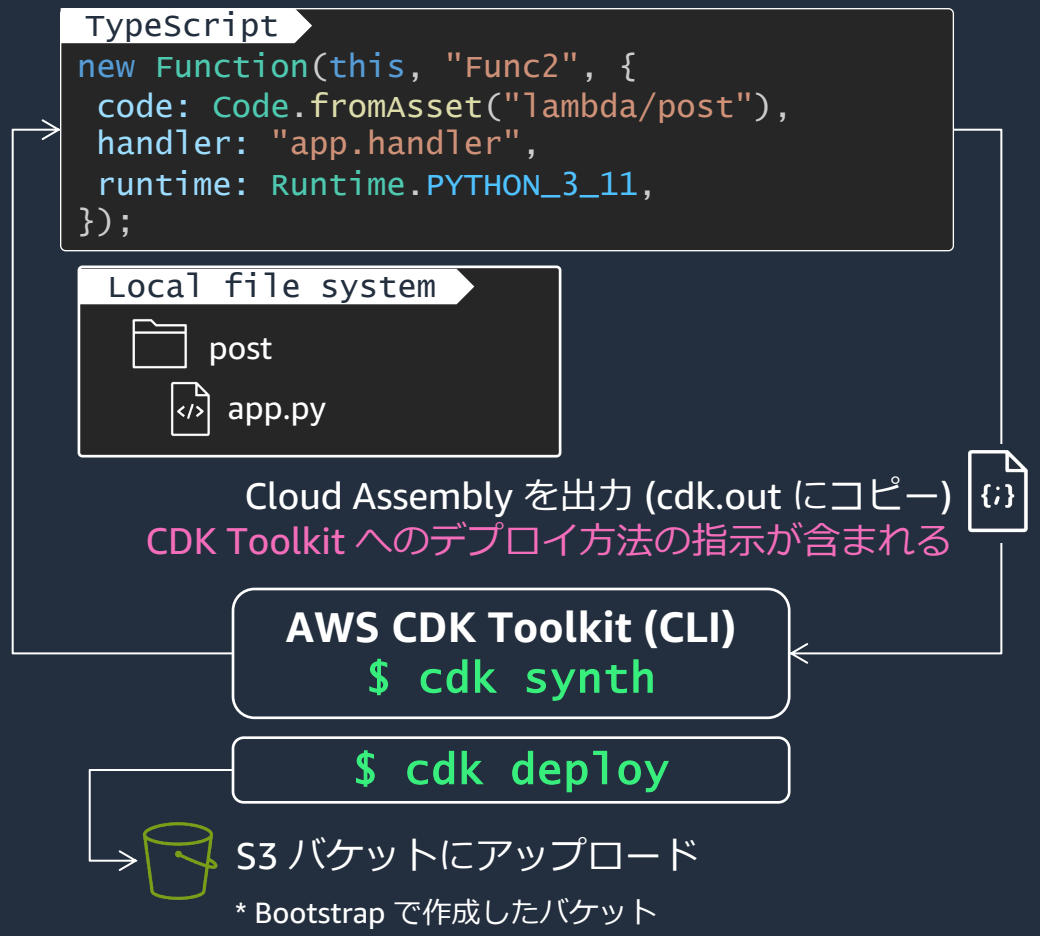
bin/cdk-sample.ts

```
new Stack(app, 'LocalStack', {
 env: {
  account: process.env.CDK_DEFAULT_ACCOUNT,
  region: process.env.CDK_DEFAULT_REGION,
 },
});

new Stage(app, 'VirginiaStage', {
 env: {
  account: '123456789012',
  region: 'us-east-1',
 },
});
```

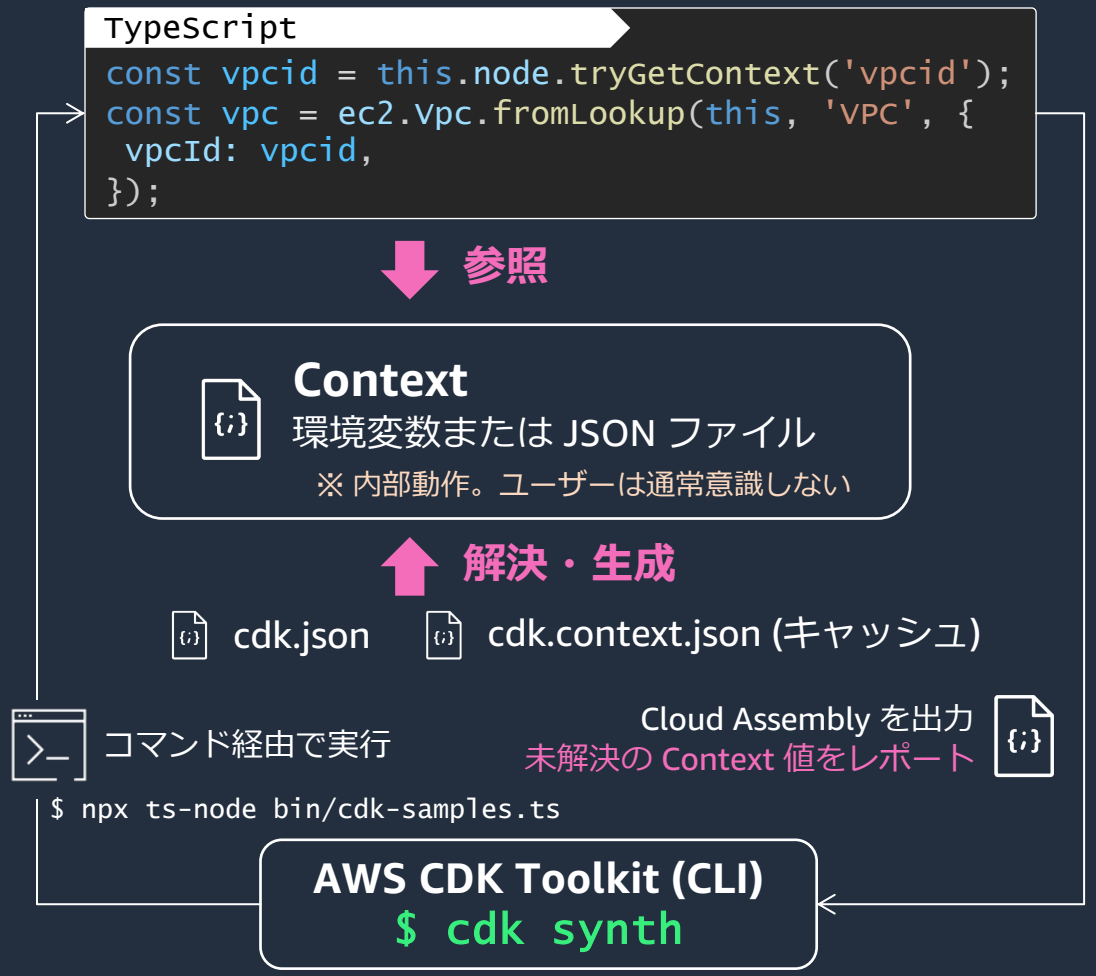
- Stack や Stage を初期化するときに
  デプロイ先の AWS アカウントとリージョンを指定
- env, account, region いずれも指定はオプション
  - 指定を省略すると、環境に依存しないスタックとみなされる
    (environment-agnostic)
  - この場合 Vpc.fromLookup や stack.region などの
    AWS 環境を特定する必要のあるコードは機能しない
- 環境変数 CDK_DEFAULT_ACCOUNT と
  CDK_DEFAULT_REGION が CDK CLI によって提供
  - CDK CLI 実行時の環境変数や AWS CLI Profile から解決
  - 外部からこの環境変数を与えることはできない
  - 指定を省略した場合とは異なり、環境に依存するコードは
    動作するが、CDK CLI 実行時の環境に影響されるため注意

Asset



- CDK App と一緒にデプロイできるファイルやディレクトリ、Docker イメージのこと
- zip 圧縮 または docker build と アップロードは CDK CLI によって行われる
- バンドル時に任意のビルドコマンドを実行可能
  - 例: フロントエンドアプリを事前にビルド
  - 参考: aws_s3_assets.AssetOptions https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_s3_assets.AssetOptions.html

Context



- CDK App から参照できる Key-Value ペア
  - `node.tryGetContext()` で参照可能
- cdk.json の context プロパティと CDK CLI の -c, --context で任意の値を渡せる
  - シンプルな Key-Value のため使い勝手は限定的
- CDK CLI は CDK App の合成中に未解決の値がレポートされると、AWS SDK を使用して解決。解決した値は cdk.context.json にキャッシュ
  - 最新の Amazon Linux 2 AMI ID や Availability Zone ID 等環境や実行タイミングによって変化しうる値を保持して決定論的なデプロイを行えるようにする
  - このため cdk.context.json は手で編集せず Git にコミットすることを推奨
  - キャッシュの削除は cdk context --reset コマンド

Feature Flag (機能フラグ)

- 後方互換性を保ちながら CDK を進化させたり、
  特定のクラスやメソッドの動作を変更したりできるようにするメカニズム
- cdk.json にコンテキスト値として指定（cdk context --reset ではリセットされない）
- cdk init で新しいプロジェクトを作成した場合、
  推奨されるすべての機能フラグを有効にした cdk.json ファイルを生成する

Currently recommended cdk.json

The following json shows the current recommended set of flags, as `cdk init` would generate it for new projects.

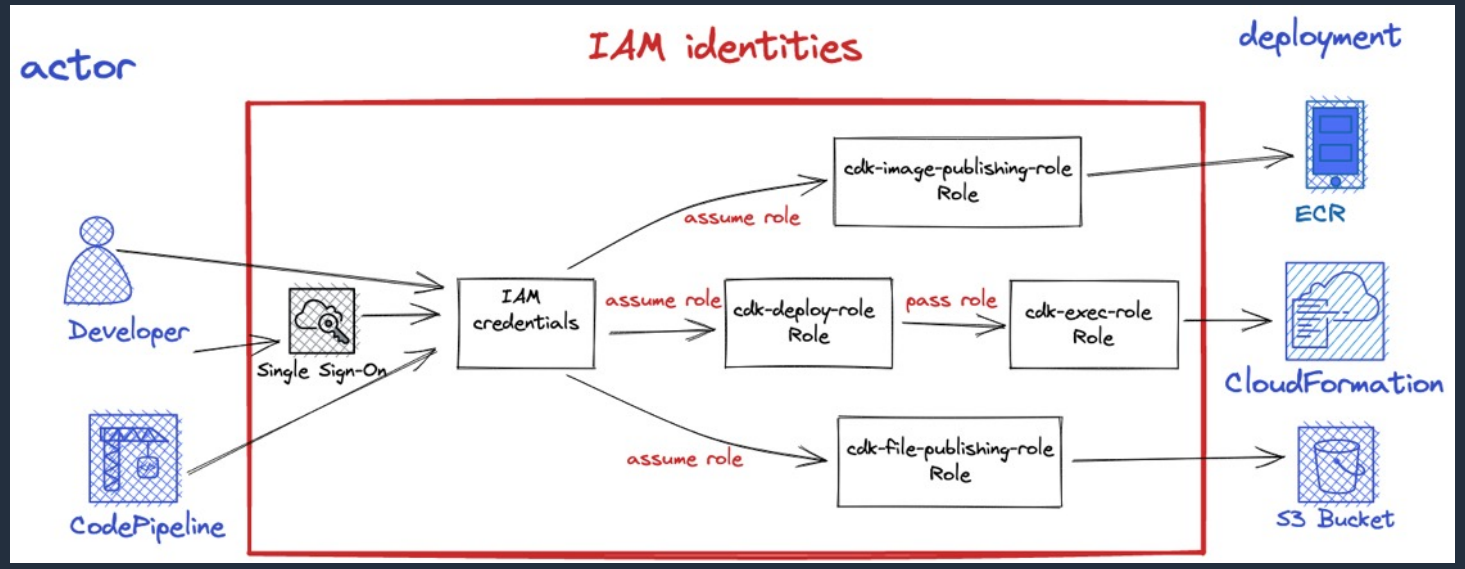
```
{
  "context": {
    "@aws-cdk/aws-lambda:recognizeLayerVersion": true,
    "@aws-cdk/core:checkSecretUsage": true,
    "@aws-cdk/core:target-partitions": [
      "aws",
      "aws-cn"
    ],
    "@aws-cdk-containers/ecs-service-extensions:enableDefaultLogDriver": true,
    "@aws-cdk/aws-ec2:uniqueImdsv2TemplateName": true,
    "@aws-cdk/aws-ecs:arnFormatIncludesClusterName": true,
```

最新の機能フラグの一覧と
推奨される機能フラグは GitHub で確認
https://github.com/aws/aws-cdk/blob/main/packages/%40aws-cdk/cx-api/FEATURE_FLAGS.md

Bootstrapping

CDK App をデプロイする前に、ファイルを保存するための S3 バケットや ECR リポジトリ、
CDK Toolkit や CloudFormation が Assume する IAM Role などを作成するプロセス

```
Console
# 現在の AWS CLI の認証情報と環境変数を使用して bootstrap
$ cdk bootstrap
# AWS アカウントとリージョンを指定して bootstrap
$ cdk bootstrap aws://123456789012/us-east-1
```

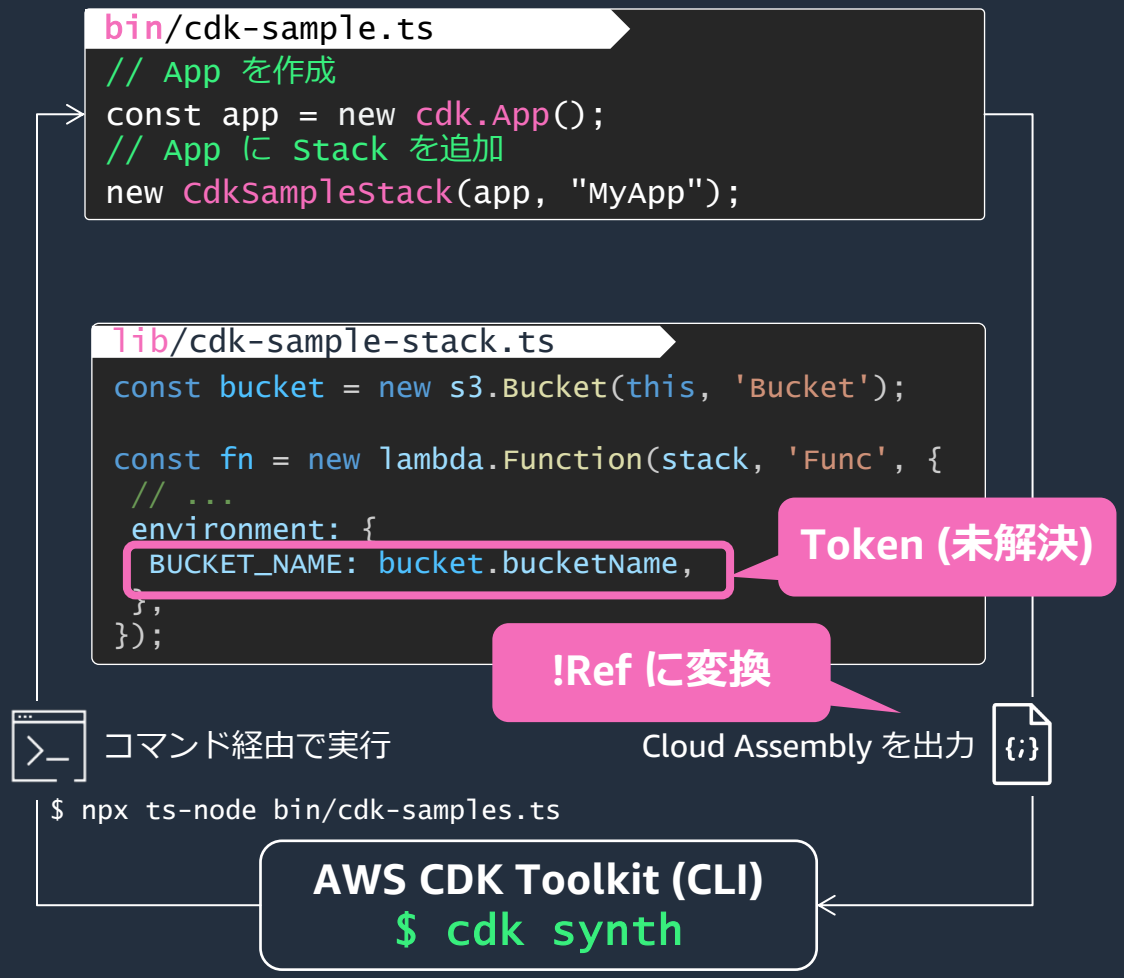


AWS アカウントとリージョンごとに
Bootstrap が必要

CDK Pipelines を使用する場合など
別のアカウントへのデプロイを許可するには
--trust オプションが使用可能
https://docs.aws.amazon.com/cdk/v2/guide/cdk_pipeline.html

Bootstrap 時に作成されるリソースや
Permissons Boundary のカスタマイズは
こちらを参照
https://github.com/aws/aws-cdk/wiki/Security-And-Safety-Dev-Guide

Token



- CDK App で未解決の値を扱うためのクラス
- 合成中の後続のコードで決定される値
  - `Lazy.string` など
  - `produce` 関数で値を返す実装を行う必要がある
- デプロイ時に決定される値
  - Amazon S3 バケット名などのプロパティは Token を返す
  - 合成後に CloudFormation の `Ref` 関数に変換される
- Token を参照するとプレースホルダが返る
  `${TOKEN[Bucket.Name.1234]}`
  - 他のリソースに渡せるが、文字列結合以外の演算は不可

Aspect

- 指定されたスコープのすべての Construct への操作を実装可能
- タグをつけたり、削除ポリシーを設定する他に、暗号化の有無などのチェックにも使用可能
- CDK App の合成中、Prepare フェーズで `visit` 関数が呼び出される
- Stage をまたいだ適用はできない

```typescript
// If Destroy Policy Aspect is present:
if (props?.setDestroyPolicyToAllResources) {
 cdk.Aspects.of(this).add(new ApplyDestroyPolicyAspect());
}
```

コード例：
すべてのリソースに `RemovalPolicy.DESTROY` を設定

https://github.com/aws-samples/cdk-integ-tests-sample/blob/0a9f9baa0c90a623242d1158d918ebf5a08f566d/lib/cdk-integ-tests-demo-stack.ts#L139

```typescript
/**
 * Aspect for setting all removal policies to DESTROY
 */
class ApplyDestroyPolicyAspect implements cdk.IAspect {
 public visit(node: IConstruct): void {
  if (node instanceof CfnResource) {
   node.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY);
  }
 }
}
```

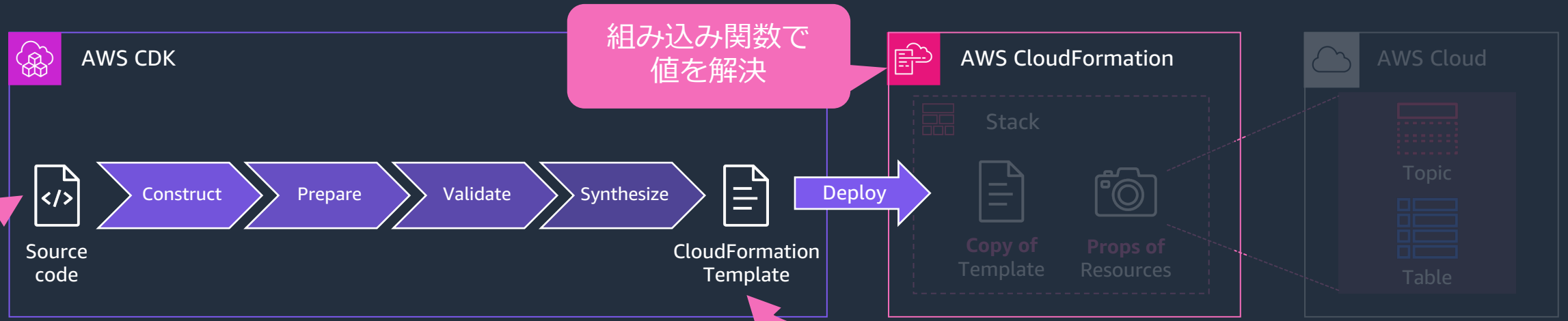
Tag をつける

- 指定されたスコープのすべての Construct に一括でタグをつけることができる
- `Tags.of( SCOPE ).add() / Tagas.of ( SCOPE ).remove()`
- タグの操作が競合した場合、 `priority` が高いものが採用される
  - デフォルト: 追加: 100, CloudFormation リソースへの直接追加: 50, 削除: 200
- Aspect を使用しているため、Stage をまたいだ適用はできない
- タグ付け可能なリソースにのみ付与されるため、ユーザーがリソースを検査する必要はない

TypeScript
```typescript
// タグを付与
Tags.of(myConstruct).add('key', 'value');
// 優先順位を指定してタグを付与
Tags.of(myConstruct).add('key', 'value', { priority: 300 });
// タグを削除
Tags.of(myConstruct).remove('key');
```

リソースとパラメータの参照

CDK App 内のリソースの参照



TypeScript

```typescript
const taskRole = new iam.Role(this, 'EcsTaskRole', {
  assumedBy: new iam.ServicePrincipal('ecs-tasks.amazonaws.com'),
});

const taskDefinition = new ecs.FargateTaskDefinition(this,
'TaskDefinition', {
  executionRole: taskExecutionRole,
  taskRole: taskRole,
  cpu: 256,
  memoryLimitMiB: 512,
});
```

Template

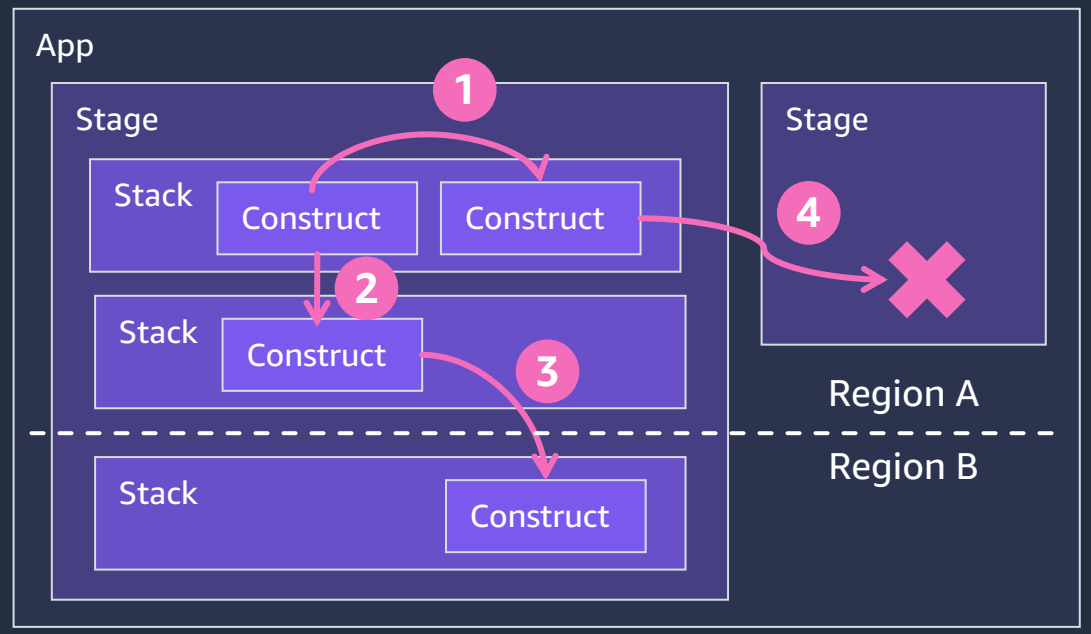
```
"TaskRoleArn": {
  "Fn::GetAtt": [
    "EcsAppEcsTaskRole2B17F49D",
    "Arn"
]}
```

CDK アプリのコード内でリソースを参照すると
CloudFormation の組み込み関数に変換される

※ CDK App 内で値にアクセスすると Token が返る

https://github.com/aws-samples/baseline-environment-on-aws/blob/f10bcf1167977976b8129483b2e26c46aef4006e/usecases/blea-guest-ecs-app-sample/lib/construct/ecsapp.ts#L197

参照するリソースの場所に応じてテンプレートを合成



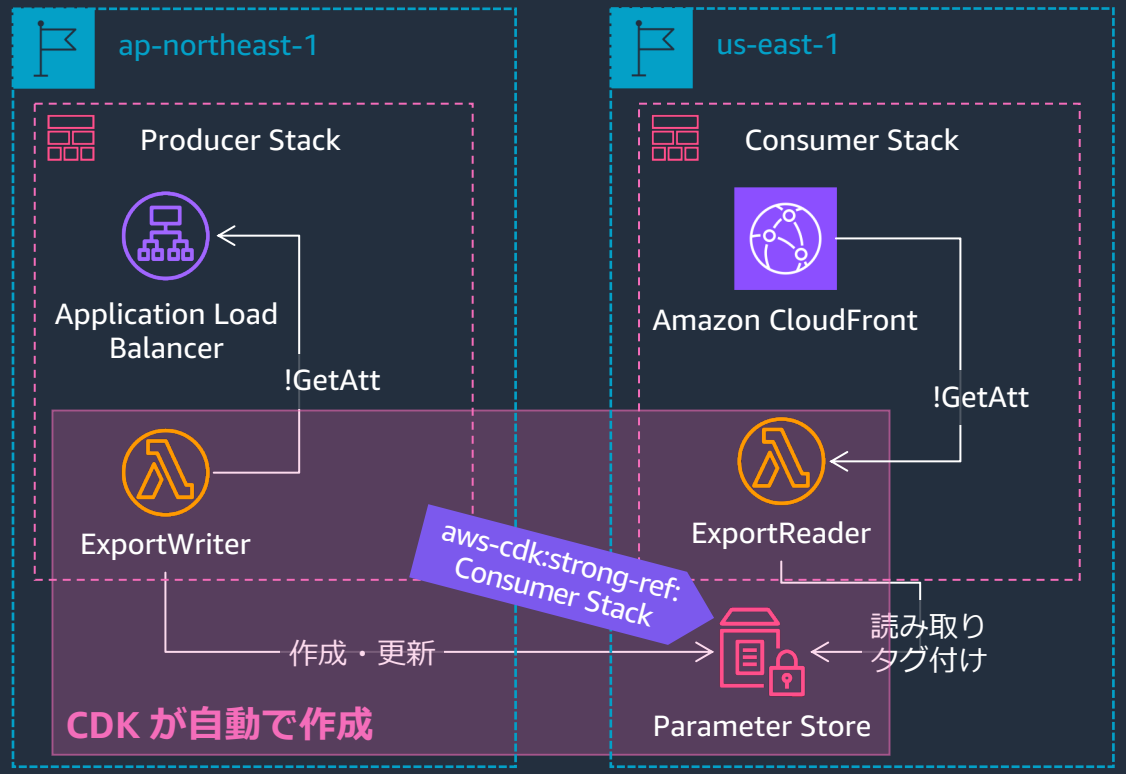
AWS CDK は参照するリソースの場所によって
CloudFormation テンプレートの出力を調整する

1. **同じ Stack 内のリソース**
   → `!Ref` または `!GetAtt`
2. **別の Stack 内のリソース**
   → `Export` と `!ImportValue` のセット
3. **別のリージョンのリソース**
   → `Export` と Custom Resource によって
   消費側のリージョンに SSM Parameter を作成し
   CloudFormation の `!GetAtt` で取得
   https://github.com/aws/aws-cdk/blob/main/packages/aws-cdk-lib/core/adr/cross-region-stack-references.md
4. **別の Stack 内のリソース**
   → 参照不可（Synth 時にエラーになる）

AWS CDK のクロスリージョン参照 (Preview)

v2.50.0 ~

CloudFront のオリジンとして他のリージョンの ALB を参照する例



CDK が自動的に Custom Resource と
SSM Parameter Store を作成。
クロススタック参照と同様に強い参照になる

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib-readme.html#accessing-resources-in-a-different-stack-and-region

TypeScript

```typescript
const ecsapp = new BLEAEcsAppStack(app,
 'Dev-BLEAEcsApp', {
  env: { region: 'ap-northeast-1' },
  crossRegionReferences: true
});

const frontend = new
BLEAEcsAppFrontendStack(app,
 'Dev-BLEAEcsAppFrontend', {
  env: { region: 'us-east-1' },
  crossRegionReferences: true
  alb: ecsapp.alb
});
```

提供側と消費側
両方に指定が必要
※ v2.79.0 現在

BLEA のサンプルコード https://github.com/aws-samples/baseline-environment-on-aws/blob/main/usecases/blea-guest-ecs-app-sample/lib/construct/frontend.ts

パラメータ・シークレット参照の使い分け

シークレットの値を使いたい

- SSM Parameter Store (Secure String)
- Secrets Manager

```TypeScript
const secure1 = SecretValue.ssmSecure('CdkConfSecret1');
const secure2 = SecretValue.secretsManager('CdkConfSecret2');
```

平文のパラメータの値を使いたい

- SSM Parameter Store

```TypeScript
const param1 = StringParameter.valueForStringParameter(stack, P1');
```

Synth 時に値を解決して埋め込みたい

`Vpc.ipAddresses` など Validation フェーズでエラーになる場合は以下を参照
https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_ssm-readme.html#lookup-existing-parameters

```TypeScript
const param2 = StringParameter.valueFromLookup(stack, P2');
```

※ 解決したコンテキスト値と
テンプレートに埋め込まれたパラメータは保存されるためセキュリティに注意

パラメータ・シークレットのリソースを参照したい

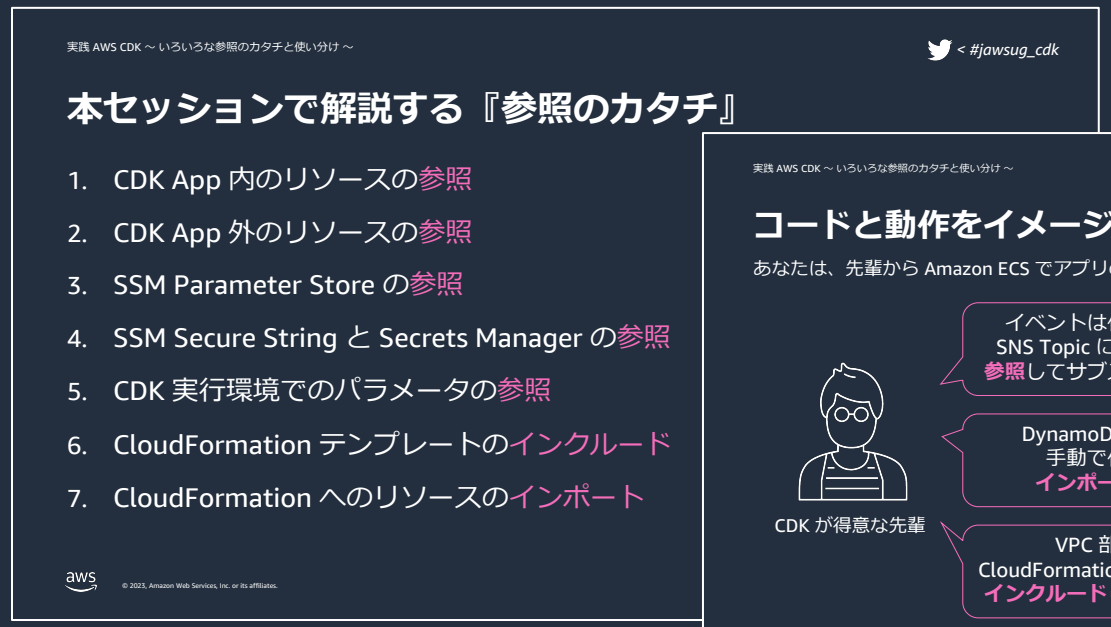
- `grant()` するとき
- リソースから値を受け取りたいとき
- RDS の `DatabaseSecret` など Construct を使うとき
  https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_rds.DatabaseSecret.html など

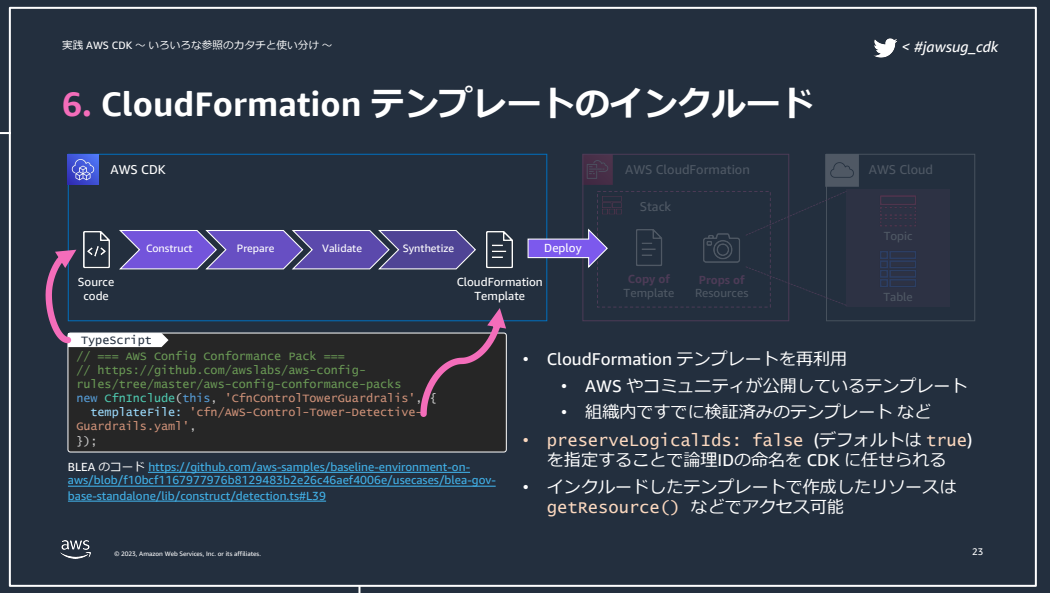
```TypeScript
const param1ref = StringParameter.fromStringParameterName(stack, 'P1Ref', 'P1');
Tags.of(bucket).add('Param1', param1ref.stringValue);
```

参考：CDK におけるリソースやパラメータ参照の詳細

実践 AWS CDK ～いろいろな参照のカタチと使い分け～

- 資料 https://speakerdeck.com/konokenj/reference-patterns-in-aws-cdk
- 動画 https://www.youtube.com/watch?v=FqsUe3nV_R0&t=2990s





参考：環境ごとにテンプレートを出し分ける方法

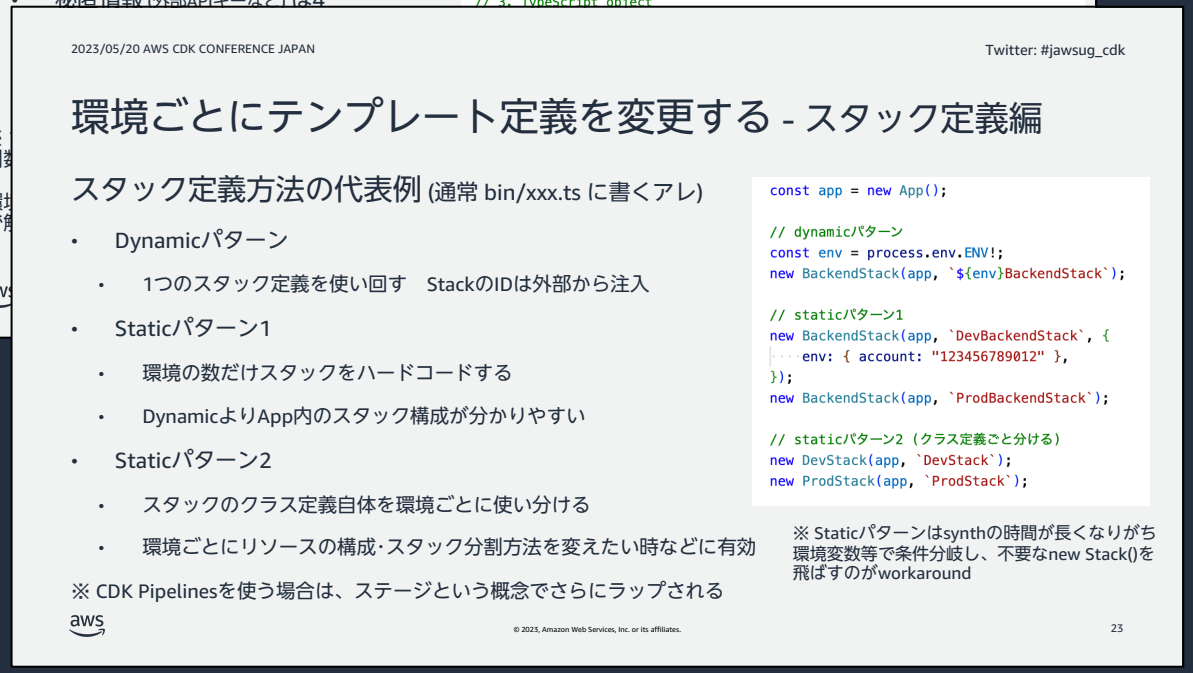
CDK の外部で決定される値を渡したり、
環境ごとに異なる設定を行ったりしたい場合
2つのことを考える必要がある

1. 外部のパラメータを読み込む方法
2. スタックを定義する方法

CDK は一般のプログラミング言語で
記述されるため方法は一つではなく
それぞれにトレードオフが存在する

『AWS CDKのあるあるお悩みに答えたい』

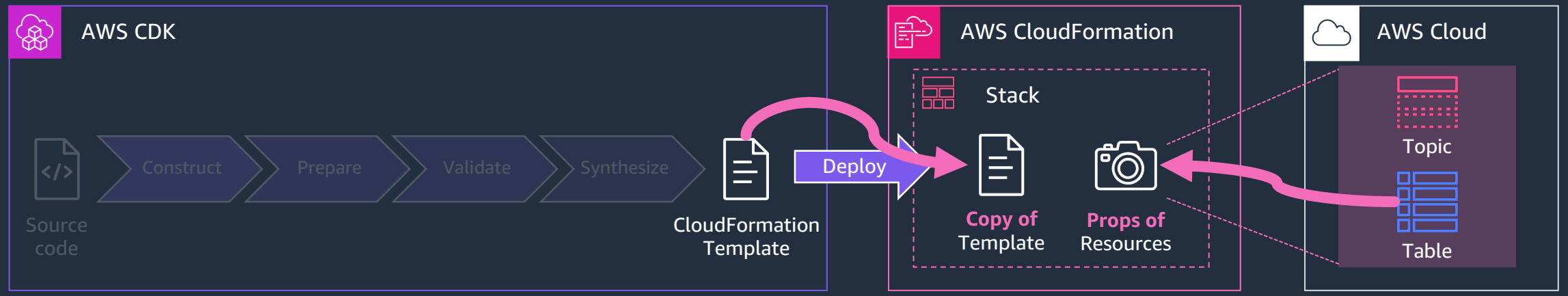
- 資料 https://speakerdeck.com/tmokmss/answering-cdk-faqs
- 動画 https://www.youtube.com/watch?v=FqsUe3nV_R0&t=18499s



リソースのインポート

CloudFormation へのリソースのインポート

v2.20.0 ~ (Preview)



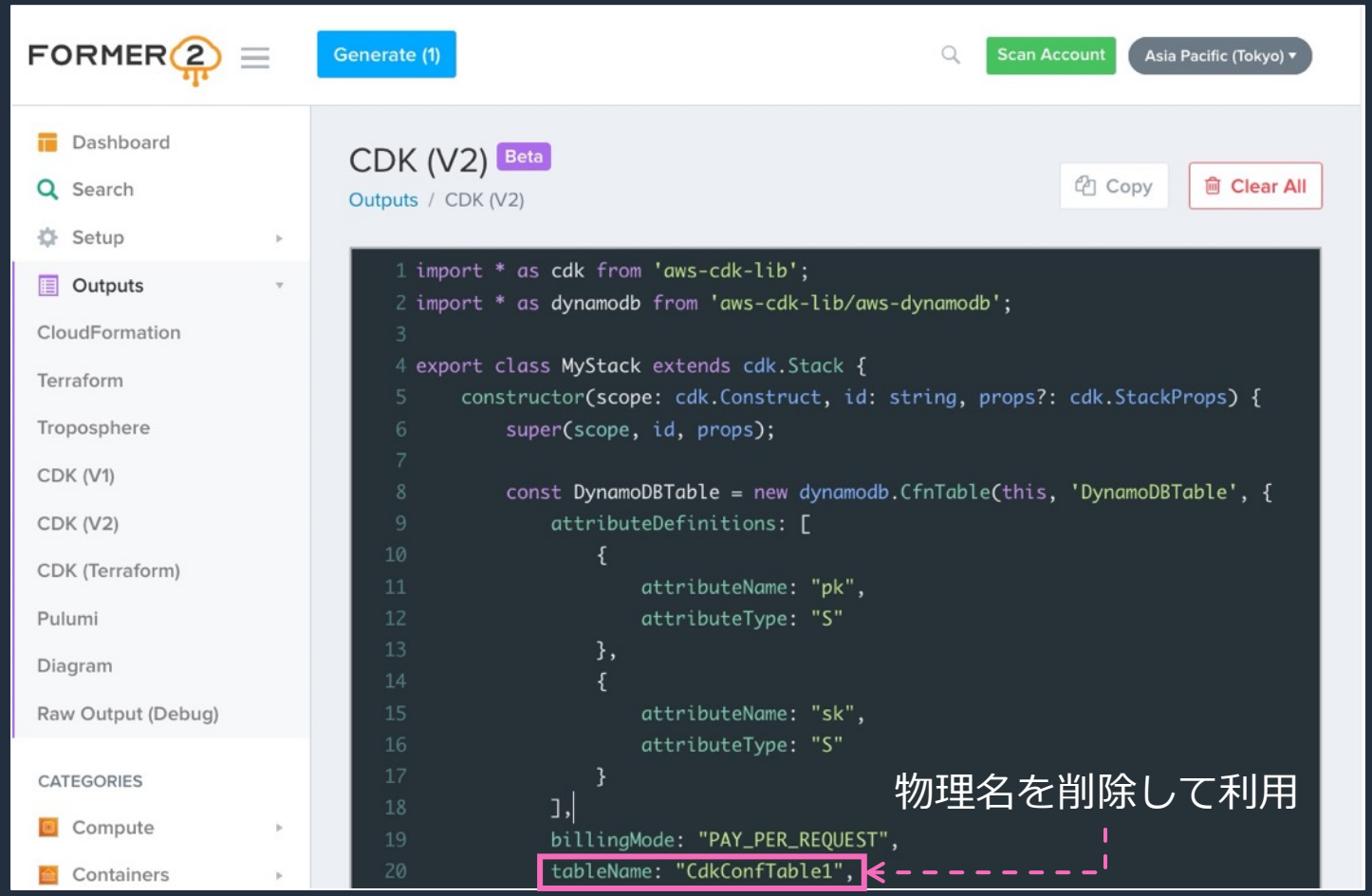

```
$ npx cdk import
The 'cdk import' feature is currently in preview.
CdkV2200Stack
CdkV2200Stack/SampleBucket/Resource (AWS::S3::Bucket): enter BucketName to impor
t (empty to skip): 3          cdk-import-sample.ap-northeast-1
CdkV2200Stack: importing resources into stack...
CdkV2200Stack: creating CloudFormation changeset...

 ✅ CdkV2200Stack
Import operation complete. We recommend you run a drift detection operation to c
onfirm your CDK app resource definitions are up-to-date. Read more here: https:/
/docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/detect-drift-stack.html
```

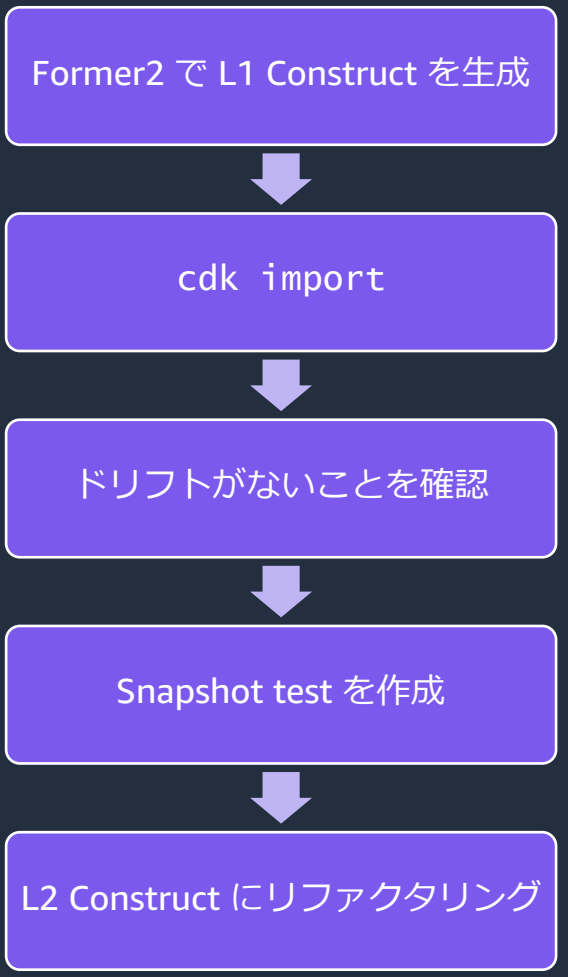
- 前提条件： テンプレートとリソースの設定はセット（テンプレートなしでインポートはできない）
- **現在のリソースの状態と一致するテンプレート**を用意してインポートを実施（コンソール or CLI）
- `cdk import` コマンドで対話的にインポート可能（Preview）

現在のリソースの状態とテンプレートが一致しなくても
インポートが成功するため、ドリフト検出が推奨されている

CDK でのリソースインポートの手順例

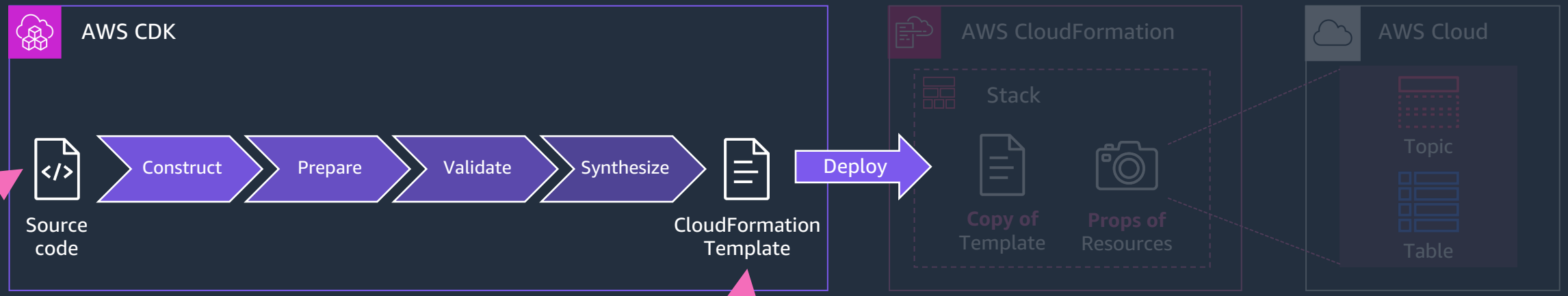


Former2 https://github.com/iann0036/former2



CloudFormation 資産の活用

CloudFormation テンプレートのインクルード

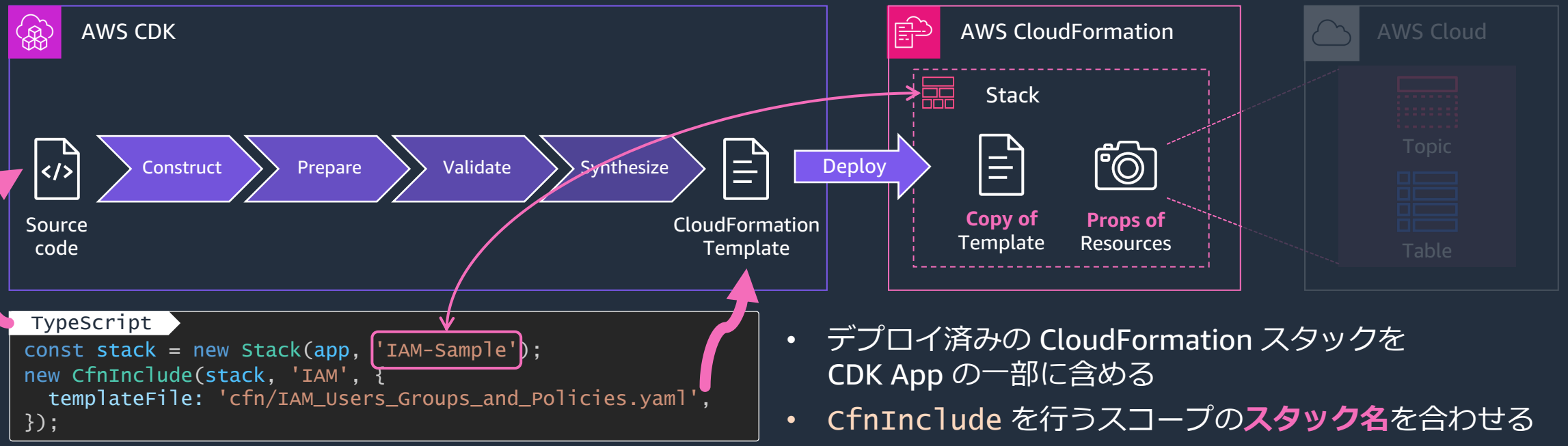



```typescript
// === AWS Config Conformance Pack ===
// https://github.com/awslabs/aws-config-rules/tree/master/aws-config-conformance-packs
new CfnInclude(this, 'CfnControlTowerGuardralis', {
  templateFile: 'cfn/AWS-Control-Tower-Detective-Guardrails.yaml',
});
```

BLEA のコード https://github.com/aws-samples/baseline-environment-on-aws/blob/f10bcf1167977976b8129483b2e26c46aef4006e/usecases/blea-gov-base-standalone/lib/construct/detection.ts#L39

- CloudFormation テンプレートを再利用
  - AWS やコミュニティが公開しているテンプレート
  - チーム内ですでに検証済みのテンプレート など
- `preserveLogicalIds: false` (デフォルトは `true`) を指定することで論理IDの命名を CDK に任せられる
- インクルードしたテンプレートで作成したリソースは `getResource()` などでアクセス可能

CloudFormation スタックを CDK に移行

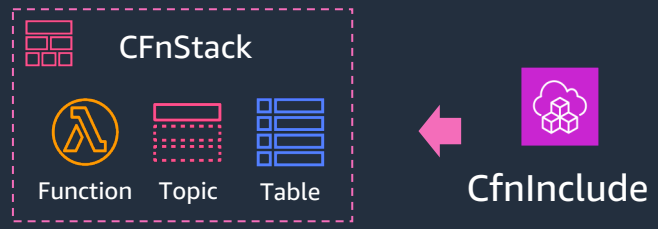


例）IAM-Sample スタックがデプロイ済みの場合

- デプロイ済みの CloudFormation スタックを CDK App の一部に含める
- `CfnInclude` を行うスコープの**スタック名**を合わせる
  - 通常は Construct ID (`new Stack()` の第 2 引数) を既存のスタック名に合わせる
  - Stage 配下の場合は `stackName` プロパティを明示的に指定
- `cdk diff` で Bootstrap 関連のパラメータや条件のみが追加されていることを確認

CloudFormation から CDK への移行方法の使い分け

デプロイ済みのスタックを維持して移行



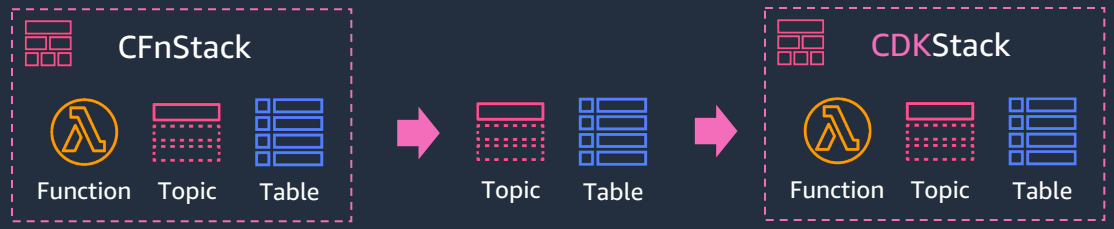
移行手順

1. `CfnInclude` で既存のスタック名と合わせてデプロイ
2. Snapshot test を作成・実行

ポイント

- デプロイ済みのスタックを変更しないため
  稼働中のアプリに影響を与えづらい
- スタック構成の変更や L2 Construct へのリファクタ
  などを段階的に行える
- リファクタ時にステートフルなリソースの論理 ID が
  変更される場合 `overrideLogicalId()` で維持

デプロイ済みのスタックを削除して移行



移行手順

1. CloudFormation で `DeletionPolicy: Retain` を設定
2. CloudFormation でスタックを削除
3. CDK でスタックを作成（他のリソースを先に作成）
4. `cdk import`

ポイント

- ステートフルなリソースのみインポート対象にする
  * インポート可能なリソースの一覧
  https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/resource-import-supported-resources.html
- 既存リソースの一部だけを利用したい場合に有効

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 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 2023 年 8 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)