AWS Black Belt Online Seminar

AWS Cloud Development Kit (CDK)

Basic #3

開発を効率化する機能

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. アプリのデプロイ
2. 複雑な設定の抽象化
3. デモ ... AWS Lambda 関数のホットスワップ
4. デプロイ中に任意の処理を実行する
5. テストとバリデーション
6. CDK Pipelines
7. Tips: npm

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

アプリのデプロイ

Asset でアプリケーションをデプロイ

Amazon S3 - aws_s3_deployment モジュール

- ローカル zip ファイル
- ローカルディレクトリ
- テキスト / JSON データ (デプロイ時の値を利用可能)

```typescript
new BucketDeployment(this, 'WebAppDeploy', {
  destinationBucket: s3Bucket!,
  distribution: cloudFrontWebDistribution, // invalidation
  sources: [
    // Deploy a React frontend app
    Source.asset('webapp/dashboard/build'),
  ],
});
```

AWS Lambda - aws_lambda モジュール

- ローカルのソースコードをバンドル
- aws_lambda_nodejs モジュールの場合は esbuild で TypeScript をトランスパイル&バンドル

```typescript
new NodejsFunction(this, 'Handler', {
  entry: 'lambda/metrics-handler.ts',
  handler: 'get',
});
```

Amazon ECS - aws_ecs モジュール

- ローカルの Dockerfile から コンテナイメージを作成し Amazon ECR リポジトリに push してデプロイ

```typescript
const taskDefinition = new FargateTaskDefinition(this, 'Def',
{
  cpu: 256,
  memoryLimitMiB: 512,
});
taskDefinition.addContainer('Recorder', {
  image: ContainerImage.fromAsset('container/recorder'),
});
```

AWS Lambda 関数の Construct

NodejsFunction *1

```
new NodejsFunction(this, "Func1", {
 entry: "lambda/item/index.ts",
 handler: "get",
 runtime: Runtime.NODEJS_18_X,
});
```

- esbuild の設定を抽象化し
  自動的に TypeScript / JavaScript
  をバンドル（L2.5 Construct）
- Tree shaking でサイズを削減
- バンドル後に AssetHash *2 を
  計算する
- package-lock.json を自動的に探索。
  package.json が指定ディレクトリ
  にない場合は内部で生成する

Function

```
new Function(this, "Func2", {
 code: Code.fromAsset("lambda/post"),
 handler: "app.handler",
 runtime: Runtime.PYTHON_3_11,
});
```

- 指定ディレクトリを zip して
  Asset として S3 にアップロード
- bundling オプションで
  任意のビルドコマンドを実行可能
  - 例：ローカルでバンドル実行を試行
- バンドル前に AssetHash *2 を計算。
  自分で計算することも可能
  - 例：node_modules を除外して計算

DockerImageFunction

```
new DockerImageFunction(this, "Func3", {
 code: DockerImageCode.fromImageAsset(
  "container",
  { cmd: ["index.put"] }
 ),
});
```

- Docker イメージをビルドして
  Asset として ECR にプッシュ
- Dockerfile でビルド設定を行う
  - レイヤーキャッシュでビルドを高速化
- オプションで Docker の設定や
  除外するファイルなどが指定可能

*1 ... 同様に PythonFunction, GoFunction も存在する (現時点では alpha)
*2 ... **AssetHash** ... Asset の同一性を識別するためのハッシュ値。
同一の Asset があればバンドルやアップロードをスキップする。
ソースまたはバンドル後の出力ファイルから計算される。

Amazon S3 Asset のオプション

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_s3_assets.AssetOptions.html

- bundling オプションで Docker コンテナ内で実行するコマンドを指定可能（例：esbuild）
  - ローカルでのコマンド実行を試みる設定も可能
- ディレクトリの内容と bundling オプションのハッシュを計算し同一ならバンドルをスキップ
  - AssetHash を自分で計算して指定することも可能

TypeScript

```typescript
new Function(this, "GetUserHandler", {
 code: Code.fromAsset("lambda/user", {
  bundling: {
   image: Runtime.NODEJS_18_X.bundlingImage,
   command: ["bash", "-c", "cp -au . /asset-output"],
  },
 }),
 handler: "index.get",
 runtime: Runtime.NODEJS_18_X,
});
```

※ NodejsFunction は
バンドル方法を自動設定するため
オプション体系が異なる

Amazon ECR Asset のオプション

- docker build コマンドに渡すオプションが指定可能
- Docker のビルドコンテキストにコピーしたくない ファイルは .dockerignore ファイルでも指定可能
- Docker のマルチステージビルドと レイヤーキャッシュを使用してビルドを高速化

TypeScript

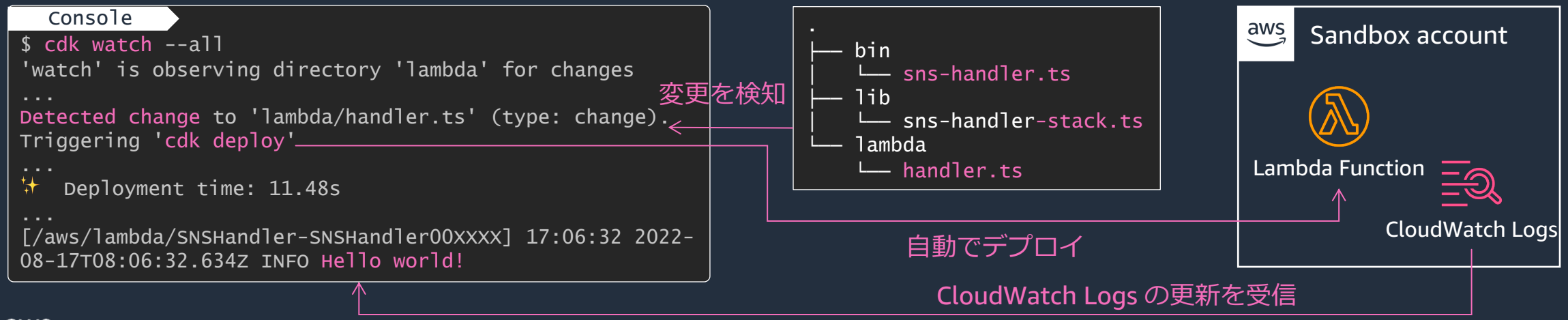
```typescript
new DockerImageFunction(this, 'DockerFunc', {
 code: DockerImageCode.fromImageAsset('container', {
  platform: Platform.LINUX_ARM64, // Arm64向けのイメージをビルド
  exclude: ['node_modules'],
  cmd: ['index.handler'],
 }),
});
```

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_ecr_assets.DockerImageAssetOptions.html

コードの変更をすばやくクラウドに反映

cdk deploy --hotswap / cdk watch

- Lambda 関数、ECS コンテナ、Step Functions ステートマシンなどを CloudFormation を経由せず API で直接デプロイすることで素早く反映
  https://github.com/aws/aws-cdk/blob/main/packages/aws-cdk/README.md#hotswap-deployments-for-faster-development
- CloudFormation スタックとドリフトが発生するため、開発用途でのみ使用
- cdk watch 実行中はスタックに含まれる CloudWatch Logs の更新を自動的に受信 (tail)
- ローカルマシンにエミュレーター等を導入しなくても、クラウドで直接開発できる



複雑な設定の抽象化

grant | 権限設定を抽象化

```typescript
const bucket = new Bucket(this, 'Bucket');

const f = new NodejsFunction(this, 'Func', {
  entry: 'lambda/list-object.ts',
  handler: 'handler',
  runtime: Runtime.NODEJS_18_X,
  environment: {
    BUCKET_NAME: bucket.bucketName,
  },
});
bucket.grantRead(f);
```

- Amazon S3 Bucket や Amazon DynamoDB Table などアクセス可能なリソースの L2 Construct には grant で始まるメソッドが実装されている
- 複雑な IAM ポリシーを設計することなく簡単に最小権限に設定できる
- IGrantable を実装しているリソースを指定可能
  - IAM User / Role / Group / Policy など
  - Lambda Function, EC2 Instance などの一部のリソース

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_iam.IGrantable.html

metric | CloudWatch メトリクスをリソースから取得

```typescript
// Alarm for ALB - HTTP 5XX Count
alb.metrics
  .httpCodeElb(elbv2.HttpCodeElb.ELB_5XX_COUNT, {
    period: cdk.Duration.minutes(1),
    statistic: cw.Stats.SUM,
  })
  .createAlarm(this, 'AlbHttp5xx', {
    evaluationPeriods: 3,
    threshold: 10,
    comparisonOperator:
cw.ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRES
HOLD,
    actionsEnabled: true,
  })
.addAlarmAction(new
cw_actions.SnsAction(props.alarmTopic));
```

https://github.com/aws-samples/baseline-environment-on-aws/blob/0b9e6f744667ce021eb54c1ad54e73523a9a811c/usecases/blea-guest-ecs-app-sample/lib/construct/ecsapp.ts#L402

- 多くのリソースの L2 Construct が CloudWatch メトリクスを返す `metricXxx` 関数または `metric` プロパティを持つ
- CloudWatch アラームやダッシュボードを作成するときに正しい名前を調べなくて済む

https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/resources.html#resources_metrics

connections | ネットワーク接続を簡単に定義する

- connections プロパティを使用して、リソース間の通信を簡単に許可できる
- セキュリティグループやネットワーク ACL の設定を抽象化
- 2 つのリソース間の接続を許可すると両方のセキュリティグループを適切に設定
- ELB や RDS などデフォルトポートを定義している一部のリソースでは、ポートの指定が不要
- Stack をまたぐ接続の場合は allowFrom / allowTo を使い分けて循環参照を回避

https://docs.aws.amazon.com/cdk/v2/guide/resources.html#resources_traffic

TypeScript

```typescript
// fleet1 -> インターネットへの通信を許可
fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443, toPort: 443 }));
// fleet2 -> fleet1 に任意の通信を許可
fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic());
// 任意のIPv4アドレス -> ALB リスナー にデフォルトポートの接続を許可
listener.connections.allowDefaultPortFromAnyIpv4('Allow public access');
// fleet1 -> RDS にデフォルトポートの接続を許可
fleet1.connections.allowToDefaultPort(rdsDatabase, 'Fleet can access database');
```

イベント通知設定の抽象化

- リソースが生成する特定のイベントに対してハンドラーを簡単に登録できる

https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/resources.html#resources_events

Amazon S3 Bucket の例：addObjectCreatedNotification … Amazon S3 イベント通知

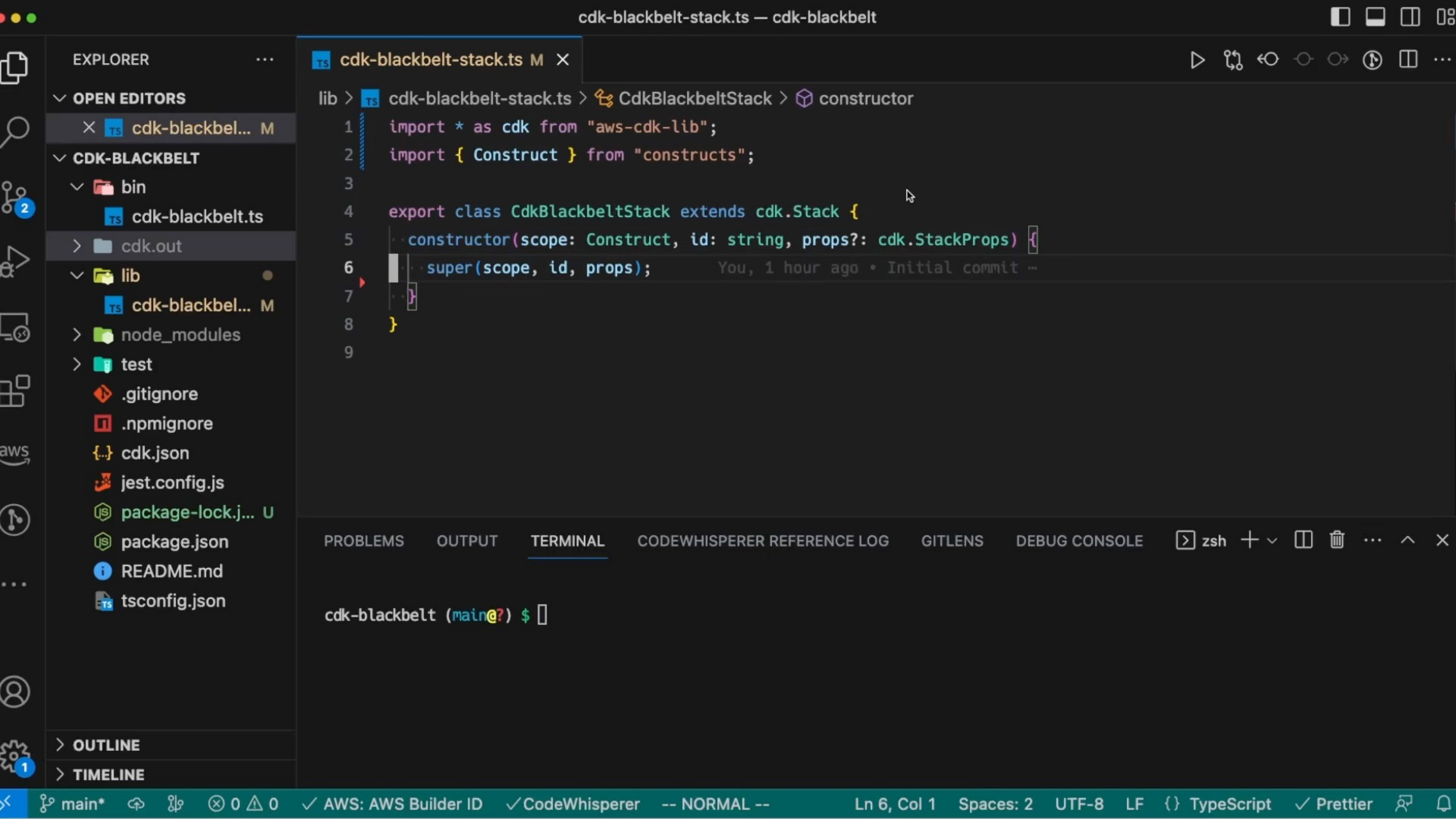
TypeScript
```
const handler = new lambda.Function(this, 'Handler', { /*…*/ });
const bucket = new s3.Bucket(this, 'Bucket');
bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler));
```

AWS CodeCommit Repository の例：onCommit … Amazon EventBridge 経由の通知

TypeScript
```
// starts a CodeBuild project when a commit is pushed to the "main" branch of the repo
repo.onCommit('CommitToMain', {
 target: new targets.CodeBuildProject(project),
 branches: ['main'],
});
```

デモ

- Amazon S3 バケットにファイルをアップロード (Asset)
- AWS Lambda 関数の開発とデプロイ (Hotswap)
- 権限設定の抽象化 (Grant)


cdk-blackbelt-stack.ts — cdk-blackbelt
EXPLORER
OPEN EDITORS
cdk-blackbel... M
CDK-BLACKBELT
bin
cdk-blackbelt.ts
cdk.out
lib
cdk-blackbel... M
node_modules
test
.gitignore
.npmignore
cdk.json
jest.config.js
package-lock.j... U
package.json
README.md
tsconfig.json
OUTLINE
TIMELINE
cdk-blackbelt-stack.ts M
lib > cdk-blackbelt-stack.ts > CdkBlackbeltStack > constructor
1 import * as cdk from "aws-cdk-lib";
2 import { Construct } from "constructs";
3
4 export class CdkBlackbeltStack extends cdk.Stack {
5 constructor(scope: Construct, id: string, props?: cdk.StackProps) {
6 super(scope, id, props); You, 1 hour ago • Initial commit
7 }
8 }
9
PROBLEMS
OUTPUT
TERMINAL
CODEWHISPERER REFERENCE LOG
GITLENS
DEBUG CONSOLE
zsh
cdk-blackbelt (main@?) $
main*
0 0
AWS: AWS Builder ID
CodeWhisperer
-- NORMAL --
Ln 6, Col 1
Spaces: 2
UTF-8
LF
TypeScript
Prettier

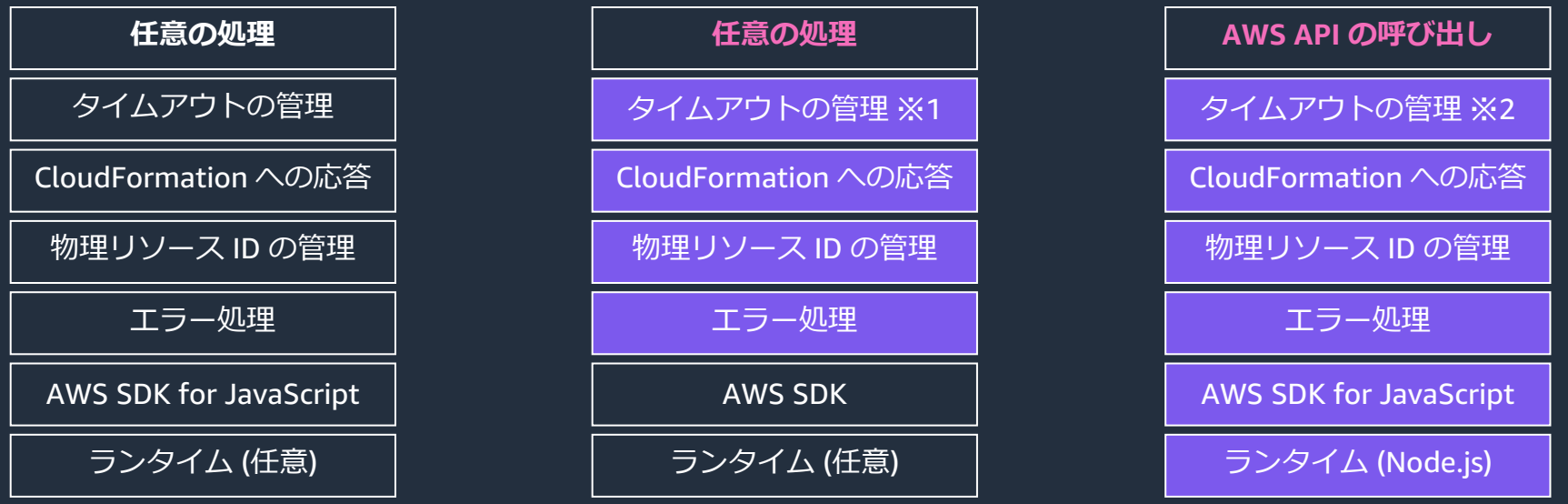
デプロイ中に
任意の処理を実行する

AWS CDK から見た CloudFormation Custom Resources

- CloudFormation Custom Resources は AWS Lambda 関数または Amazon SNS に
  イベント (Create/Update/Delete) を送信し、ユーザーが作成したハンドラーを実行する機能
- CloudFormation で未対応、または外部のリソースをプロビジョニングするのに使用されるが
  この用途では代替として CloudFormation Registry の利用が推奨されている
- CDK では CloudFormation によるデプロイ中に任意の処理を差し挟む拡張ポイントとして扱える

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/template-custom-resources.html

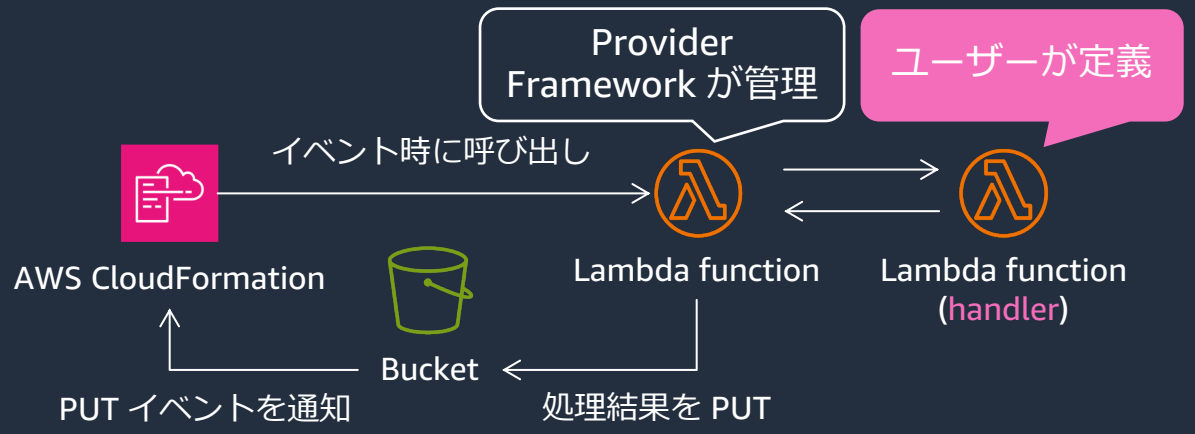
AWS CDK でカスタムリソースを扱う方法



AWS CDK ではカスタムリソースを実装するための Provider Framework と
Lambda の実装不要で AWS API の呼び出しを簡単に行える AwsCustomResource を提供

※1 ... 非同期モードにより Lambda のタイムアウト (15分) を超えることが可能
※2 ... AWS SDK のタイムアウトを管理

Provider Framework



https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.custom_resources-readme.html#provider-framework

- ユーザーが実装する Lambda 関数（ハンドラー）の処理を簡素化するための小さなフレームワーク
  - CloudFormation のライフサイクルイベントを受けてハンドラーを呼び出し、結果を S3 に PUT
  - ハンドラーの戻り値の検証や例外処理を代行
- ユーザーは任意の言語でハンドラーを実装
- 非同期ハンドラーで Lambda のタイムアウト (15分) を超える処理を実装可能
- 物理リソース ID のデフォルト動作を実装
  - ハンドラーが PhysicalResourceId を返さない場合 Create 時はイベントの Request ID が使用される。Update 時は現在の PhysicalResourceId を使用。

Provider Framework

CDK App

lib/stack.ts

```
// 任意のランタイムを使用して lambda.Function を定義
const handler = new NodejsFunction(this, 'Handler', {
 entry: 'lambda/handler.ts',
 handler: 'handler',
 runtime: Runtime.NODEJS_18_X,
});

// handler を呼び出す Provider を宣言
const provider = new Provider(this, 'Provider', {
 onEventHandler: handler,
});

// CustomResource が Provider を呼び出すように設定
new CustomResource(this, 'Resource1', {
 serviceToken: provider.serviceToken,
});
```

Lambda 関数 (ハンドラー)

lambda/handler.ts

```
import { CdkCustomResourceHandler } from 'aws-lambda';

export const handler: CdkCustomResourceHandler = async
function (event) {
 console.log(event.RequestType);
 switch (event.RequestType) {
  case 'Create':
  case 'Update':
   // エラーが発生した場合は throw new Error() で例外を送出
   return {
    PhysicalResourceId: 'hoge',
    Data: { key: 'value' }
   };
  case 'Delete':
   return {};
 }
};
```

TypeScript でハンドラーを定義する場合、型定義をインストール

```
npm install -D @types/aws-lambda
```

https://github.com/DefinitelyTyped/DefinitelyTyped/blob/master/types/aws-lambda/trigger/cdk-custom-resource.d.ts

AwsCustomResource

- AWS API を 1つだけ呼び出したい場合、Lambda 関数のコードを書かずに実装可能
- AWS API のレスポンスをリソースの属性として抽出し、他のリソースから参照可能
- AWS SDK for JavaScript を使用するため `service`, `action`, `parameter` の値は SDK に準拠
  - AWS CDK v2.87.0 以上では `service` にパッケージ名、`action` に `XxxCommand` を指定 *1
- 最新の AWS SDK をインストールするには `installLatestAwsSdk` を `true` に設定

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.custom_resources-readme.html#custom-resources-for-aws-apis

大文字・
小文字を区別

*1 ... AWS CDK v2.87.0 以上では
Lambda 関数のランタイムが
デフォルトで Node.js 18 に変更。
(AWS SDK for JavaScript v3 を同梱)

AWS SDK for JavaScript v2 の
記法も引続き有効だが、
今後は v3 の記法を推奨。

TypeScript

```typescript
new AwsCustomResource(this, 'GetParameter', {
 resourceType: 'Custom::SSMParameter',
 onUpdate: {
  service: '@aws-sdk/client-ssm', // 'SSM' in v2
  action: 'GetParameterCommand', // 'getParameter' in v2
  parameters: {
   Name: 'foo',
   WithDecryption: true,
  },
  physicalResourceId: PhysicalResourceId.fromResponse('Parameter.ARN'),
 },
});
```

Triggers

- デプロイ中（CloudFormation Stack の実行中）に Lambda 関数を実行できる
- リソースのテストや初期データの投入、前提条件の確認などに利用可能
- Lambda 関数が失敗した場合は CloudFormation Stack のデプロイに失敗
  - Lambda 関数の結果が重要でない場合は InvocationType.EVENT を指定すると無視できる
- executeAfter / executeBefore で実行順序を制御（Construct 単位）

TypeScript

```
const myTrigger = new TriggerFunction(this, 'MyTrigger', {
 runtime: Runtime.NODEJS_18_X,
 handler: 'index.handler',
 code: Code.fromAsset('my-trigger'),
});
myTrigger.executeAfter(hello, world);
myTrigger.executeBefore(goodbye);
```

テストとバリデーション

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
- jest など言語固有のテストコマンドで実行
- CDK App を作成するすべての人におすすめ

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
- 主に Construct ライブラリの提供者が使用

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
- 主に Construct ライブラリの提供者が使用

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）
- 主にガバナンスチームが作成・配布

スナップショットテスト

- CloudFormation テンプレートの出力を比較して、意図しない変更がないことを確認
- 実環境へのアクセス不要で実装コストも低い。リファクタリングやアップデートを安全に行える

テストコード

```
test("Snapshot test", () => {
  const app = new cdk.App();
  const stack = new ServerlessFullstackWebappStarterKitStack(app, "TestStack"
  const template = Template.fromStack(stack);
  expect(template).toMatchSnapshot();
});
```

スナップショット（Git にコミットする）

実行結果

```
➜  serverless-full-stack-webapp-starter-kit git:(main) ✗ npm test

> serverless-fullstack-webapp-starter-kit@0.1.0 test
> jest

 FAIL  test/serverless-fullstack-webapp-starter-kit.test.ts (9.794 s)
  × Snapshot test (468 ms)

  ● Snapshot test

    expect(received).toMatchSnapshot()

    Snapshot name: `Snapshot test 1`

    - Snapshot  - 3
    + Received  + 3

    @@ -349,11 +349,11 @@
              ],
              "EmailVerificationMessage": "The verification code to you
              "EmailVerificationSubject": "Verify your new account",
              "Policies": Object {
                "PasswordPolicy": Object {
    -             "MinimumLength": 8,
    +             "MinimumLength": 16,
                  "RequireNumbers": true,
                  "RequireSymbols": true,
```

静的解析の実装例 (cdk-nag)

簡単かつ継続的に脆弱性を監視

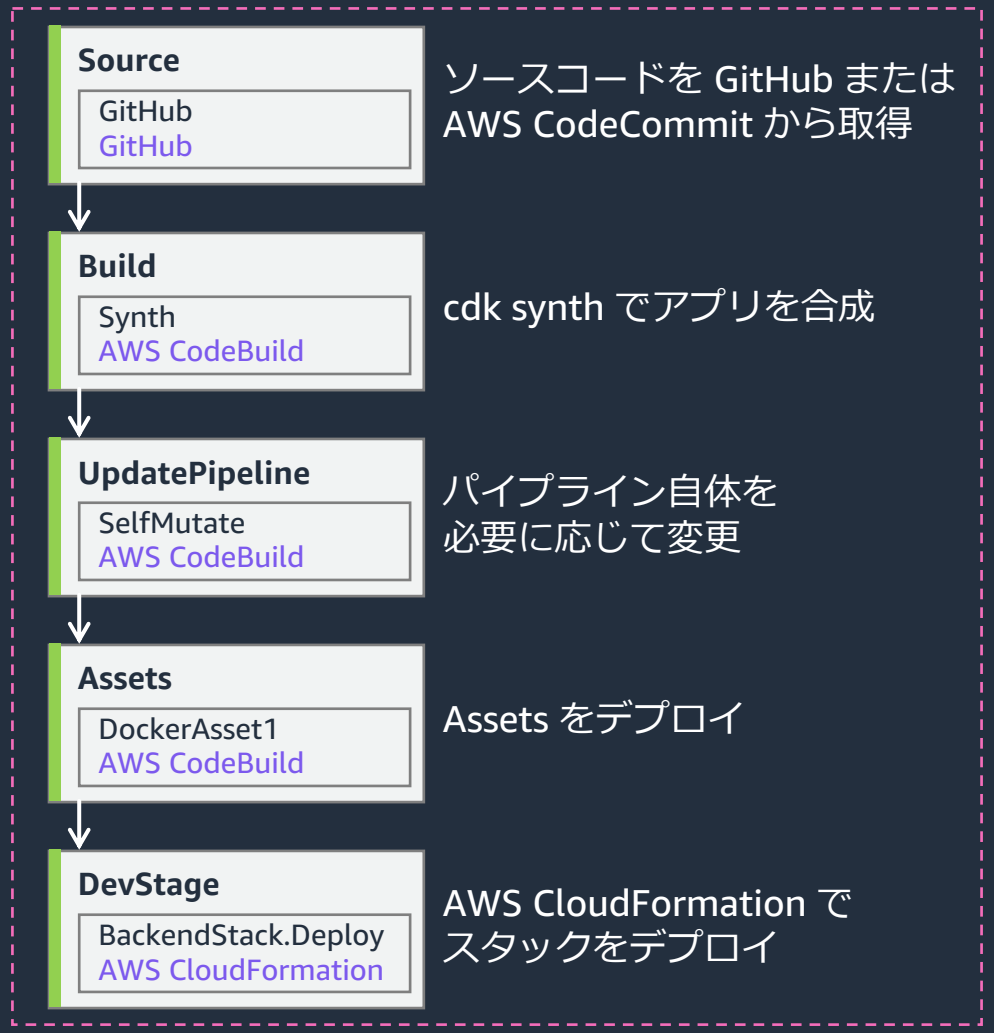
```
import { Aspects } from 'aws-cdk-lib';
import { AwsSolutionsChecks } from 'cdk-nag';
Aspects.of(app).add(new AwsSolutionsChecks());
```

Rule ID	Resource ID	Compliance	Rule Level	Rule Info
AwsSolutions-S10	ServerlessFullstackWebapp	Compliant	Error	The S3 Bucket or bucket policy does not require requests to use SSL.
AwsSolutions-S1	ServerlessFullstackWebapp	Non-Compliant	Error	The S3 Bucket has server access logs disabled.
AwsSolutions-S2	ServerlessFullstackWebapp	Compliant	Error	The S3 Bucket does not have public access restricted and blocked.
AwsSolutions-S3	ServerlessFullstackWebapp	Compliant	Error	The S3 Bucket does not default encryption enabled.
AwsSolutions-S5	ServerlessFullstackWebapp	Compliant	Error	The S3 static website bucket either has an open world bucket policy or does not use a Origin Access Identity (OAI) in the bucket policy for limited getObject and/or putObject permissions.
AwsSolutions-S10	ServerlessFullstackWebapp	Compliant	Error	The S3 Bucket or bucket policy does not require requests to use SSL.
AwsSolutions-IAM5	ServerlessFullstackWebapp	Compliant	Error	The IAM entity contains wildcard permissions and does not have a cdk-nag rule suppre evidence for those permission.
AwsSolutions-IAM4	ServerlessFullstackWebapp	Non-Compliant	Error	The IAM user, role, or group uses AWS managed policies.
AwsSolutions-IAM5	ServerlessFullstackWebapp	Compliant	Error	The IAM entity contains wildcard permissions and does not have a cdk-nag rule suppre evidence for those permission.
AwsSolutions-SQS2	ServerlessFullstackWebapp	Compliant	Error	The SQS Queue does not have server-side encryption enabled.
AwsSolutions-SQS3	ServerlessFullstackWebapp	Non-Compliant	Error	The SQS queue does not have a dead-letter queue (DLQ) enabled or have a cdk-nag ru suppression indicating it is a DLQ.
AwsSolutions-SQS4	ServerlessFullstackWebapp	Non-Compliant	Error	The SQS queue does not require requests to use SSL.

参考：AWS Cloud Development Kit と cdk-nag でアプリケーションのセキュリティとコンプライアンスを管理する
https://aws.amazon.com/jp/blogs/news/manage-application-security-and-compliance-with-the-aws-cloud-development-kit-and-cdk-nag/

CDK Pipelines

CDK Pipelines

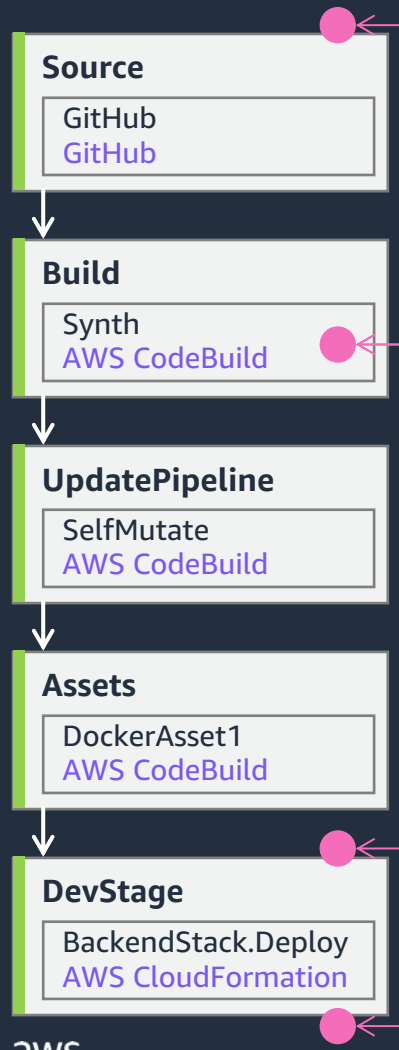


AWS CodePipeline を使った
CDK アプリの CI/CD パイプラインを
簡単にセットアップできるコンストラクトライブラリ

- cdk synth を一度だけ実行して一貫性を担保
- cdk deploy の代わりに CodePipeline でデプロイ
  - CodeBuild を最小権限でセキュアに実行
  - サードパーティライブラリや
    コンテナイメージビルド時の脆弱性の影響を縮小
- Assets と CloudFormation スタックを並列デプロイ
- Self Mutation によりパイプラインが自動更新されるため
  cdk deploy コマンドは最初の一度限り実行
- 複数の AWS アカウントとリージョンにデプロイ可能

* GitHub Actions などの他のエンジンを使用することも可能 (実験的)
https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines-readme.html#using-a-different-deployment-engine

CDK Pipelines でコマンドを追加できるポイント



既存の CodePipeline を使用する

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines-readme.html#using-an-existing-aws-codepipeline

Synth ステップ (ShellStep クラスの commands プロパティ)

(例) cdk synth の前に npm run build をフロントエンドアプリのディレクトリで実行

(例) cdk synth の後に npm run test を実行

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines-readme.html#synth-and-sources

Assets のバンドル時 (Code.fromAsset 関数の bundling プロパティなど)

(例) Lambda 関数のバンドル時に pip install を実行

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.BundlingOptions.html

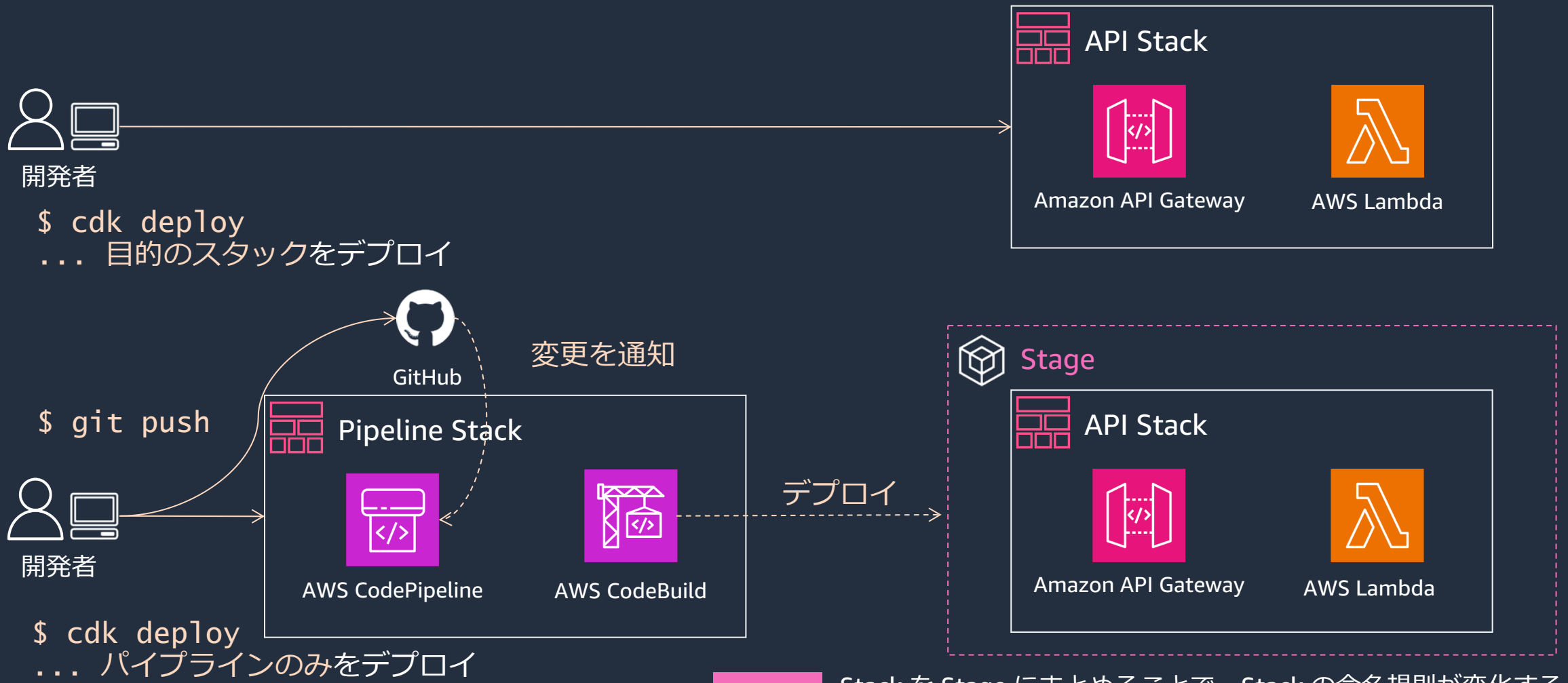
Stage の前または後 (CodePipeline.addStage 関数の pre/post プロパティ)

(例) Stage のデプロイ前に CodePipeline の手動承認ステップを追加

(例) Stage のデプロイ後に ShellStep でインテグレーションテストを実行

https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.pipelines.AddStageOpts.html

既存の CDK アプリを CDK Pipelines に移行する



Tips

Stack を Stage にまとめることで、Stack の命名規則が変化する。
CDK Pipelines への移行前後で同じ Stack にデプロイする場合は **stackName** を明示的に指定することが可能

Tips: npm

npm

- cdk init でプロジェクトを作成するとデフォルトで npm が設定される
  - 最新の npm を使用することを推奨 (npm install -g npm)
- 代わりに Yarn も使用可能 ※ Plug-and-play モードには非対応
- 依存関係を package.json に記述する
  - npm update コマンドは package.json に指定された範囲内での最新バージョンに更新（例： "aws-cdk-lib": "^2.88.0" の場合最新のマイナーバージョンに更新）
  - package.json の記述を更新するには ncu を使用 (npm install -g npm-check-updates)
    - 更新するパッケージを限定 (-f), 除外 (-x), バージョン指定 (-t minor) などのオプションが指定可能

Console

```
$ ncu          # 最新パッケージがあるかチェック
$ ncu -u       # package.jsonを書き換え
$ npm install # package.jsonに基づいてパッケージをインストール
```

AWS CDK Toolkit の呼び出し方の違い

cdk

- 一般的なコマンドと同様に環境変数 PATH にある cdk コマンドを実行
- aws-cdk パッケージをグローバルインストールすると使用可能になるが、各プロジェクトで決めたバージョンと乖離することがあるため、通常推奨しない

npm run cdk

- package.json の scripts に記載されたコマンドを実行
  - ./node_modules/.bin が PATH に追加されるためローカルパッケージが使用可能
- aws-cdk はデフォルトで scripts に "cdk": "cdk" を定義
- 位置引数はそのままコマンドに渡されるが、オプションを渡す前には `--` が必要

Console

```
$ npm run cdk ls # = cdk ls
$ npm run cdk synth -- --all
# --all が npm のオプションとして
解釈されないように -- を前置
```

npx cdk

- コマンドを探索して実行
  1. package.json の bin オプション
  2. ./node_modules/bin のコマンド（なければ親ディレクトリも探す）
  3. グローバルの bin にあるコマンド
  4. ローカルおよびグローバルの同名パッケージの bin オプション
  5. いずれもなければ一時ディレクトリにインストールして実行
- npm プロジェクト外でも使える
- npx cdk と npx aws-cdk は多くのケースで同じ動作になる *

Console

```
$ npx cdk init # プロジェクト外
$ npx cdk synth --all
```

* cdk という aws-cdk を呼び出すだけのパッケージが提供されているため

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)