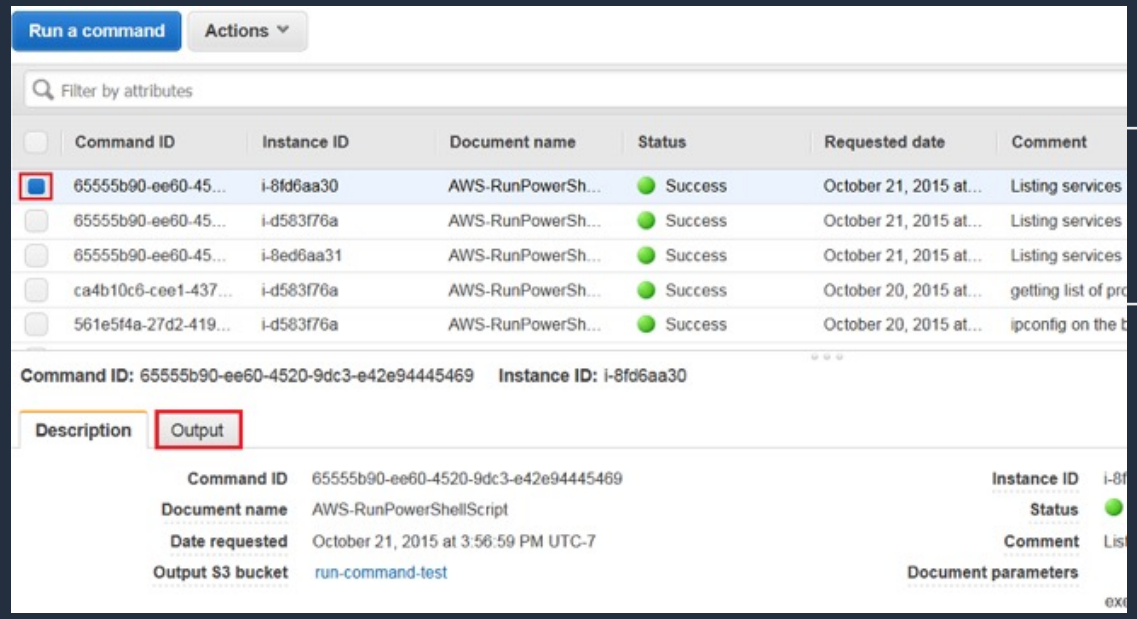
AWS CDK 入門 勉強会

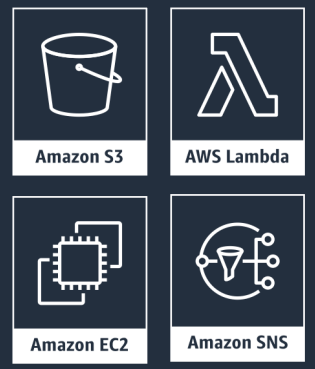
Mizuki Ugajin

Partner Solutions Architect
Amazon Web Services Japan G.K.
2022/11/18

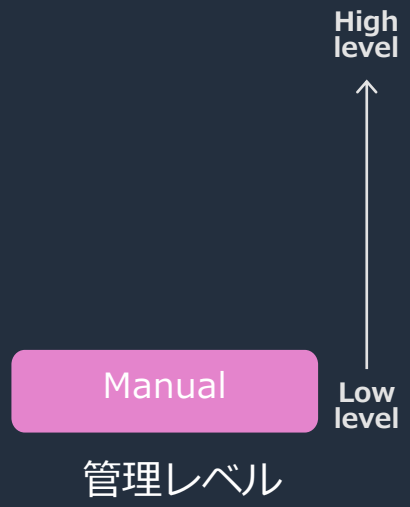
手動操作（マネジメントコンソール）

操作手順書が別途必要





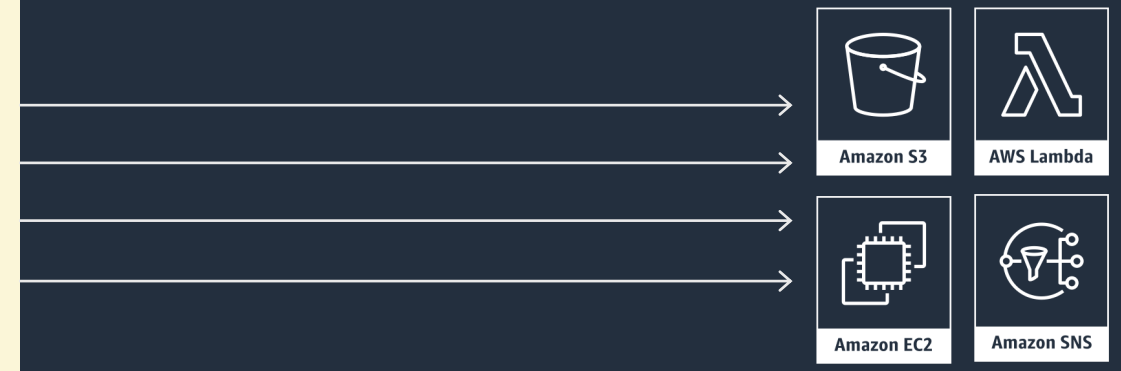
- ○ 始めるのは簡単
- × 繰り返し可能ではない
- × エラーが起きやすい
- × 時間がかかる



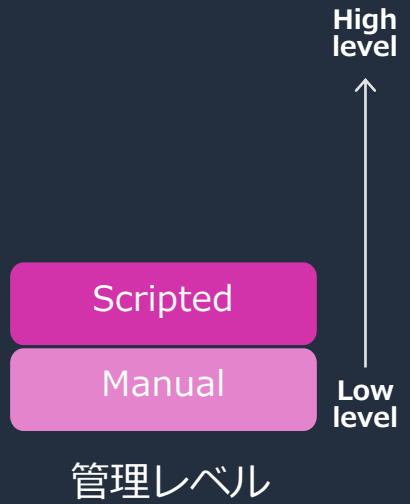
スクリプト (SDK, CLI)

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'
  }
```

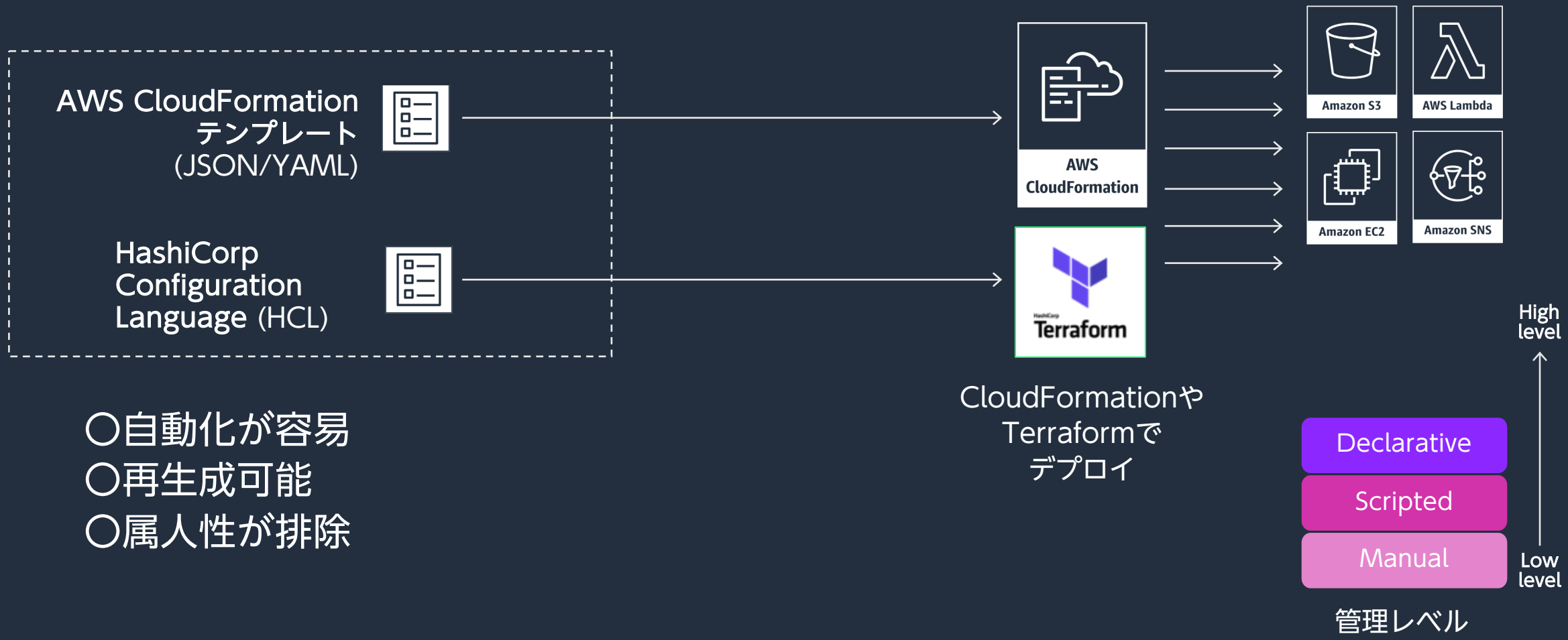


- ？ APIコールが失敗したら何が起こる？
- ？どうやってアップデートする？
- ？リソースが準備完了なのはどうやって知る？
- ？どうやってロールバックする？



プロビジョニング ツール (CloudFormationなど)

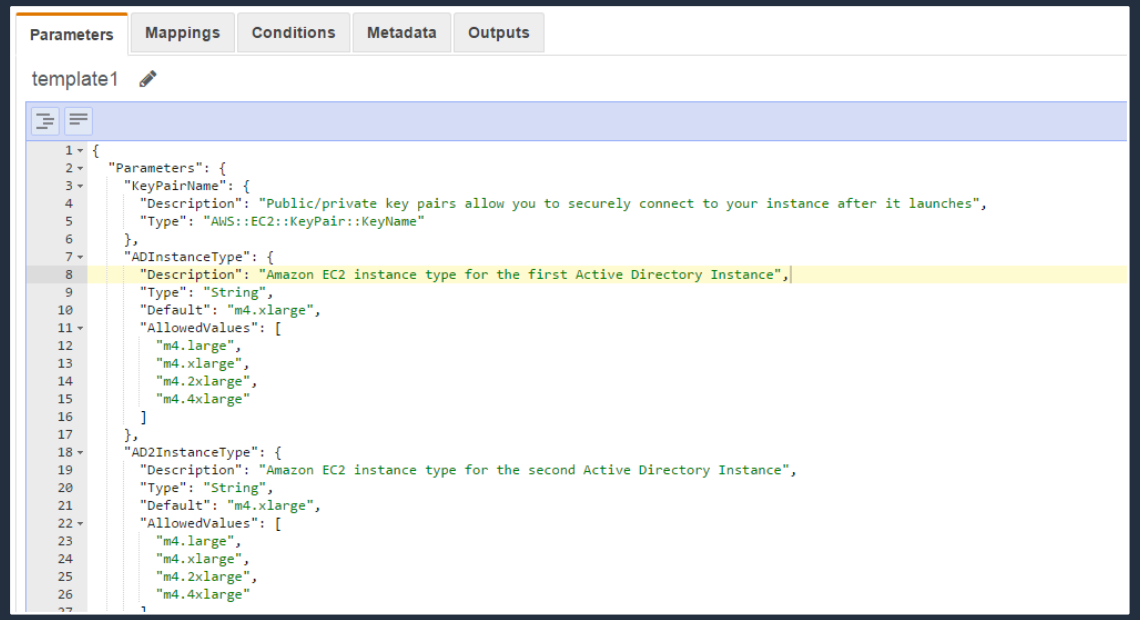
あるべき状態の定義が可能



〇自動化が容易
〇再生成可能
〇属人性が排除

Infrastructure as Code

コードによってリソースをモデル化

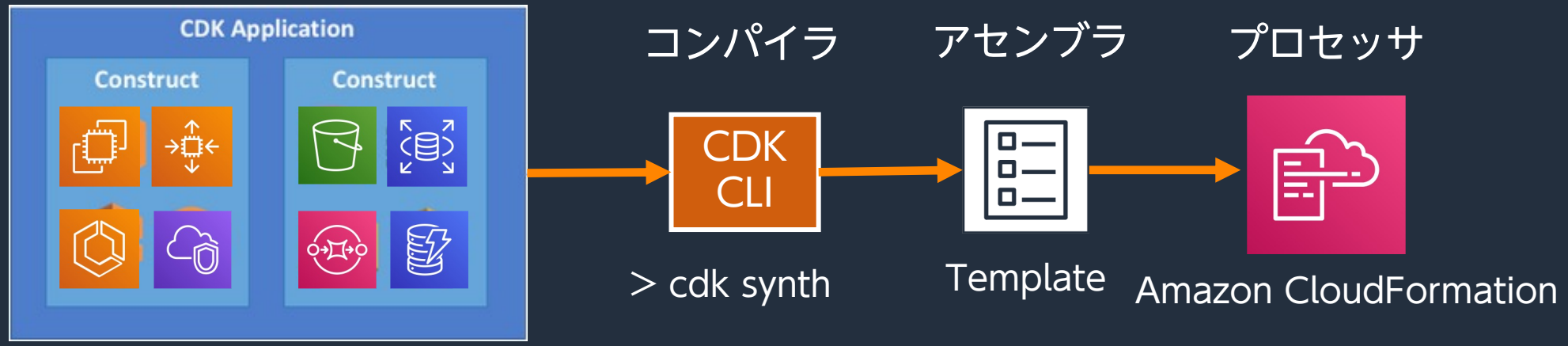


- 可能な限り自動化
- アプリケーションと
  インフラストラクチャの両方
- 宣言的にインフラを定義
- セキュリティを含め注意深くインフラを設計
- 定義や設定を
  アプリケーションコードのごとく扱う
- バージョンコントロール
- アプリケーションの一部としてのインフラ
- ロールバックのためのプラン

AWS Cloud Development Kit (CDK)

AWS CDK（AWS Cloud Development Kit）

- インフラをPython, TypeScript等のコードで定義してAWS CloudFormationでプロビジョニング
- オブジェクト指向的に記述&ループ等が使用可能
- 推奨デフォルト値が設定済みのためコード量が少なく済む
- 現在、JavaScript、TypeScript、Python、 Java、 C#に対応

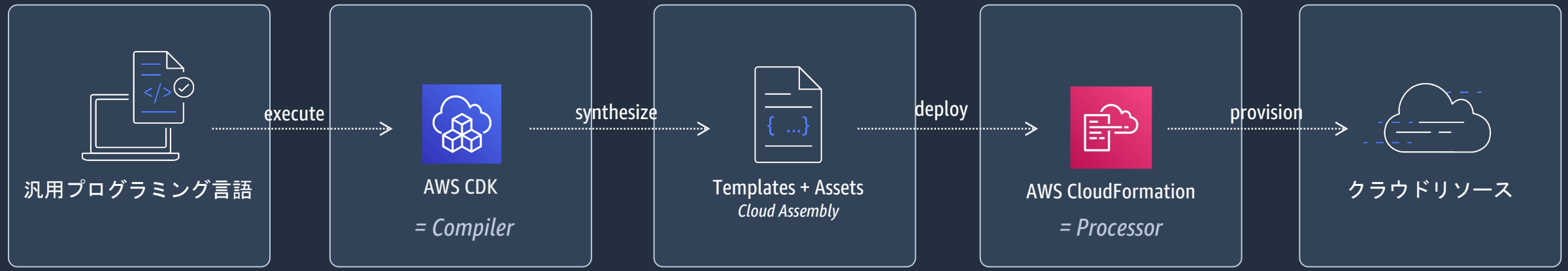


AWS CDK（AWS Cloud Development Kit）

例）S3バケット作成

```
const cdk = require('@aws-cdk/core');
const s3 = require('@aws-cdk/aws-s3');

class MyStack extends cdk.Stack {
    constructor(parent, id, props) {
        super(parent, id, props);

        new s3.Bucket(this, 'MyFirstBucket', {
            versioned: true
        });
    }
}
```

AWS CDK のメリット



AWS CDK による抽象化のメリット

- ✓ 安全なデフォルト値とベストプラクティスによる抽象化
- ✓ コード量が少なく可読性が高い
- ✓ 一般のプログラミング言語の制御構文や抽象化を利用可能
- ✓ IDE やテキストエディタによる補完
- ✓ コンパイラや静的解析ツール、テストによる実行前のチェック
- ✓ リソースに加え、スタック間の依存関係を自動で解決
- ✓ インフラだけでなくアプリのコードや設定を含め一元管理

AWS CloudFormation のメリット

- ✓ あるべき状態が定義されている (宣言的)
- ✓ 人手を介さず、繰り返し実行可能
- ✓ サービスが冪等性、順序、ロールバックを担保
- ✓ バージョン管理が可能
- ✓ プルリクエストやコードレビューなど、アプリ開発のプラクティスを適用可能

aws

AWS CDK のコンセプト

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

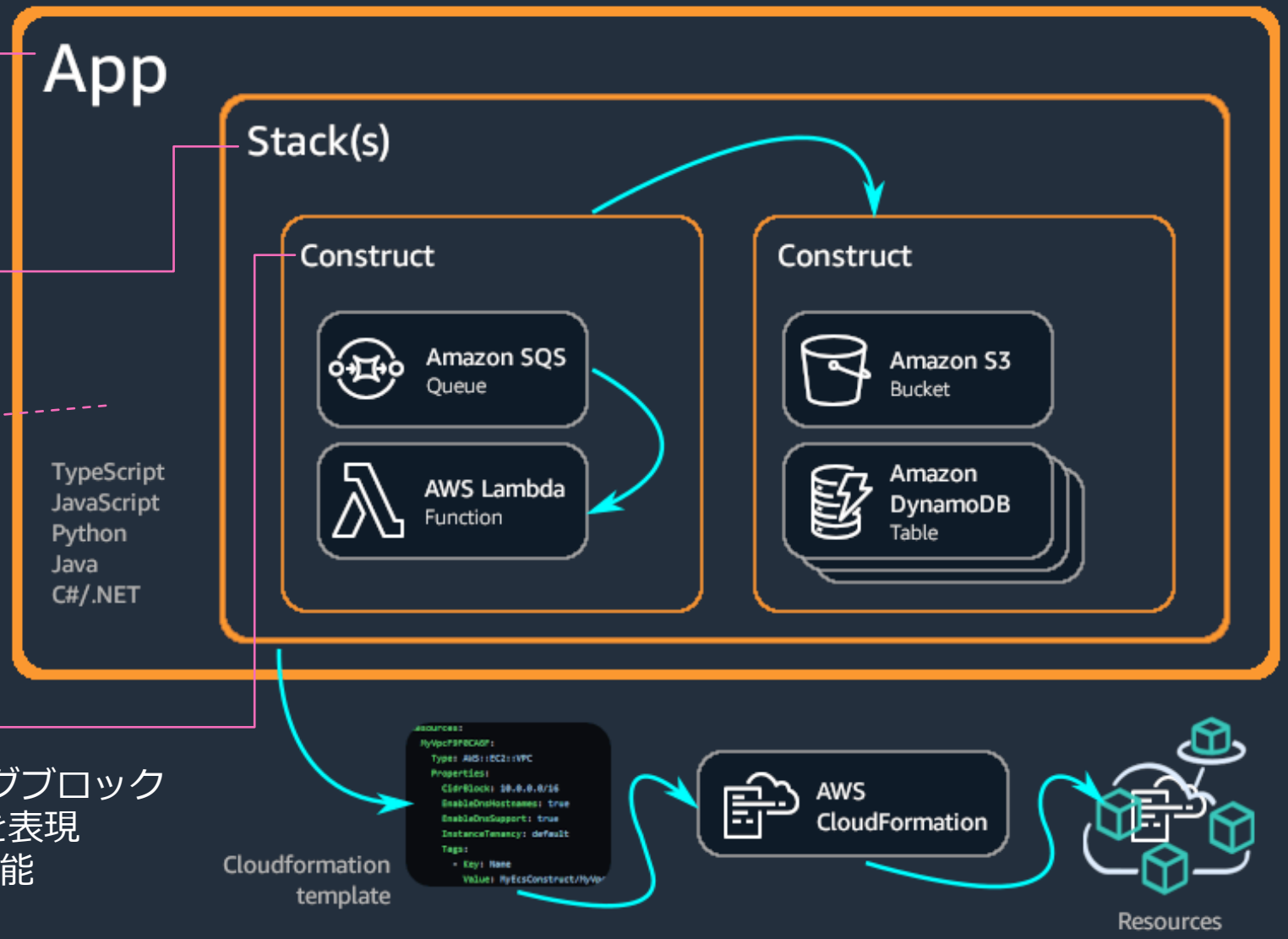
- CloudFormation スタックに対応
- デプロイ可能な最小単位

Stage (Optional)

- 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

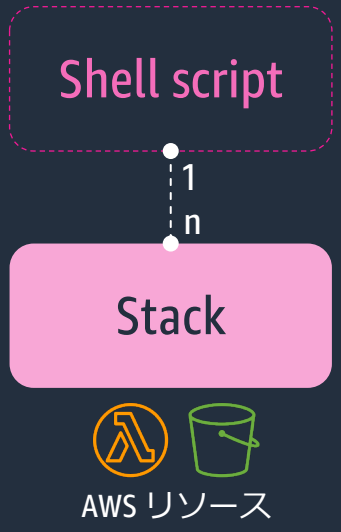
Construct

- 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



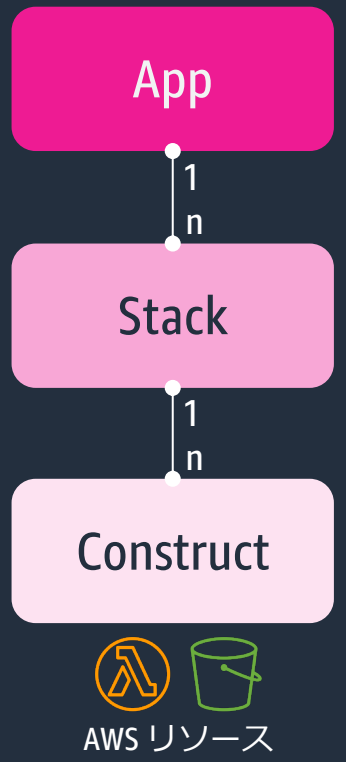
AWS CDK の概念 - CloudFormation との比較

AWS CloudFormation



- スタックを1つずつデプロイ
  $ aws cloudformation deploy HogeStack
- 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

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事前に定義したもの
- `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 (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

aws

CloudFormation テンプレート

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

```yaml
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

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


```
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

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

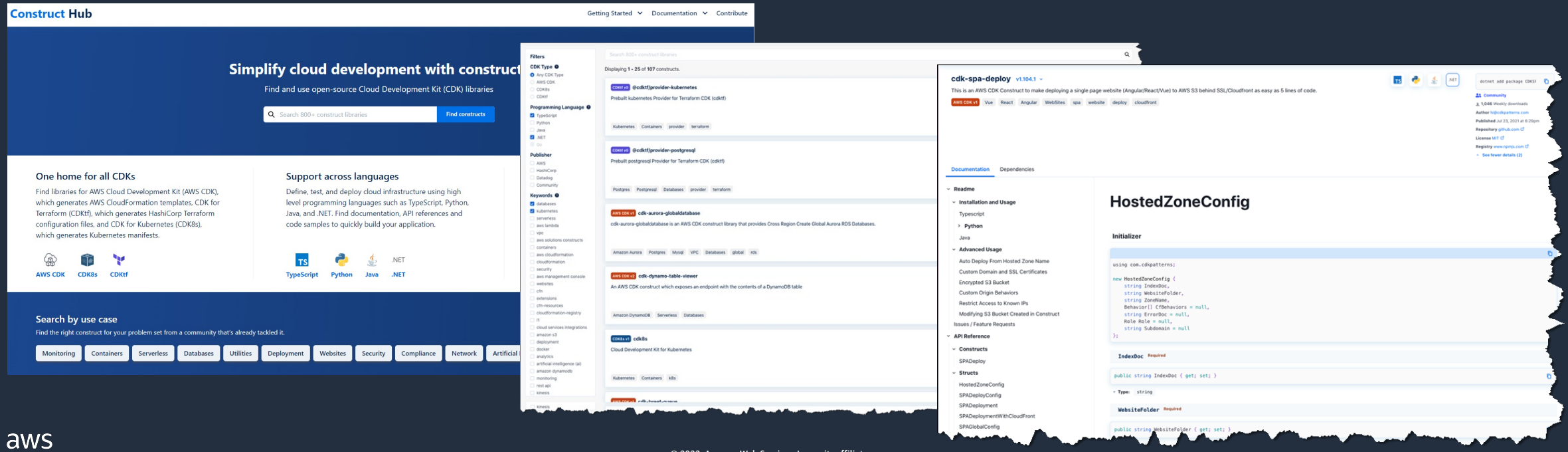
```
const bucket = new s3.Bucket(this, 'MyBucket');

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 Hub

https://constructs.dev/

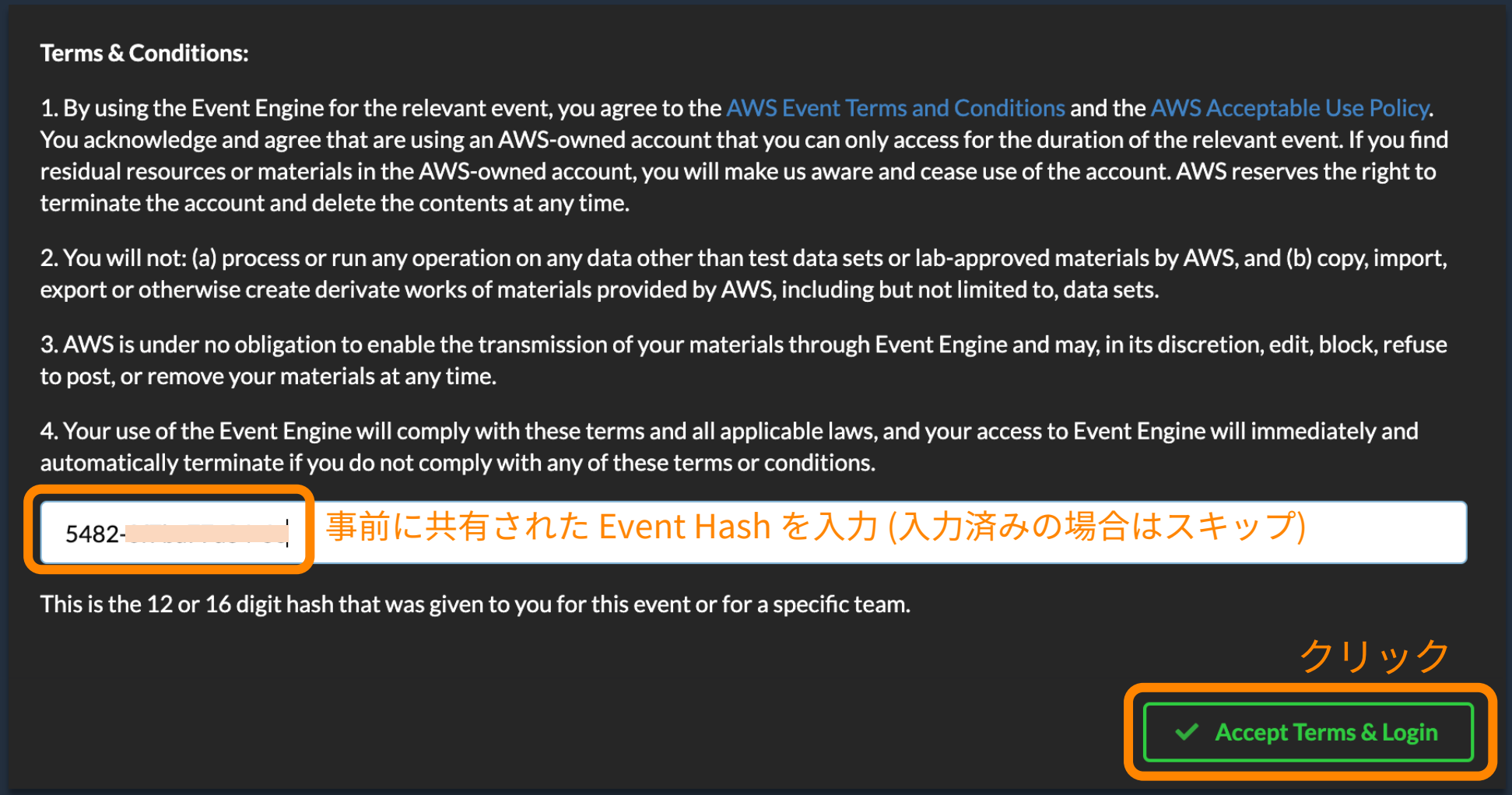
- 1,000 以上のオープンソースのコンストラクトを公開
  - 公開されたコンストラクトを組み合わせることで、
    目的に合わせたアプリケーション環境をさらに迅速に構成できる
  - CDK, CDK8s, CDKtf などのタイプやバージョン、言語、パブリッシャーなどで検索可能



ハンズオン前準備

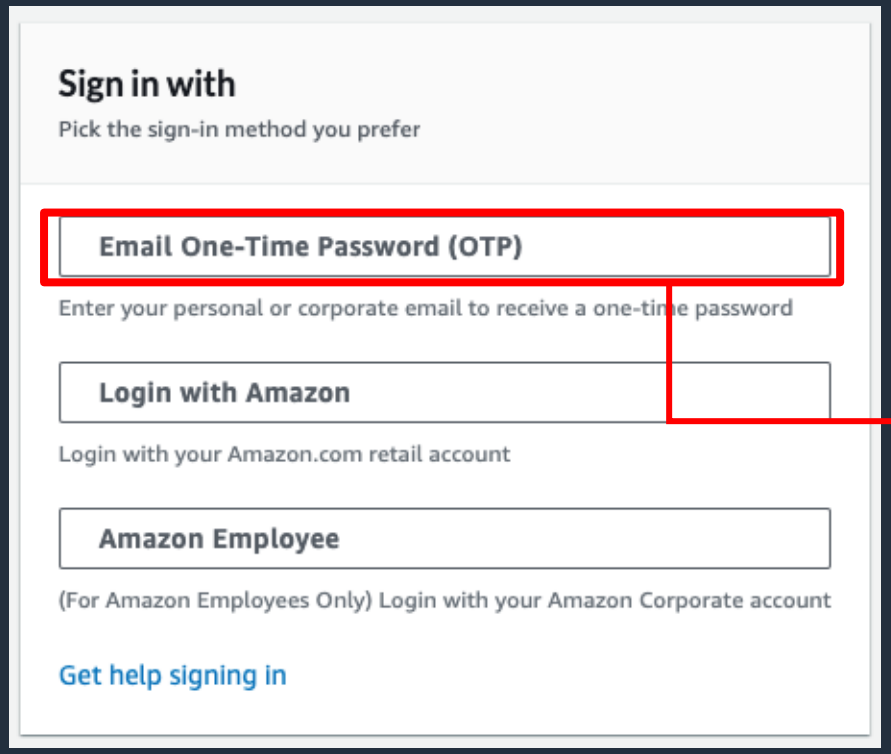
ハンズオン環境へのアクセス

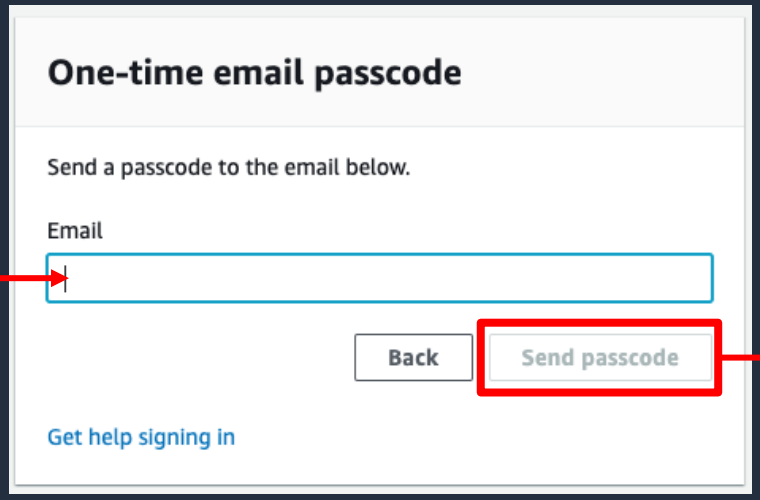
- https://dashboard.eventengine.run/login へアクセス



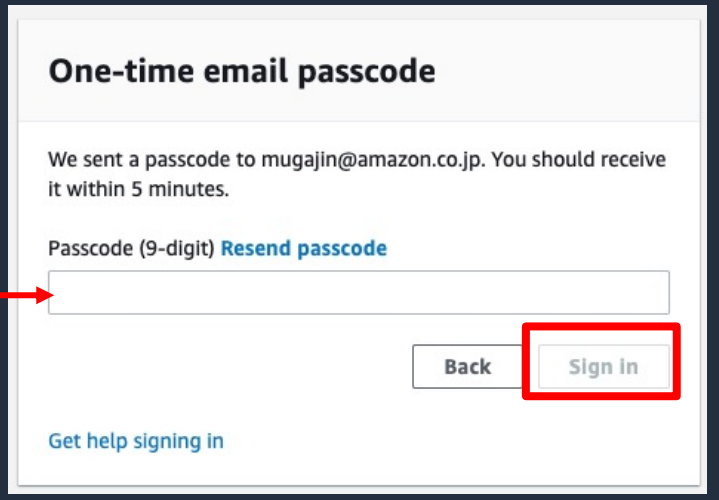
ハンズオン環境へのアクセス

OTPを選択し、メールアドレスを入力
→メールで受け取ったパスコードを入力してサインイン



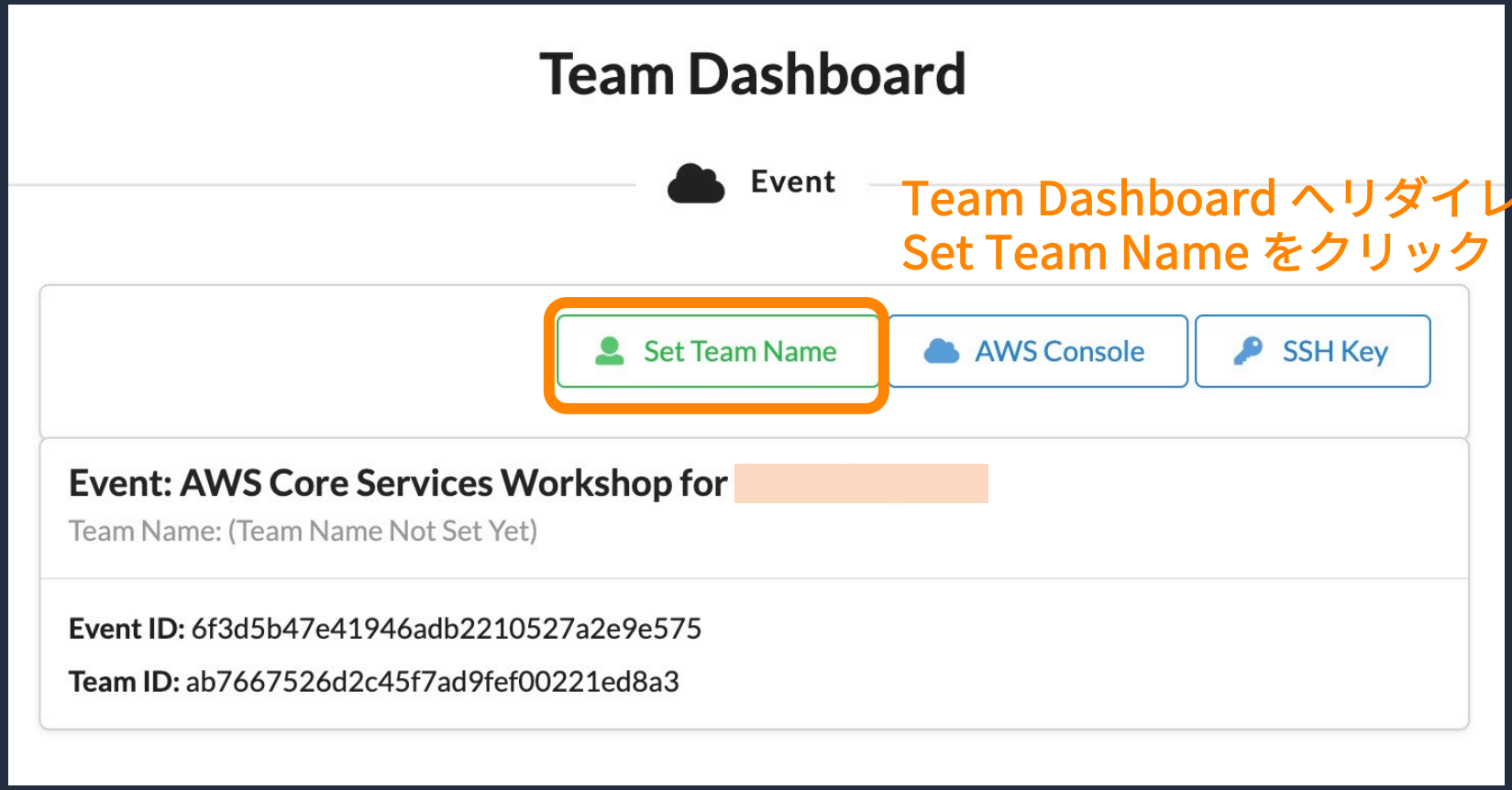


メールアドレスを入力



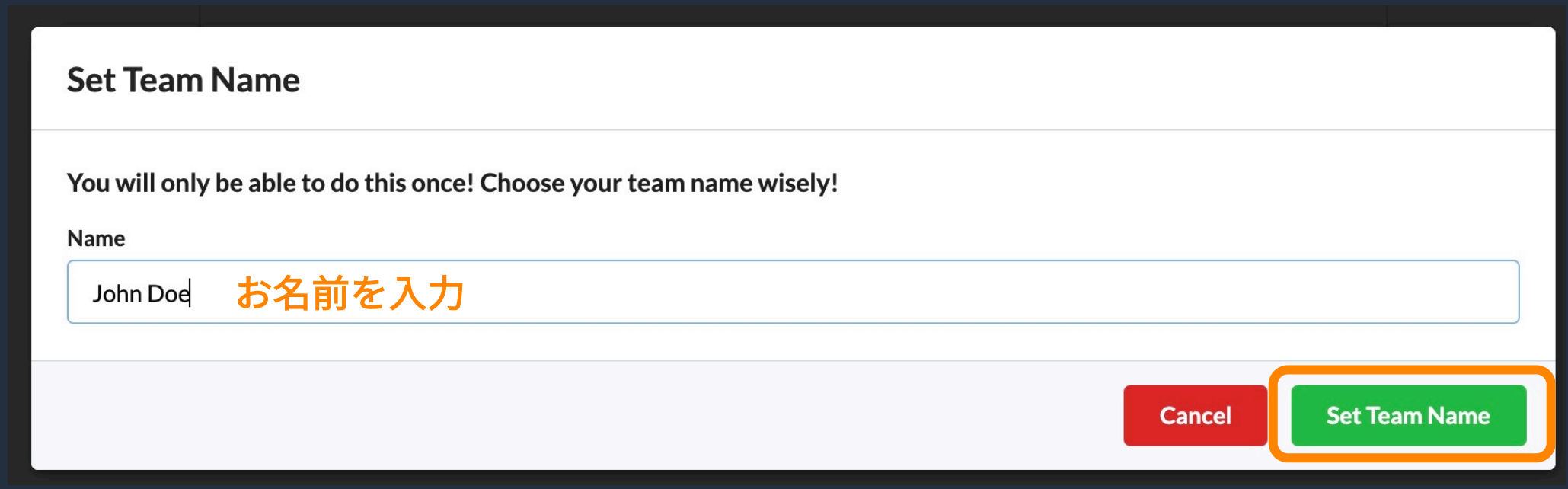
メールで届いた
パスコードを入力

チーム名をご自身のお名前に変更



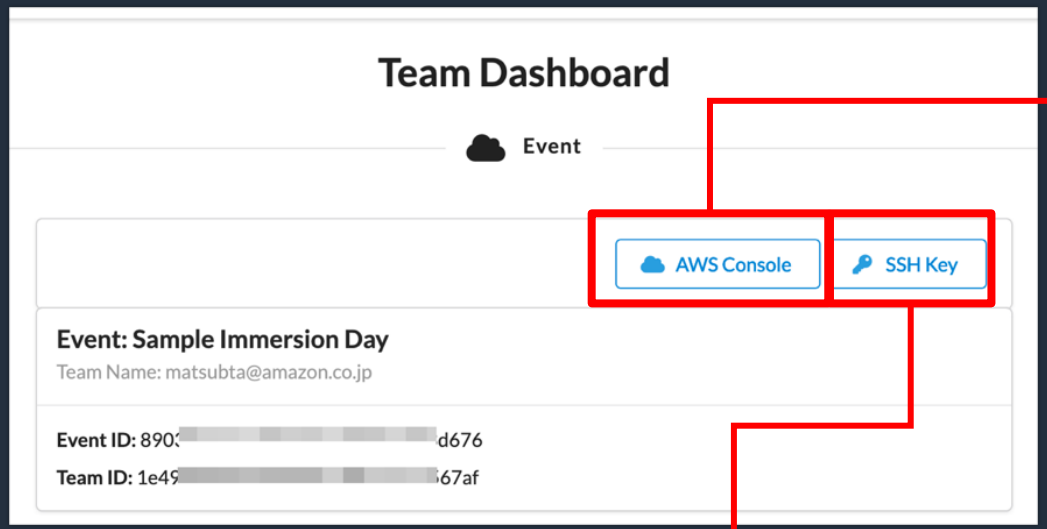
チーム名をご自身のお名前に変更

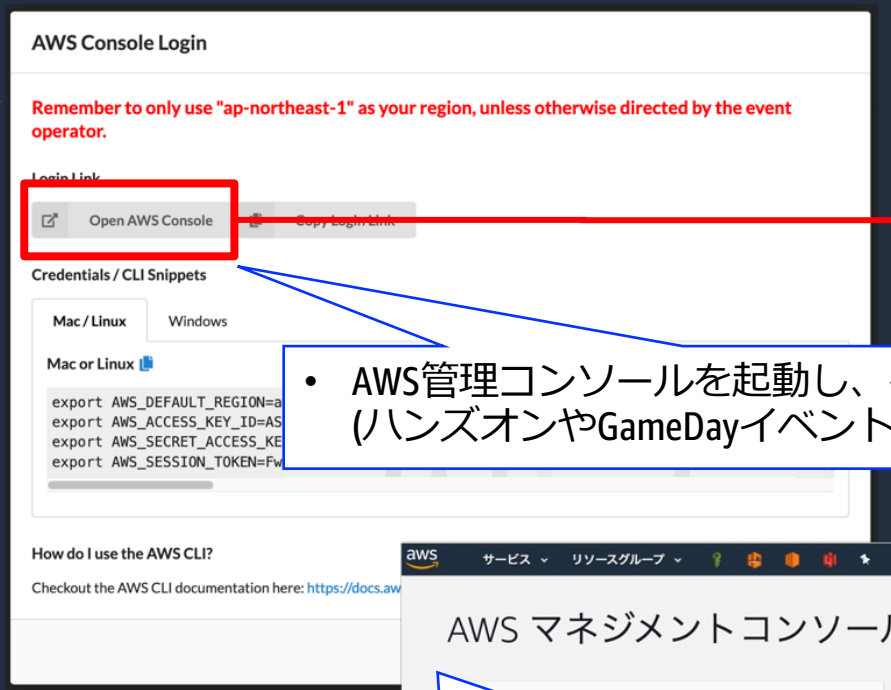
お名前を設定していただくことで、
ハンズオン手順がわからなくなった場合のサポートがスムーズになります



ハンズオン環境へのアクセス

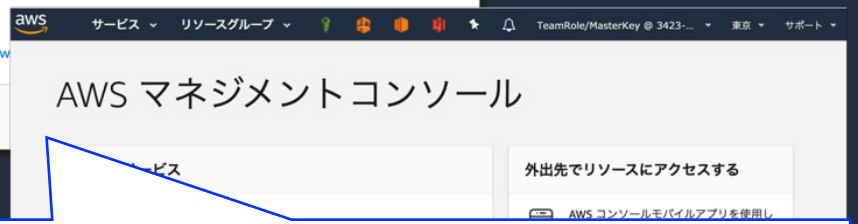
表示されるチーム画面の「AWS Console > Open AWS Console」にて、
払い出されたアカウントでAWS管理コンソールにログイン可能





AWS
管理
コンソール

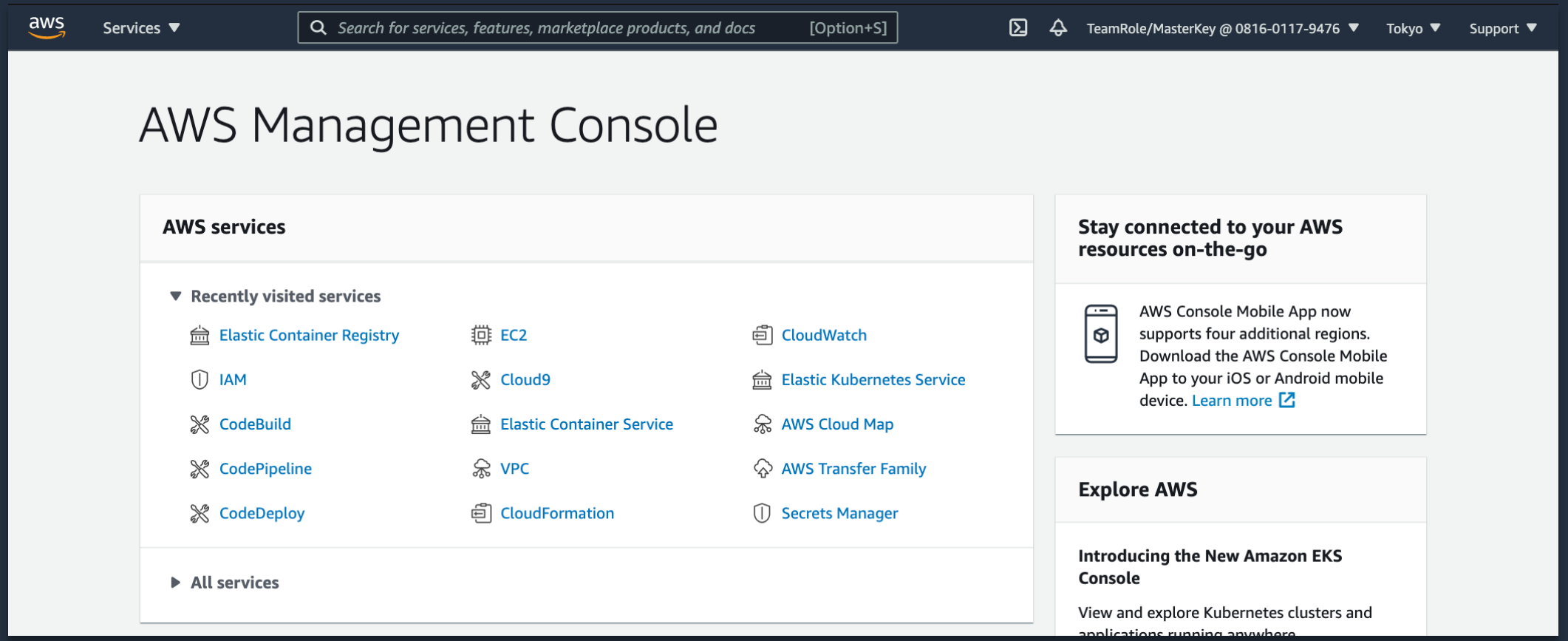
- AWS管理コンソールを起動し、イベント活動(ハンズオンやGameDayイベント)を開始



- ログイン後は、IAMユーザーの作成含め、制限内でのAWSの利用が可能

AWS マネジメントコンソールへのアクセス

AWSコンソールが表示されたら準備OK!
ハンズオンを楽しんでいきましょう！

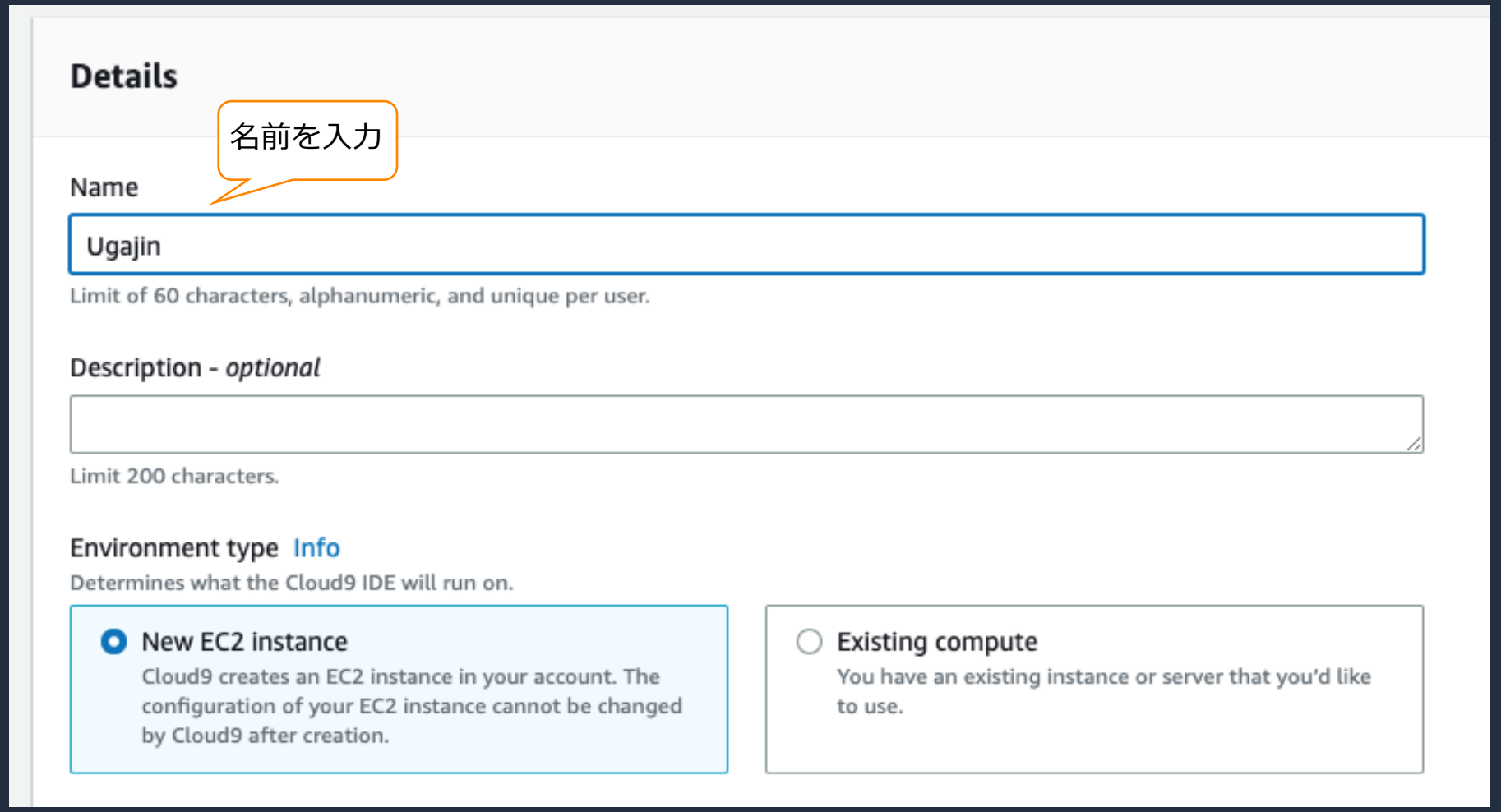


ハンズオン環境についての注意事項

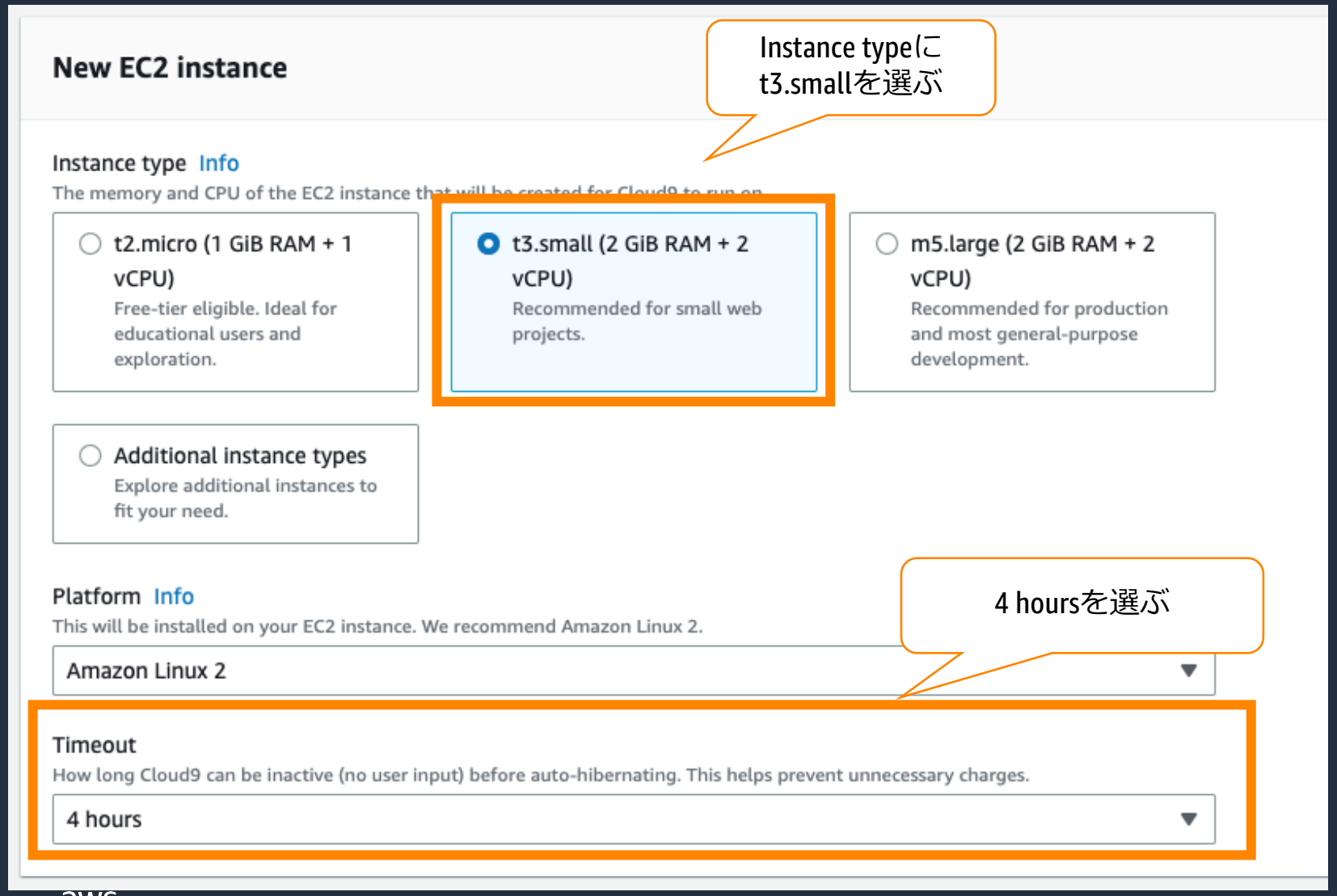
- 参加者それぞれに個別のAWSアカウントを発行しています
- この環境は**イベント終了後に削除**されます
    - 保存しておきたいファイル等は、**PCにダウンロード**しておいてください
- サポートへの問い合わせやサービス上限緩和申請は行わないでください
- ハンズオンへの参加以外の目的で利用しないでください

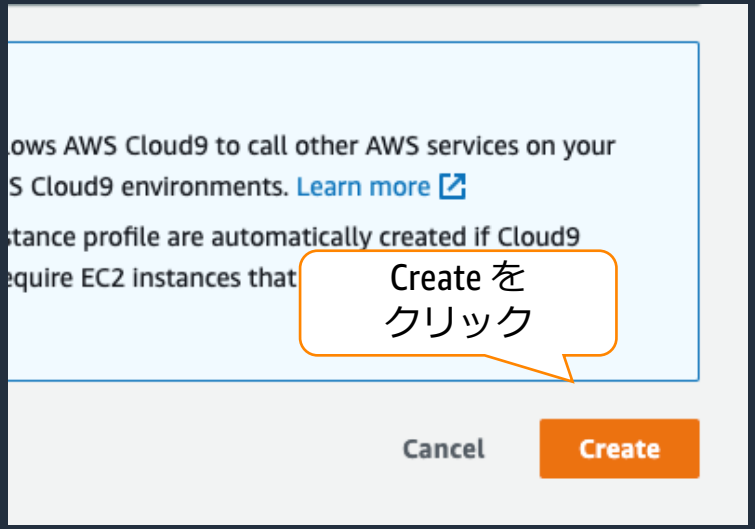
AWS Cloud9の準備

AWS Cloud9の準備



AWS Cloud9の準備

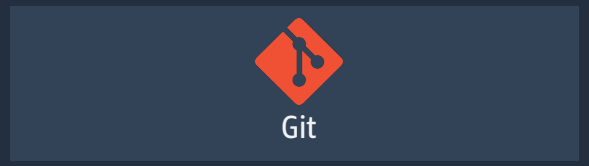




CDK を使った開発方法

AWS CDK の開発に必要なもの

Git



- Git のインストール https://git-scm.com/
- git-secrets のインストール https://github.com/awslabs/git-secrets
  - セキュリティ対策のため導入推奨 (Python 3 が必要)

IaC ツールチェーンと言語ランタイム

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

- AWS CLI v2 のインストール
  https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html
- Node.js のインストール https://nodejs.org/en/
- AWS CDK Toolkit のインストール (または npx で実行)

```
$ npm install -g aws-cdk
```

テキストエディタ / IDE

または お好きな
テキストエディタ

- (例) VSCode をインストール https://code.visualstudio.com/

(準備) 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>
```

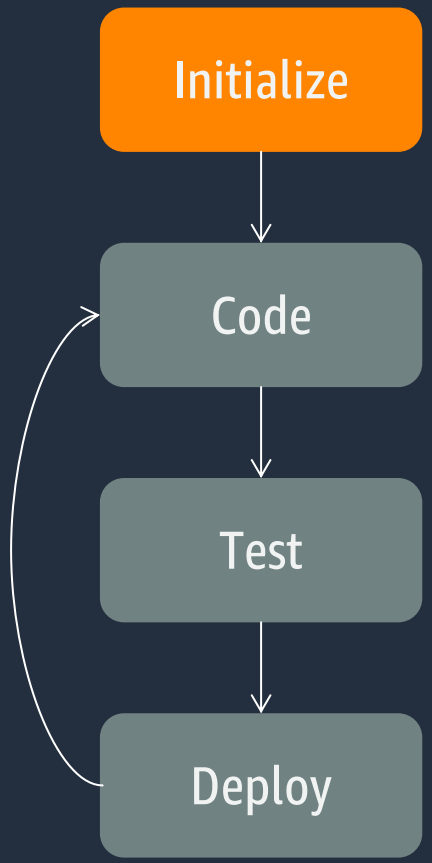
AWS SSO を使用する場合 (CDK v2.18.0~)

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

Console

```
$ aws configure sso
```

AWS CDK での開発フロー 1/4 Initialize

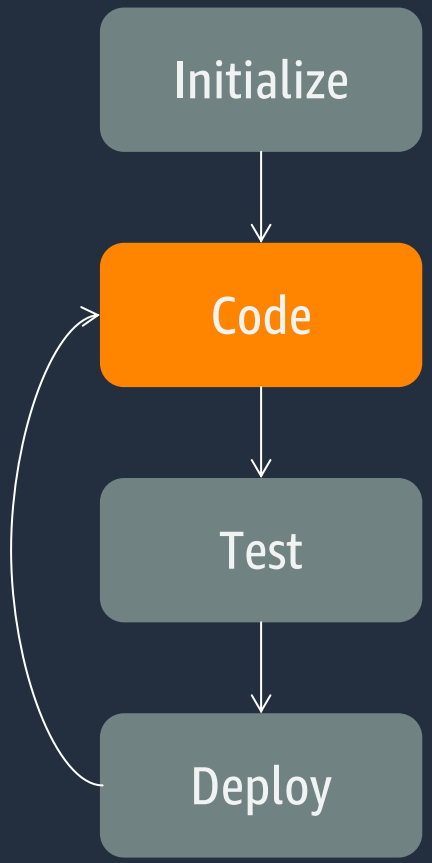


Git リポジトリを作成し、cdk init でプロジェクトを作成

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
$ cdk init app --language=typescript

# CDK Bootstrapping
$ cdk bootstrap
```

AWS CDK 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

```
.
├── 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

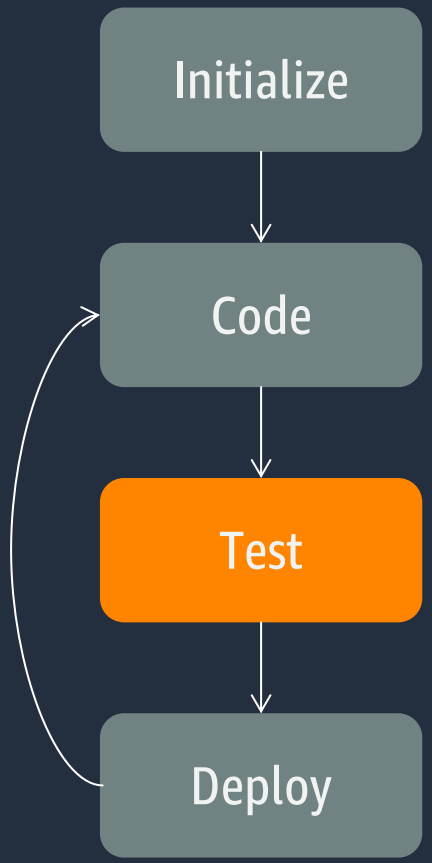
```
# App を作成
const app = new cdk.App();
# App に Stack を追加
new CdkSampleStack(app, "MyApp");
```

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

AWS CDK での開発フロー 3/4 Test

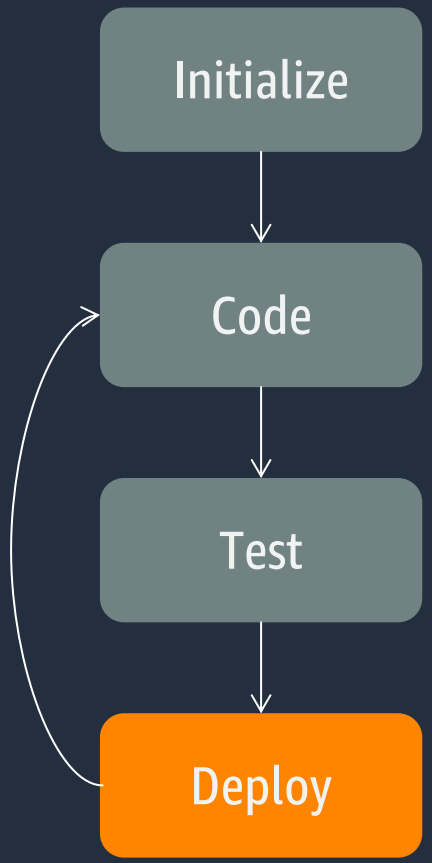


CDK アプリをテスト

Console

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

# テスト
$ npm run test
```

AWS CDK での開発フロー 4/4 Deploy



CDK アプリを AWS にデプロイ

Console

```
# AWS 上のリソースとの差分を確認
$ cdk diff

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

(参考) デプロイ時の代表的なオプション

Console

```
$ cdk deploy SampleStack # スタックを指定してデプロイ
$ cdk deploy --all --require-approval=never # デプロイ時の確認を行わない
$ cdk deploy --all --hotswap # Lambda関数などの開発時に変更を高速に反映する
$ 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
$ cdk deploy --all -c key=value # Context を指定
```

v2.0.0 ~

すばやく変更を反映: hotswap & no-rollback

`cdk deploy` と `cdk watch` (ファイルの変更を監視して自動デプロイ) に
デプロイ時間を短縮するためのオプションが追加

`--hotswap` (cdk watch ではデフォルトで有効)

- Lambda 関数、ECS コンテナイメージ、Step Functions ステートマシンを
  CloudFormation を経由せず API で直接デプロイすることで素早く反映
- 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
- Hotswap 対応リソースはこちらを参照

  https://github.com/aws/aws-cdk/blob/main/packages/aws-cdk/README.md#hotswap-deployments-for-faster-development

--no-rollback

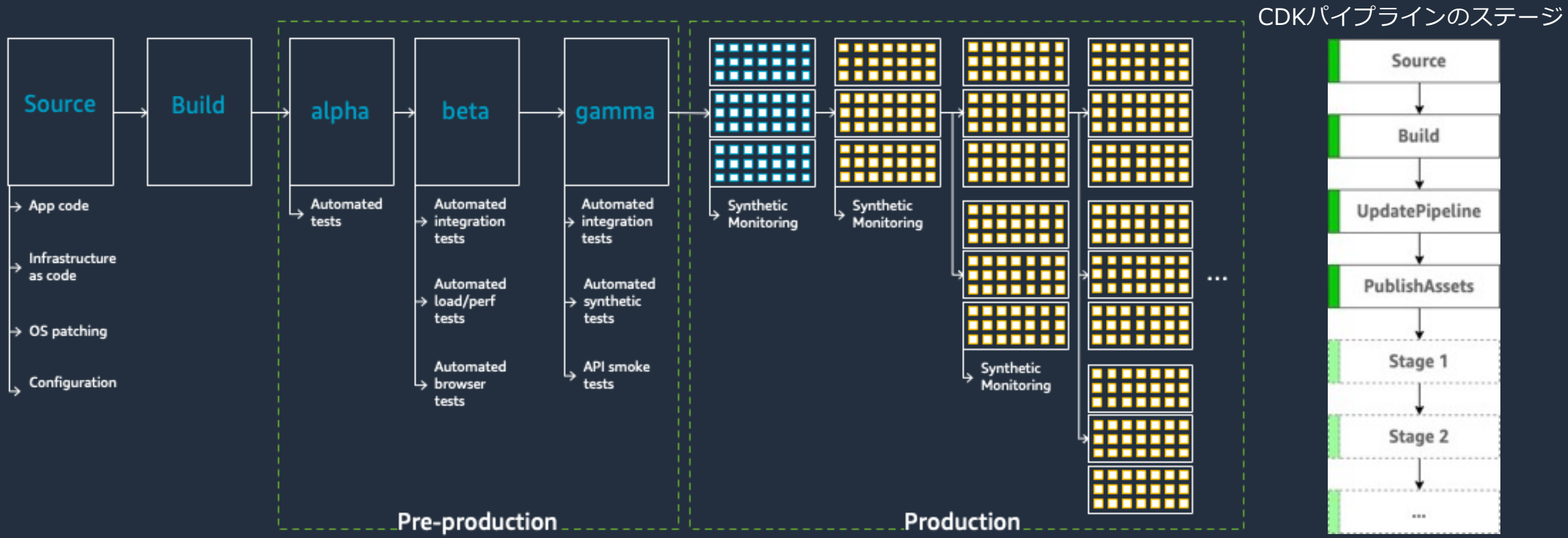
- デプロイ失敗時に CloudFormation を自動でロールバックしない
- Amazon RDS などの作成に時間がかかるリソースがある場合に有効

https://aws.amazon.com/blogs/developer/increasing-development-speed-with-cdk-watch/

aws

CDK Pipelines

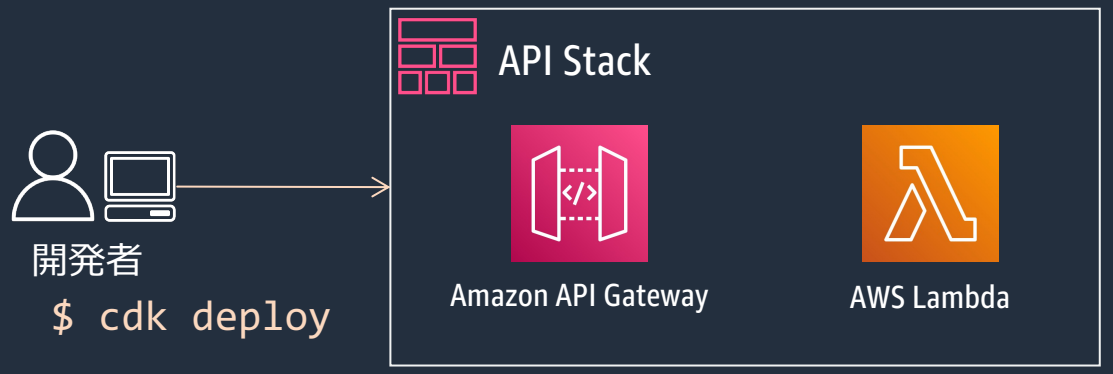
CDKアプリケーションのCI/CDパイプラインのための高レベルライブラリ。
パイプラインは、cdk deployを1回実行するだけで自己構成可能。その後、ソースコードに
新しいCDKアプリケーションまたはステージを追加すると、パイプラインが自動的に更新される



CDKパイプラインのステージ

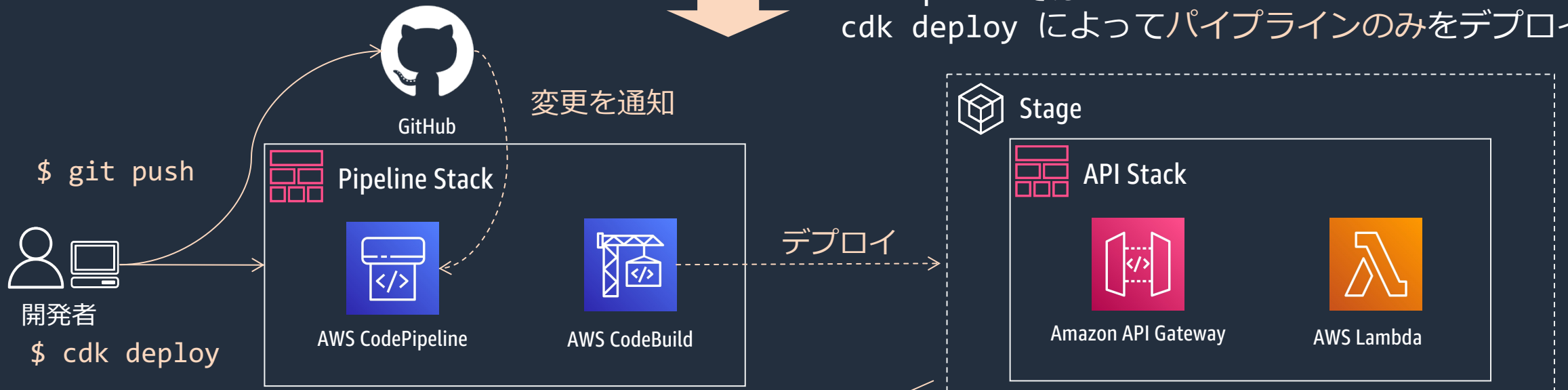
https://aws.amazon.com/jp/blogs/news/cdk-pipelines-continuous-delivery-for-aws-cdk-applications/

既存の CDK アプリを CDK Pipelines に移行する



cdk deploy によって目的のスタックをデプロイする

CDK Pipelines では
cdk deploy によってパイプラインのみをデプロイする

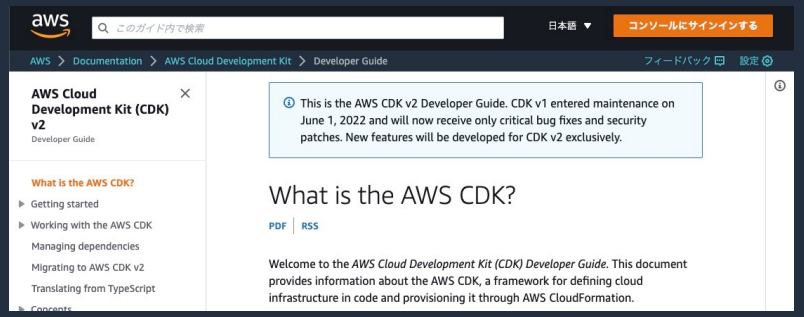


各環境にデプロイするためにスタックを Stage にまとめる

AWS CDK の学習をはじめる

AWS CDK のリソース

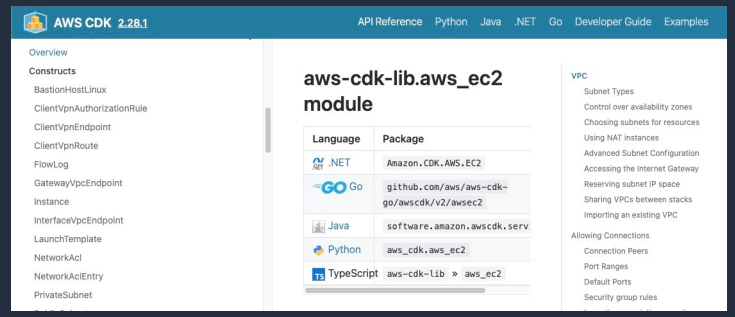
AWS CDK Developer Guide



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

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

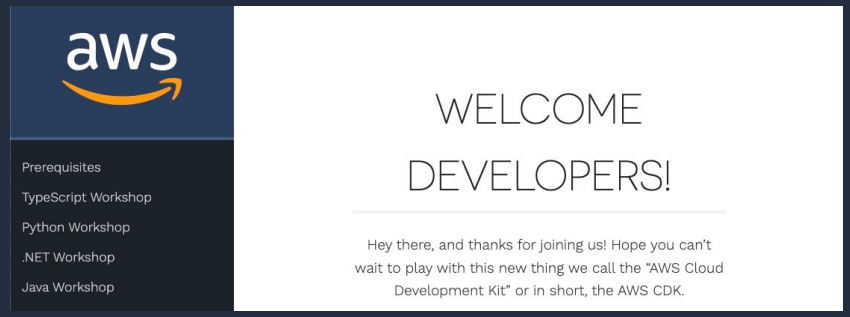
API Reference



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

API の仕様はこちらで確認

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, .NET, Java に対応

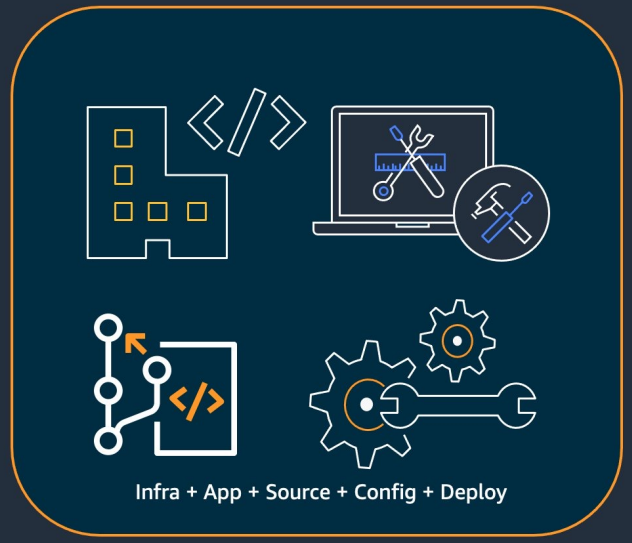
AWS CDK Blackbelt

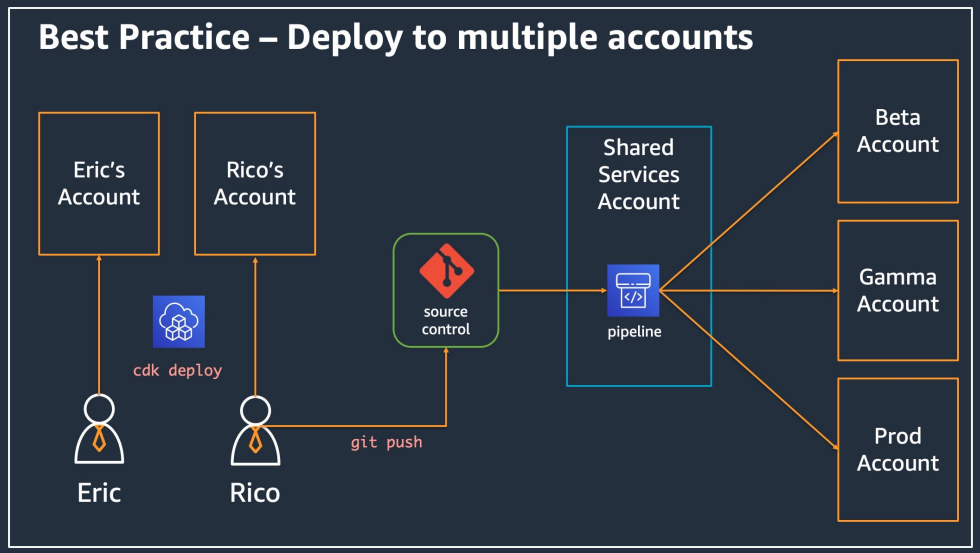
https://aws.amazon.com/jp/blogs/news/webinar-bb-aws-cloud-development-kit-cdk-2020/

CDK のベストプラクティス

AWS CDKでクラウドアプリケーションを開発するためのベストプラクティス

https://aws.amazon.com/jp/blogs/news/best-practices-for-developing-cloud-applications-with-aws-cdk/





Thank you!