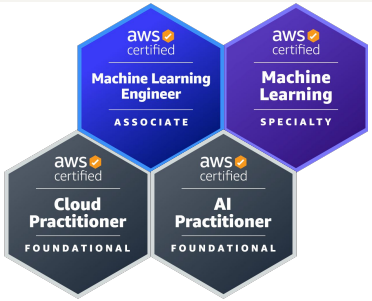
JAWS-UG 京都支部 × KyotoLT
AWSを活用したマルチテナントSaaSの設計と実践

CDKハンズオン

株式会社シーズ　中村 勇太



株式会社シーズ - クラウドソリューション事業部

中村 勇太 Yuta Nakamura

FROM Kyoto. lightning Talks

経歴

2024年3月 大阪電気通信大学 卒業

2024年4月 株式会社シーズ 入社

好きなAWSサービス

Amazon Lightsail / AWS Step Functions

目次

ハンズオン環境構築

IaC / CDKについて

CDKハンズオン

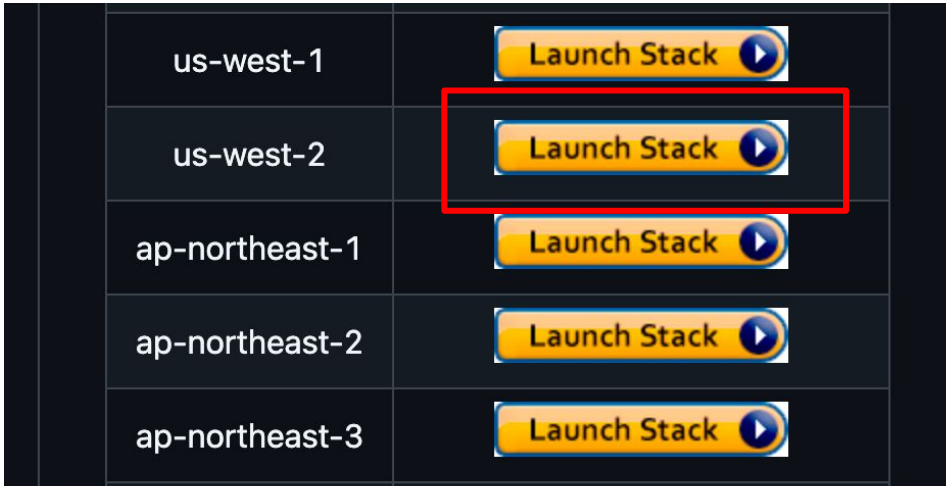
SaaSハンズオン環境構築（先にデプロイだけします！）

ハンズオン環境構築

ハンズオン環境構築　1/2

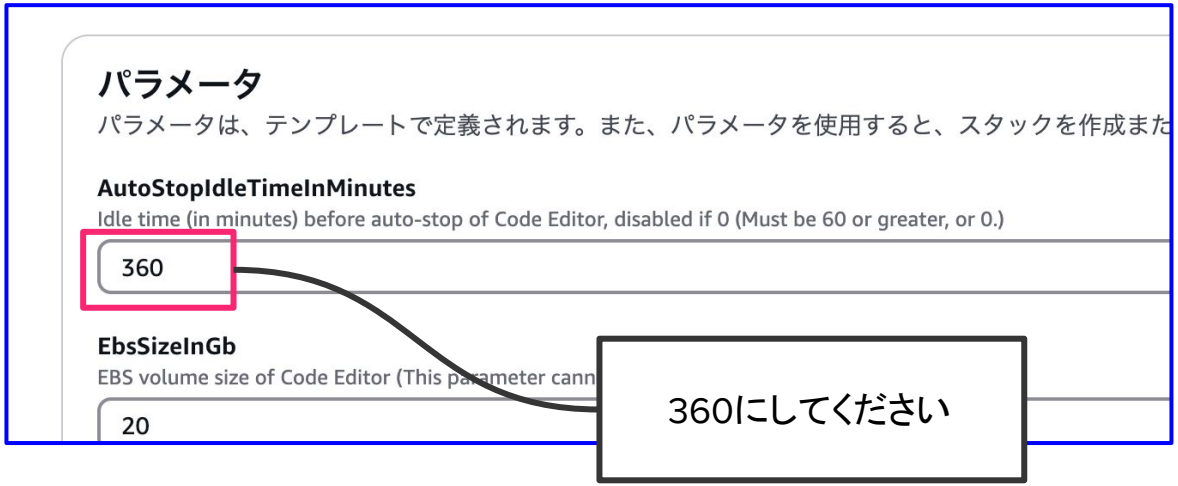
https://github.com/aws-samples/sagemaker-studio-code-editor-template

us-west-2のLaunch Stackをクリックする



参考：https://qiita.com/moritalous/items/656793cd97e1e967f045

ハンズオン環境構築　1/2



IaC / CDKとは

IaCとは 1/2

Infrastructure as Code（IaC）とは、インフラストラクチャの構成をコードとして管理するアプローチです。
手動でのインフラ構築やGUIツールを使った設定に代わり、テキストファイルやコードを使ってインフラの構成を定義します。

IaCとは 2/2

一般的に知られているEC2を作成する方法に、コンソールにログインして作成する方法がありますが、IaCを用いて作成することもできます。


```
{
  "version": "tree-0.1",
  "tree": {
    "id": "App",
    "path": "",
    "children": {
      "CdkSampleStack": {
        "id": "CdkSampleStack",
        "path": "CdkSampleStack",
        "children": {
          "Vpc": {
            "id": "Vpc",
            "path": "CdkSampleStack/Vpc",
            "children": {
              "Resource": {
                "id": "Resource",
                "path": "CdkSampleStack/Vpc/Resource",
                "attributes": {
                  "aws:cdk:cloudformation:type": "AWS::EC2::VPC",
                  "aws:cdk:cloudformation:props": {
```

IaCのメリット

- 再現性：同じコードから常に同じ環境が構築される
- バージョン管理：GitなどでIaCコードを管理が可能になる
- 自動化：デプロイメントプロセスの自動化が容易になる
- ドキュメント化：IaCコード自体がドキュメントとなる
- スケーラビリティ：環境の複製や拡張が容易

AWSにおけるIaC

AWS CloudFormation（CFn）
YAML/JSONでAWSリソースを記述、幅広くリソースを管理できる。

AWS SAM
CFnをベースに、AWS Lambda中心にサーバレスな構成を記述できる。

AWS CDK
コンストラクトによる部品化に加えて条件分岐、ループといった
プログラミングの恩恵を享受できる。

https://d2908q01vomqb2.cloudfront.net/da4b9237bacccdf19c0760cab7aec4a8359010b0/2017/08/09/aws_sam_introduction-1024x286.png

CDKとは

AWS CDK (Cloud Development Kit) は、プログラミング言語（TypeScript、Python、Java、C#など）を使ってAWSリソースを定義できるオープンソースのIaCフレームワークです。
CDKで書かれたコードは最終的にCFnに変換されてデプロイされます。

CDKのメリット

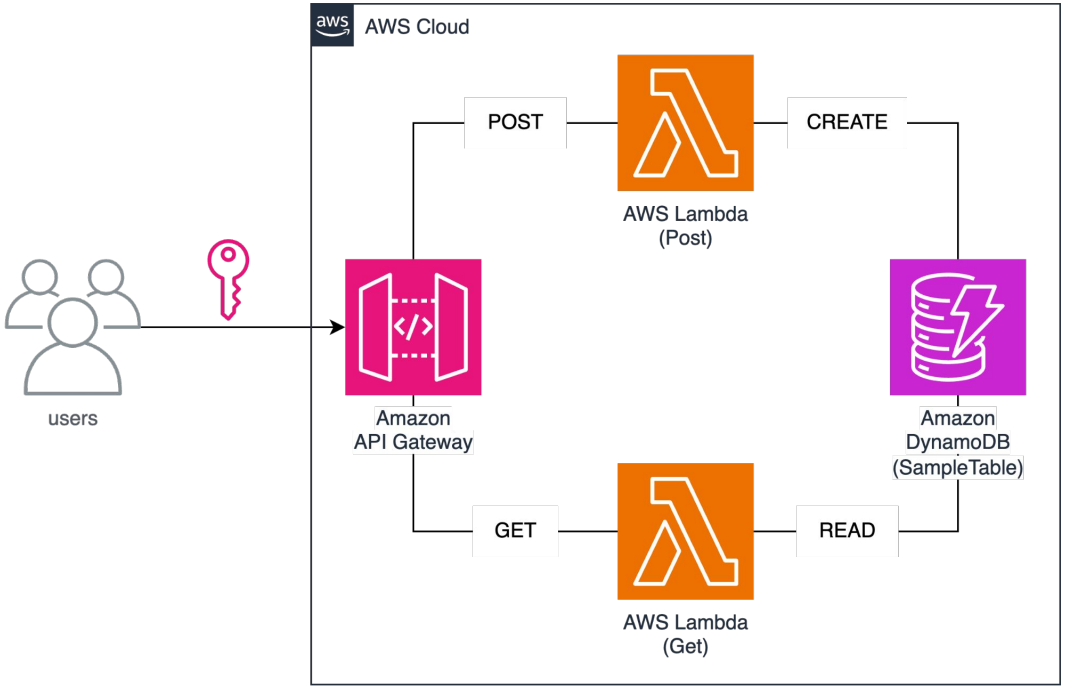
- プログラミング言語の利用：馴染みのある言語でAWSリソースを定義
- 型安全性：コンパイル時にエラーを検出（TypeScriptなど）
- 抽象化：高レベルの構成要素（Construct）を使用可能
- 再利用性：コンポーネントの再利用が容易
- IDE統合：コード補完やリファクタリング、AIエージェントの機能が使える

CDK ハンズオン

CDK触ったことある人 

CDKハンズオンの構成

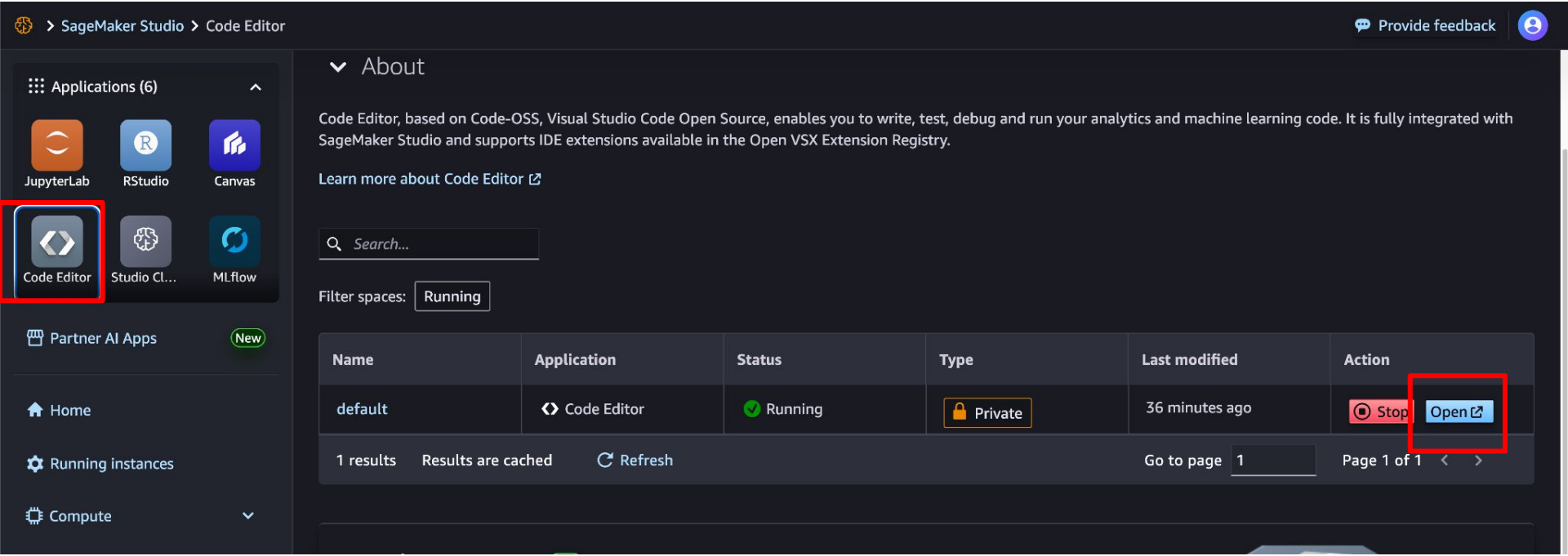
シンプルなCreateとReadができる
サーバレスなアプリケーション



ハンズオン環境へのアクセス　1/2

Amazon SageMaker AIのコンソールにアクセス

ハンズオン環境へのアクセス　2/2

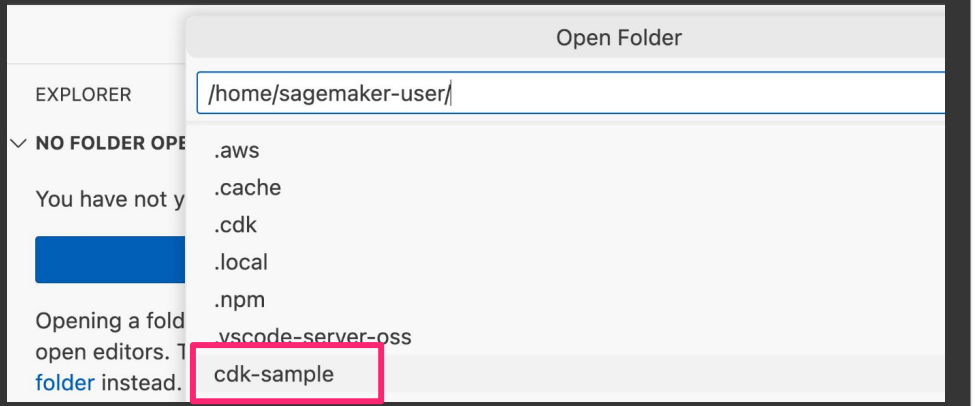


CDKハンズオンのコードの用意

> git clone https://github.com/papi-tokei/cdk-sample.git

Code Editorでフォルダを開く→

> cd cdk-sample
> npm i



リポジトリのファイル構成

```
> npx cdk bootstrap // CDK初めてだよという方は実行してください
```

```
├── README.md
├── bin
│   └── cdk-sample.ts
├── cdk.json
├── lambda
│   ├── get.py                  DBに読み取りをするコード
│   └── post.py                 DBに書き込みをするコード
└── lib
    └── cdk-sample-stack.ts     今回メインで見ていくコード
```

リポジトリのファイル構成

```
> npx cdk bootstrap // CDK初めてだよという方は実行してください
```

```
├── README.md
├── bin
│   └── cdk-sample.ts
├── cdk.json
├── lambda
│   ├── get.py          DBに読み取りをするコード
│   └── post.py         DBに書き込みをするコード
└── lib
    └── cdk-sample-stack.ts    今回メインで見ていくコード
```

さっそく見てみましょう

(git stashすればなんとかなります)

cdk-sample-stack.ts L11-16 - DynamoDB作成

```
// DynamoDBのテーブルを作成する
const table = new dynamodb.Table(this, 'SampleTable', {
  partitionKey: { name: 'id', type: dynamodb.AttributeType.STRING },
  tableName: 'SampleTable',
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});
```

cdk-sample-stack.ts L18-26, L38-40 - Lambda(Get)作成

```
// Lambda関数を作成する(GET用)
const getLambda = new lambda.Function(this, 'GetLambda', {
  runtime: lambda.Runtime.PYTHON_3_9,
  handler: 'get.handler',
  code: lambda.Code.fromAsset('lambda'),
  environment: {
    TABLE_NAME: table.tableName,
  },
});
=== 略 ===
// DynamoDBテーブルへのアクセス権限を Lambda関数に付与する
// GET Lambdaに対してテーブルの読み取り権限を付与
table.grantReadData(getLambda);
```

cdk-sample-stack.ts L44-51 - API Gateway作成 1/3

```
// API Gatewayを作成する
const api = new apigateway.RestApi(this, 'SampleApi', {
  restApiName: 'Sample Service',
  apiKeySourceType: apigateway.ApiKeySourceType.HEADER,
});

// APIキーを作成する
const apiKey = api.addApiKey('ApiKey');
```

cdk-sample-stack.ts L53-64 - API Gateway作成 2/3

```
// 使用量プランを作成し、APIキーと紐づける
const usagePlan = api.addUsagePlan('UsagePlan', {
  name: 'BasicUsagePlan',
  throttle: {
    rateLimit: 10,
    burstLimit: 2,
  },
});
usagePlan.addApiKey(apiKey);
usagePlan.addApiStage({
  stage: api.deploymentStage,
});
```

cdk-sample-stack.ts L66-70 - API Gateway作成 3/3

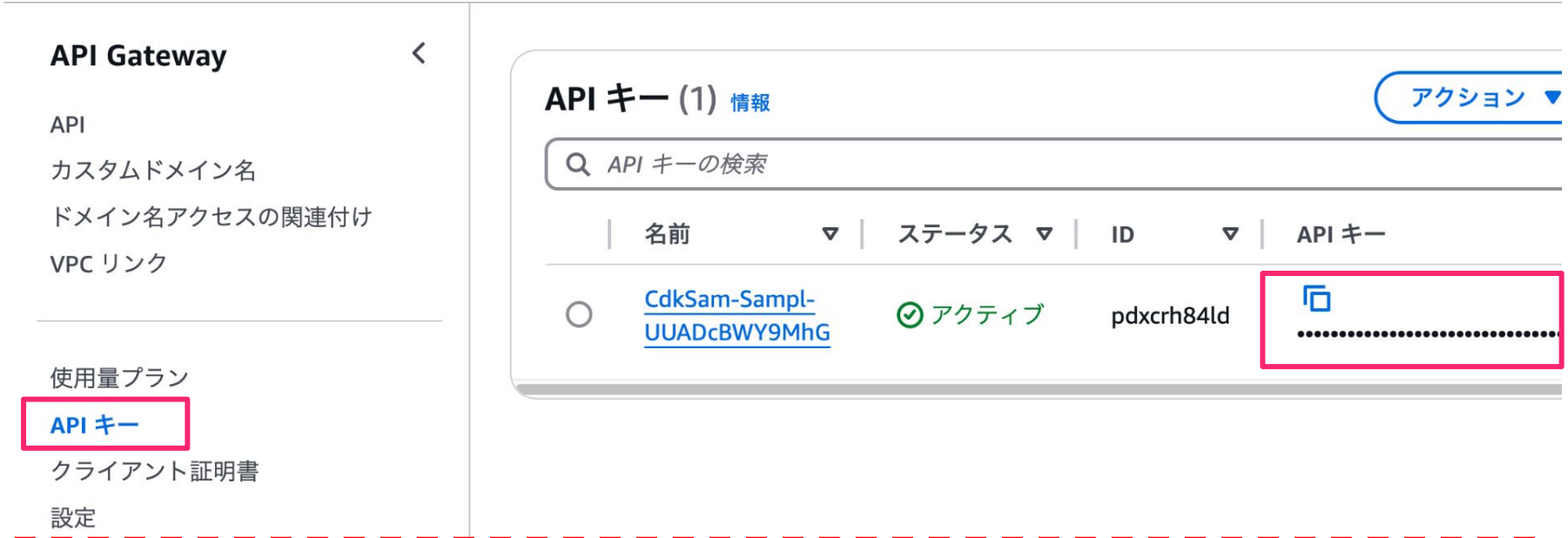
```
// API GatewayとLambda関数を紐づける
const getIntegration = new apigateway.LambdaIntegration(getLambda);
api.root.addMethod('GET', getIntegration, {
  apiKeyRequired: true, // Require API key
});
```

デプロイ

```
> npx cdk synth  // CFnテンプレートを生成
> npx cdk bootstrap // cdkが初めての方は実行してください(再掲)
> npx cdk deploy // デプロイ
この内容でデプロイしてOKか聞かれますので y で返答してください。
```

コンソールから確認

API キーのチェック



⚠️このAPIキーはこの後使用します ⚠️

APIエンドポイントのチェック

```
✅  CdkSampleStack

✨  Deployment time: 1.69s

Outputs:

CdkSampleStack.SampleApiEndpoint =
https://xxxxxxxxxx.execute-api.ap-northeast-1.amazonaws.com/prod/

Stack ARN:

arn:aws:cloudformation:ap-northeast-1:123456789012:stack/CdkSampleSt
ack/xxxxxxx-xxxx-xxxxxxx
```

動作チェック

```
> curl -X POST '<ここにエンドポイントを入れる>' \
    -H 'Content-Type: application/json' \
    -H 'x-api-key: <ここにAPIキーを入れる>' \
    -d '{"name":"TARO", "age":55}'

> curl -X GET '<ここにエンドポイントを入れる>' \
    -H 'x-api-key: <ここにAPIキーを入れる>'
```

EX) Lambdaのアーキテクチャを Graviton(ARM64)に 1/2

```
// Lambda関数を作成する(GET用)
const getLambda = new lambda.Function(this, 'GetLambda', {
  runtime: lambda.Runtime.PYTHON_3_9,
  handler: 'get.handler',
  architecture: lambda.Architecture.ARM_64, // ここを追加
  code: lambda.Code.fromAsset('lambda'),
  environment: {
    TABLE_NAME: table.tableName,
  },
});
```

GravitonはAmazonが独自設計したARMベースのプロセッサ
高いパフォーマンスと低コスト、低消費電力が特徴

EX) Lambdaのアーキテクチャを Graviton(ARM64)に 2/2

```
> npx cdk diff // 手元のCFnとデプロイされたCFnの差分を表示

Resources

[~] AWS::Lambda::Function GetLambda GetLambdaXXXXXX
 └─ [+] Architectures
     └─ ["arm64"]
[~] AWS::Lambda::Function PostLambda PostLambdaXXXXXX
 └─ [+] Architectures
     └─ ["arm64"]
```

CDKハンズオン環境の削除

```
> npx cdk destroy
```

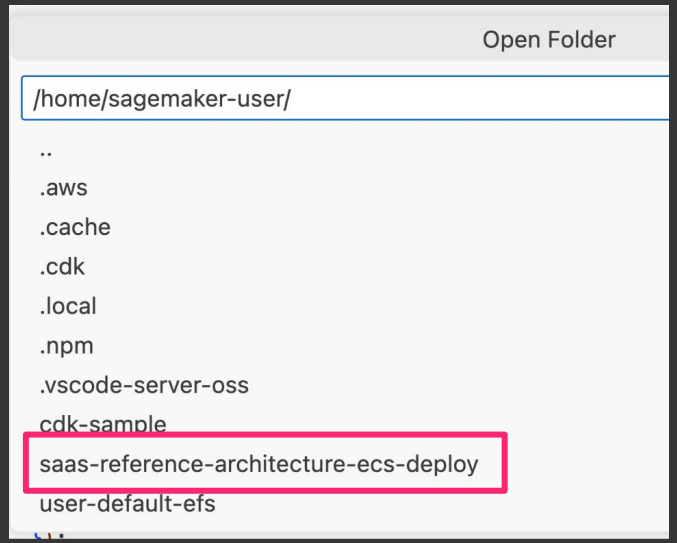
削除していいか聞かれますので y で返答してください。

SaaS ハンズオン環境構築

CDKハンズオンのコードを用意

> git clone https://github.com/papi-tokei/saas-reference-architecture-ecs-deploy.git

Code Editor上でフォルダを開く→



> cd saas-reference-architecture-ecs-deploy

> npm i

SaaSハンズオン環境構築 1/4

lib/saas-reference-architecture-ecs-deploy-stack.ts L28-34

```
  build: {
    commands: [
      'cd scripts',
      './build-application.sh',
      './install.sh <ここにメールアドレスを入れる >'
    ],
  },
```

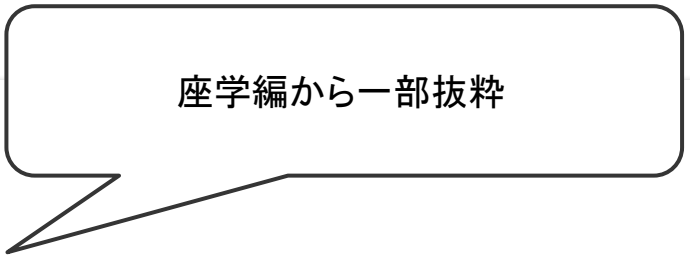
⚠️必ず受信できるメールアドレスを入力してください ⚠️

SaaSハンズオン環境構築 2/4

```
> npx cdk deploy

この内容でデプロイしてOKか聞かれますので y で返答してください。
```

デプロイ中にまとめ

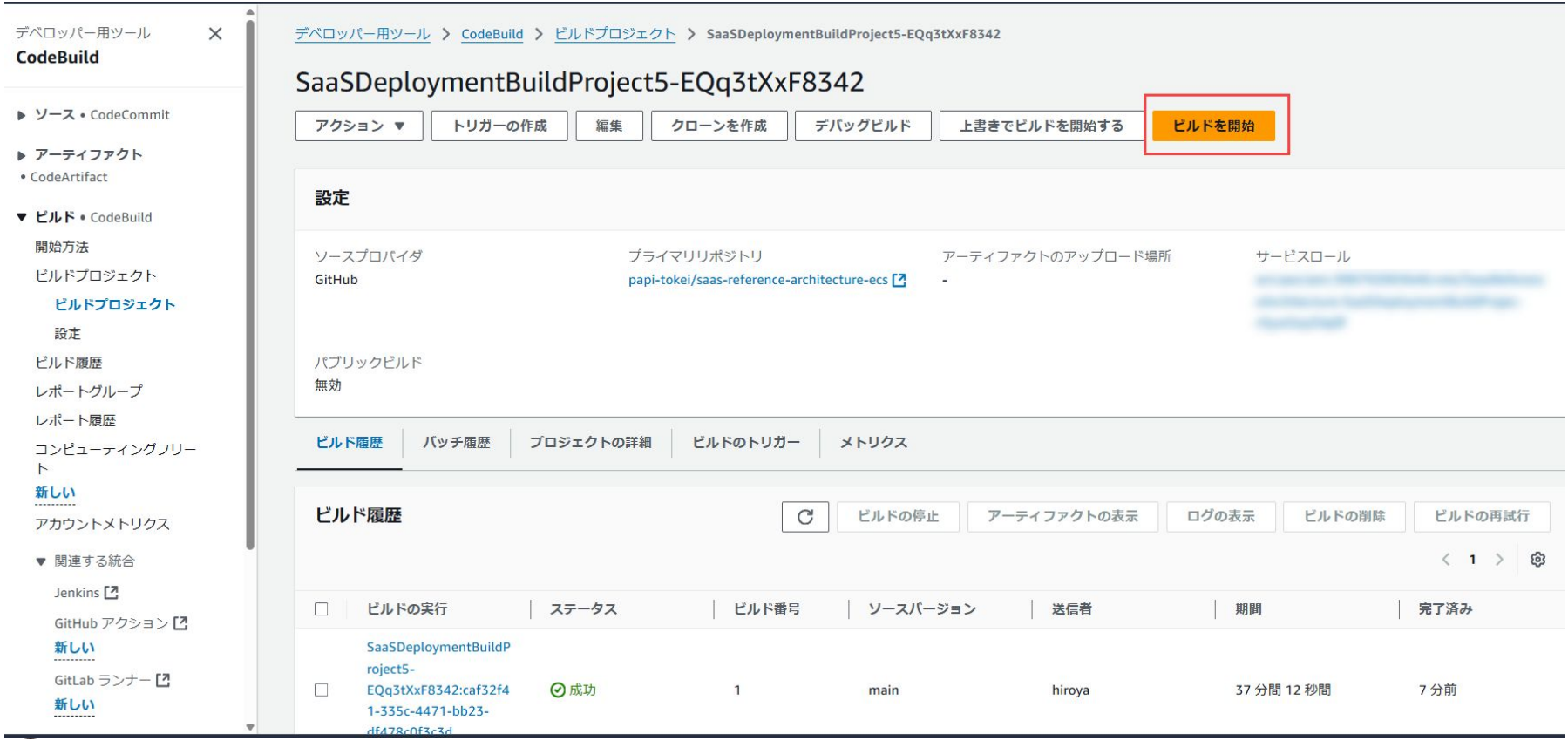


オンボーディング/オフボーディング

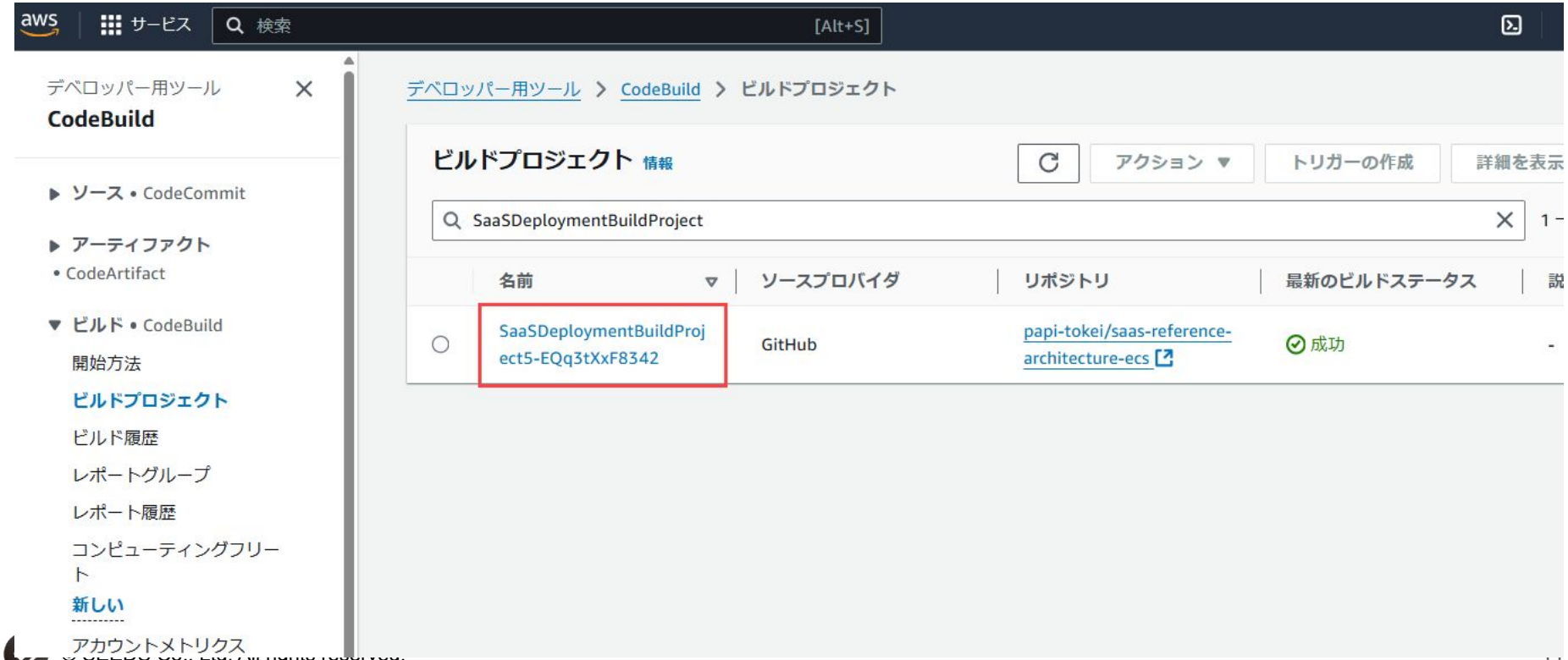
オンボーディングとは、新規顧客がサービスを使うことができるようになるまでの一連の作業のことです。

オンボーディングはできる限り自動化できることを目指す必要があります。

SaaSハンズオン環境構築 3/4



SaaSハンズオン環境構築 4/4



ありがとうございました！

コメントやワイワイは

#jawsug

でお待ちしております

[付録] cdk deployの裏で何が動いているのか