AWS CDK v2 ワークショップ

2022/02/24

シニアエバンジェリスト亀田

[はじめに]

AWS Cloud Development Kit (AWS CDK) v2 はクラウドインフラをコードで定義し、AWS CloudFormation でプロビジョニングするためのフレームワークです。2021 年 12 月に新しい Verson2 が一般提供開始となっています。このワークショップでは、Getting Started として英語で提供されている 2 つのハンズオンを再構成し日本語化したものです。手順正しく動作しない場合、英語版を参照ください。

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

2022 年 2 月 24 日現在対応している言語は以下の通りです。

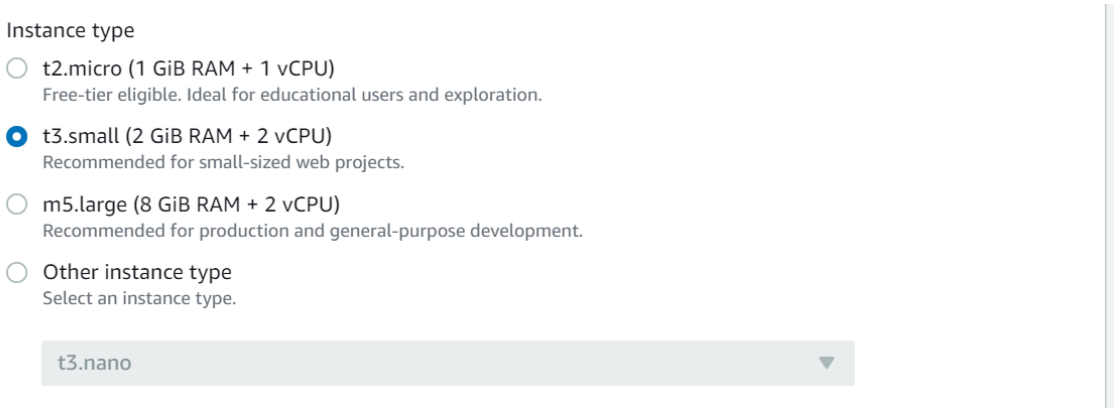
TypeScript、JavaScript、Python、Java、C#

このワークショップでは、JavaScript、TypeScript の順番で動作を確認していきます。

[Cloud9]の起動

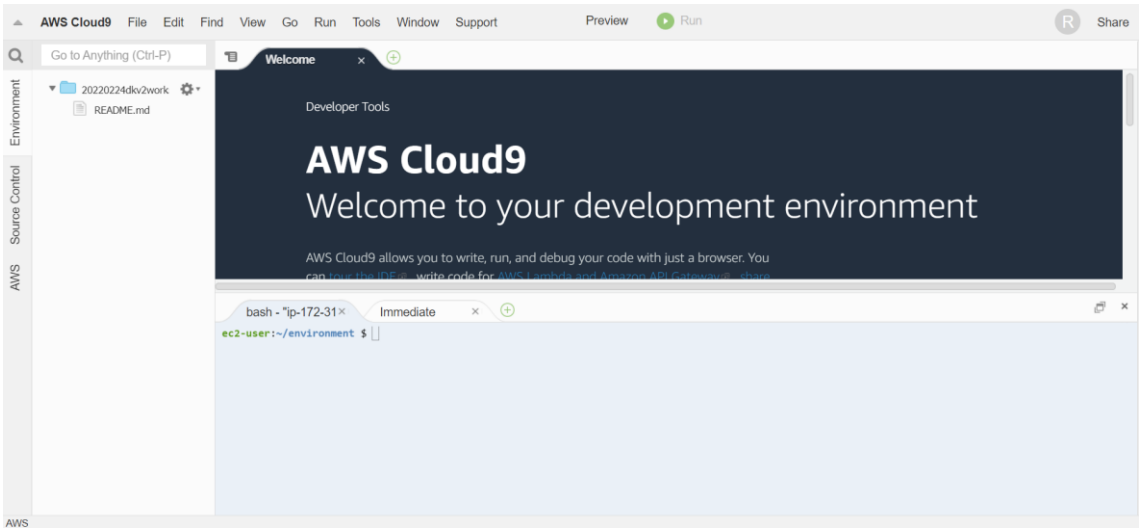
AWS Cloud9 は AWS が提供しているクラウド型 IDE のサービスです。

1. Cloud9 マネージメントコンソールにアクセスします（リージョンはどこでも問題ありません。作業は必ず、CDKv2 を用いていないリージョンを使ってください。また、CDKv2 の初期設定が完了している AWS アカウントで行わないことを推奨します。）
2. [Create environment]をおします
3. [Name]に適当な値をいれ、[Next Step]をおします
4. [Instance Type]に[t3.small]を選びます



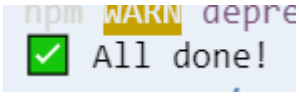
5. [network settings]で defaultVPC もしくは、任意の VPC の Public Subnet が指定されていることを確認し、[Next step]をおします

6. 確認画面で[Create environment]をおし、コンソールへアクセス可能となるまで数分間待ちます。
7. 以下の画面が起動すれば完了です。

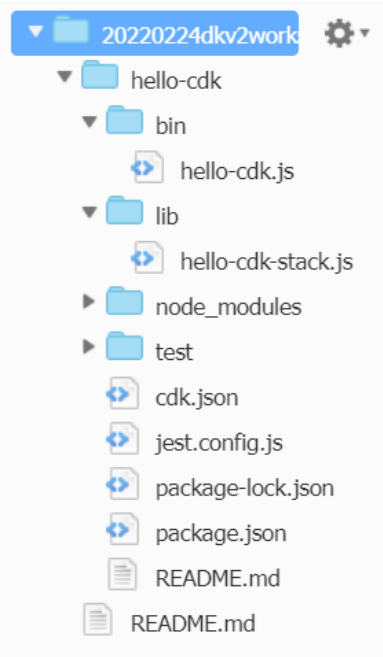


[CDK v2 の動作確認]

8. [cdk --version]を実行します。Cloud9 は CDK や CDK の実行に必要なモジュールなどがあらかじめインストールされています。別環境を用いる場合のインストール手順は、このページを参考にしてください。また、Cloud9 は起動時にマネージメントコンソールにログインしている IAM アカウント情報をもとに一時的に払い出された権限で動作しますが、別 IDE 環境を用いる場合、別途 IAM アカウントの作成と設定(aws configure 等)も必要です。
9. [commands.txt]の 1 番を実行し、作業ディレクトリを作成します
10. [commands.txt]の 2 番を実行し、JavaScript 用に CDK を初期化します。以下が表示されたら成功です。



11. 以下のようにいくつかのフォルダやファイルができています。



主要な設定ファイルをいくつか紹介します。

[package.json]

CDK v2 は CDK v1 と異なり AWS の各リソースを操作するモジュールが[aws-cdk-lib]としてまとまっているのが特徴です。v1 では AWS のリソース毎にモジュールを設定し依存関係が存在していたため、管理が煩雑でしたが、主要なリソースが一つのライブラリに統合され依存関係も管理しやすくなっています。

```
  "devDependencies": {
    "aws-cdk": "2.12.0",
    "jest": "^26.4.2"
  },
  "dependencies": {
    "aws-cdk-lib": "2.12.0",
    "constructs": "^10.0.0"
  }
```

現在 AWS IoT Core 等未対応の AWS サービスも存在しており、その場合明示的にコンストラクタというものを読み込むことで追加リソースを認識させることが可能です。[aws-cdk-lib]には順次安定版が統合された形でマージされていきます。コミュニティで作成された外部コンストラクタは以下のサイトから入手が可能です。

https://constructs.dev/

[bin/hello-cdk.js]

前述の package.json から呼び出されている[bin/hello-cdk.js]は以下のように{ HelloCdkStack }が'../lib/hello-cdk-stack'により定義されていることがわかります。定義されているスタックはコマンドラインの[cdk ls]コマンドで一覧の確認が可能で

す。

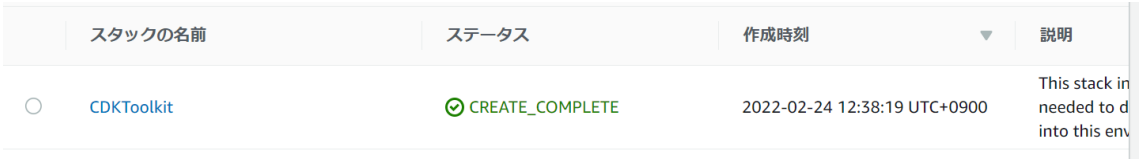
```
const cdk = require('aws-cdk-lib');
const { HelloCdkStack } = require('../lib/hello-cdk-stack');
```

[/lib/hello-cdk-stack]
実際に CDK から起動させる AWS リソースなどを定義するファイルです。現在は何も設定されておらず
次以降のハンズオンステップで設定を行っていきます。

12. [lib/hello-cdk-stack.js]をダブルクリックで開き、[commands.txt]の 3 番の内容に置換保存します。(置換のあと、タブを閉じると保存ダイアログが出てきます)
このスクリプトは以下を行います。
・'aws-cdk-lib/aws-s3 の読み込み
・MyFirstBucket という名前を含んだ S3 Bucket の作成
・バージョニングの有効化

13. [cdk synth]を実行します
14. このコマンドにより行った作業がローカルの CDK 環境に反映され、CloudFormation Template が生成されます。CloudFormation テンプレートに比べて CDK はかなりシンプルかつ可読性が高いことがわかります。
CDK には 2 種類のレイヤが存在します。このハンズオンで用いているのは L2 と言われている上位レイヤです。L2 レイヤでは、CloudFromation の様々なパラメーターが抽象化されておりより少ない設定で作業を行うことができます。ただし、L2 レイヤでは、CloudFormation で設定可能なすべてのパラメーターを設定できないケースがあります。この場合Ｌ1 レイヤという下位レイヤを用いることで設定が可能となりますが、その分設定項目が増えてしまいます。
Ｌ2 レイヤで設定可能なパラメーター一覧はこちらをご覧ください。
https://docs.aws.amazon.com/cdk/api/v2/docs/aws-construct-library.html

15. [cdk bootstrap]を実行します。このコマンドは CDK の初期設定に必須であり、リージョンごと、言語ごとに必ず初回作業時の実行が必要です。マネージメントコンソールで CloudFormation を確認すると以下のスタックが生成されています。



	スタックの名前	ステータス	作成時刻	説明
○	CDKToolkit	CREATE_COMPLETE	2022-02-24 12:38:19 UTC+0900	This stack in needed to d into this env

これは、CDK が動作に必要とする情報を保存する S3 バケットや IAM ロール等を含む CDK の動作環境構築を行うスタックです。

16. [cdk deploy]を実行すると、CloudFormation 経由で S3 バケットが作成されます

	スタックの名前	ステータス	作成時刻	説明
○	HelloCdkStack	CREATE_IN_PROGRESS	2022-02-24 12:43:04 UTC+0900	-

17. S3 バケットをマネージメントコンソールで見ると 2 つ出来ていることがわかります。1 つ目（下）が、CDK の bootstrap で作成されたもので、2 つ目（上）が今作成されたものです。

	名前	AWS リージョン	アクセス	作成日
○	hellocdkstack-myfirstbucketb8884501-57uccycp664o	米国西部 (オレゴン) us-west-2	オブジェクトは公開することができます	2022/02/24 12:43:16 PM JST
○	cdk-hnb659fds-assets-294963776963-us-west-2	米国西部 (オレゴン) us-west-2	非公開のバケットとオブジェクト	2022/02/24 12:38:33 PM JST

生成されたバケット名に文字列が修飾されていることがわかります。[lib/hello-cdk.js]の以下の部分により値が修飾されています。

```
4  class HelloCdkStack extends cdk.Stack {
5    constructor(scope, id, props) {
6      super(scope, id, props);
7
8      new s3.Bucket(this, 'MyFirstBucket', {
9        versioned: true
10     });
```

くわしくは以下のページをご覧ください。
https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/hello_world.html

18. では先ほど作成した S3 バケットのプロパティを少し変更します。[commands.txt]の 4 番の中身で、[lib/hello-cdk.js]の中身を置換し保存します。
19. [cdk diff]を実行します。以下のように変更点が出力されます

```
[~] AWS::S3::Bucket MyFirstBucket MyFirstBucketB8884501
  ├─ [+] Tags
  │    └─ [{"Key":"aws-cdk:auto-delete-objects","Value":"true"}]
  ├─ [~] DeletionPolicy
  │    ├─ [-] Retain
  │    └─ [+] Delete
  └─ [~] UpdateReplacePolicy
       ├─ [-] Retain
       └─ [+] Delete
```

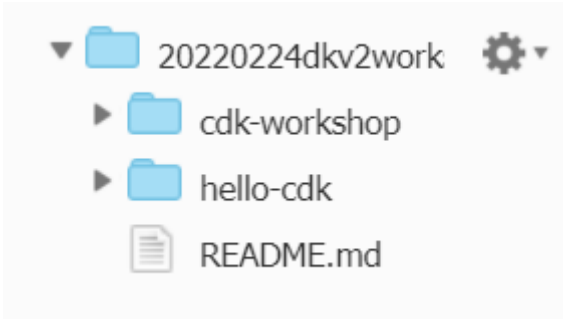
20. サイド[cdk deploy]を実行します。作業を進めていいか聞いてきますので[y + Enter]をおします。以下の通り Stack が Update 中となりますのでしばらく待ちます。

スタックをクリックし[テンプレート]タブを押すと、実行されている Yaml CloudFormation テンプレートを確認できます。

21. これにて Step1 が終わです。[cdk destroy]を実行し、環境を削除します。確認が求められますので[y + Enter]を入力します。
22. 削除が完了すると、CloudFormation のスタック画面からは先程存在していた、HelloCdkStack スタックが消えていることがわかります。S3 バケットも、削除されています。(CDK の動作そのものが必要とするバケットは残っています)

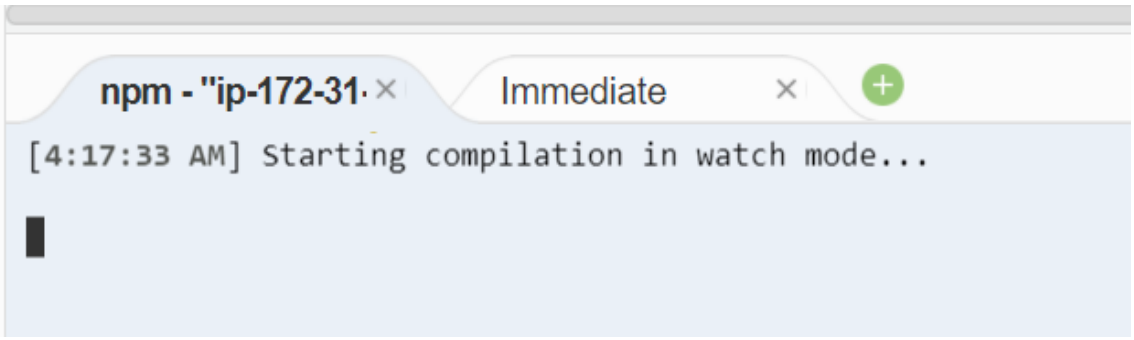
[TypeScript による Lambda の作成]
今までの手順では、JavaScript を用いて S3 バケットを操作しました。次のステップでは typescript を用いて、Lambda 関数を作成してみます。

23. [cd ..]を実行し、上の階層に移動します。
24. コマンド 5 番を実行します。階層構造は以下になります。



25. 今度は 6 番を実行し、TypeScript 用に init を実行します。今度は TypeScript をベースとして先程とほぼ同じフォルダ構造とファイルが展開されています。

26. TypeScript は JavaScript へ変更される必要があります。ファイルを変更するたびにそれがリアルタイムで行われるよう[npm run watch]コマンドを実行します。
27. そのターミナルを閉じないように、+ボタンで別のターミナルを開きます(New Terminal)（cdk-workshop のディレクトリに必ず移動して下さい）

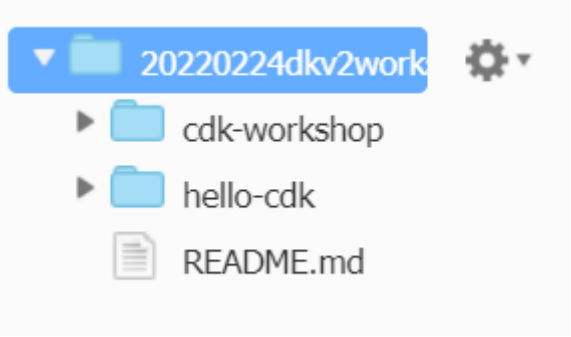


28. [lib/cdk-workshop-stack.ts]を見てみると、先程 s3 を指定していた個所に、デフォルトで SNS,SQS が設定されていることがわかります。

```
import { Duration, Stack, StackProps } from 'aws-cdk-lib';
import * as sns from 'aws-cdk-lib/aws-sns';
import * as subs from 'aws-cdk-lib/aws-sns-subscriptions';
import * as sqs from 'aws-cdk-lib/aws-sqs';
import { Construct } from 'constructs';
```

29. [cdk synth]を実行します。生成される CloudFormation テンプレートと CDV v2 設定ファイルを見比べてみてください。(オリジナルの英語版シナリオページは、CFn テンプレートの説明が CDK v1 ベースになっているので注意してください)
30. サイド[cdk bootstrap]を行います。(project 毎に bootstrap を行う必要はありませんが、今回言語を変更したため必要となります)
31. [cdk deploy]を実行します。実行を行うか確認されますので[y + Enter]をおします。以下の通り新しい CFn スタックが生成されていますので、完了まで待ちます。

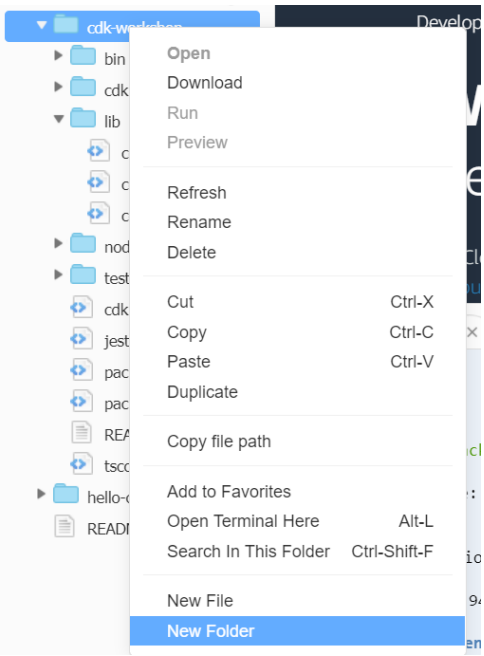
32. 実行が完了したら、SQS / SNS ができていることを確認してください。念のため Step1 で作成した環境の S3 バケットが作成されていないことを確認して下さい、フォルダ単位で別々のプロジェクトとして動作していることがわかります。



33. [lib/cdk-workshop-stack.ts]を開いて、コマンド 7 番の内容に置換します。保存した際に、先程実行した watch のターミナルが存続していれば自動で js ファイルも書き換わります。
34. [cdk diff]で差分をとります。中身は何も定義されていませんので、実行すると SNS や SQS が Destroy されると表示されています。

```
Resources
[-] AWS::SQS::Queue CdkWorkshopQueue50D9D426 destroy
[-] AWS::SQS::QueuePolicy CdkWorkshopQueuePolicyAF2494A5 destroy
[-] AWS::SNS::Subscription CdkWorkshopQueueCdkWorkshopStackCdkWorkshopTopicD7BE96438B5AD106 destroy
[-] AWS::SNS::Topic CdkWorkshopTopicD368A42F destroy
```

35. [cdk deploy]を実行します。実質今回の deploy は、最初に作成したものを削除するコマンドと同意義になります。
36. ではここから lambda 関数を作成していきます。まず bin や lib と同じ階層に[lambda]というフォルダを作成します



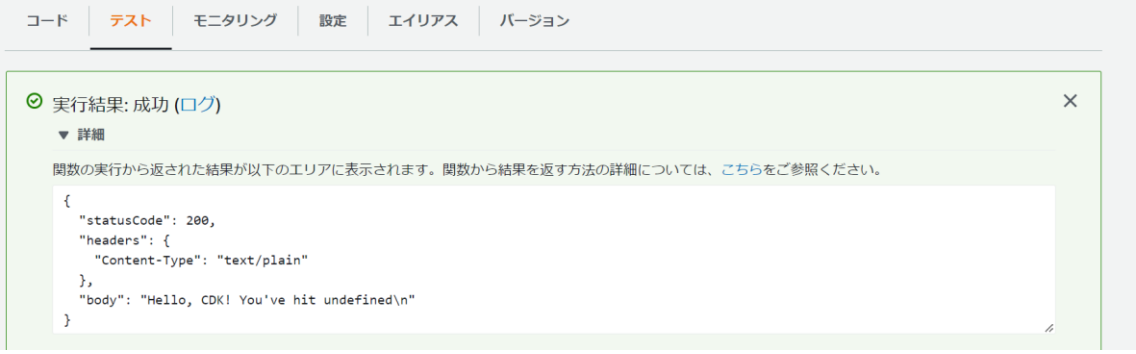
37. コマンド 8 番の内容をコピペして、[hello.js]を作成します。
38. この後英語版手順では Lambda コンストラクトライブラリの定義となりますが、CDK v2 では宣言のみでありこの手順は不要です。[lib/cdk-workshop-stack.ts]の内容をコ

マンド 9 番で置換し保存します

39. [cdk diff]を実行し差分を確認します。新しく Lambda 関数と IAM ロールが追加されることがわかります。

```
Resources
[+] AWS::IAM::Role HelloHandler/ServiceRole HelloHandlerServiceRole11EF7C63
[+] AWS::Lambda::Function HelloHandler HelloHandler2E4FBA4D
```

40. [cdk deploy]で実行します。確認が求められますので[y + Enter]をおします
41. Lambda 関数ができていますので、クリックし、テストタブからテスト実行をしてみてください。



[Hot Swap を用いたより素早い Deploy]

先程の手順では人によってばらつきはありますが、関数の Deploy に数十秒かかっています。

```
✅ CdkWorkshopStack

✨ Deployment time: 67.61s

Stack ARN:
arn:aws:cloudformation:us-west-2:294963776963:stack/CdkWorkshopStack/26001f30-952a-11ec-9a07-0afbe269e2c7

✨ Total time: 81.44s
```

ここからの手順では緊急時などにてで　関数を書き換えずに済むような、より高速なデプロイを試します

42. [hello.js]の中身をコマンド 10 番で置換します。(先ほどとテスト時に表示されるメッセージが異なるだけです。重要なのは差分を意図的に存在させ Deploy が動作する状態を作ることです)
43. コマンド 11 番を実行し、hot swap deploy を行います。デプロイが先程よりはかなり高速に終了していることがわかります。

```
✅  CdkWorkshopStack

✨  Deployment time: 2.38s

Stack ARN:
arn:aws:cloudformation:us-west-2:294963776963:stack/CdkWorkshopStack/26001f30-952a-11ec-9a07-0afbe269e2c7

✨  Total time: 16.68s
```

44. 次に Lambda 関数を呼び出せる API Gateway を作成します。[lib/cod-workshop-stack.ts]の中身を再度コマンド 12 番に置換して保存します。
45. [cdk diff]で差分を確認します
46. [cdk deploy]を実行します。途中確認が求められますので[y + Enter]をおします
47. 生成された URL をブラウザで開くと Lambda のテストメッセージが出てきます。

```
✨  Deployment time: 82.97s

Outputs:
CdkWorkshopStack.Endpoint8024A810 = https://bs4aebg0xd.execute-api.us-west-2.amazonaws.com/prod/
Stack ARN:
arn:aws:cloudformation:us-west-2:294963776963:stack/CdkWorkshopStack/26001f30-952a-11ec-9a07-0afbe269e2c7
```

48. 最後に[cdk destroy]を実行して環境を削除しておきます。

おつかれさまでした！削除は以下を行ってください。

・CFn スタック（CDKToolkit と Cloud9）
・S3 バケット
・CloudWatch Logs（”CDK”で検索）