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。

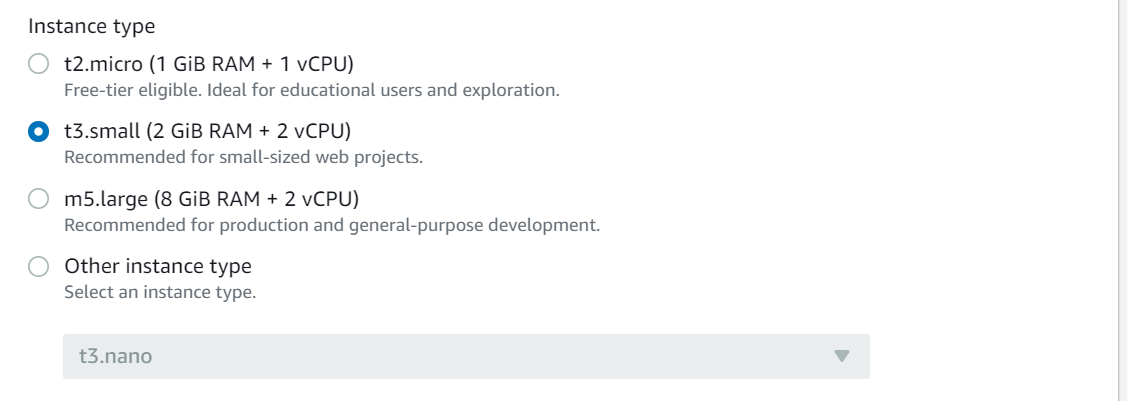
TypeScript、JavaScript、Python、Java、C#

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。

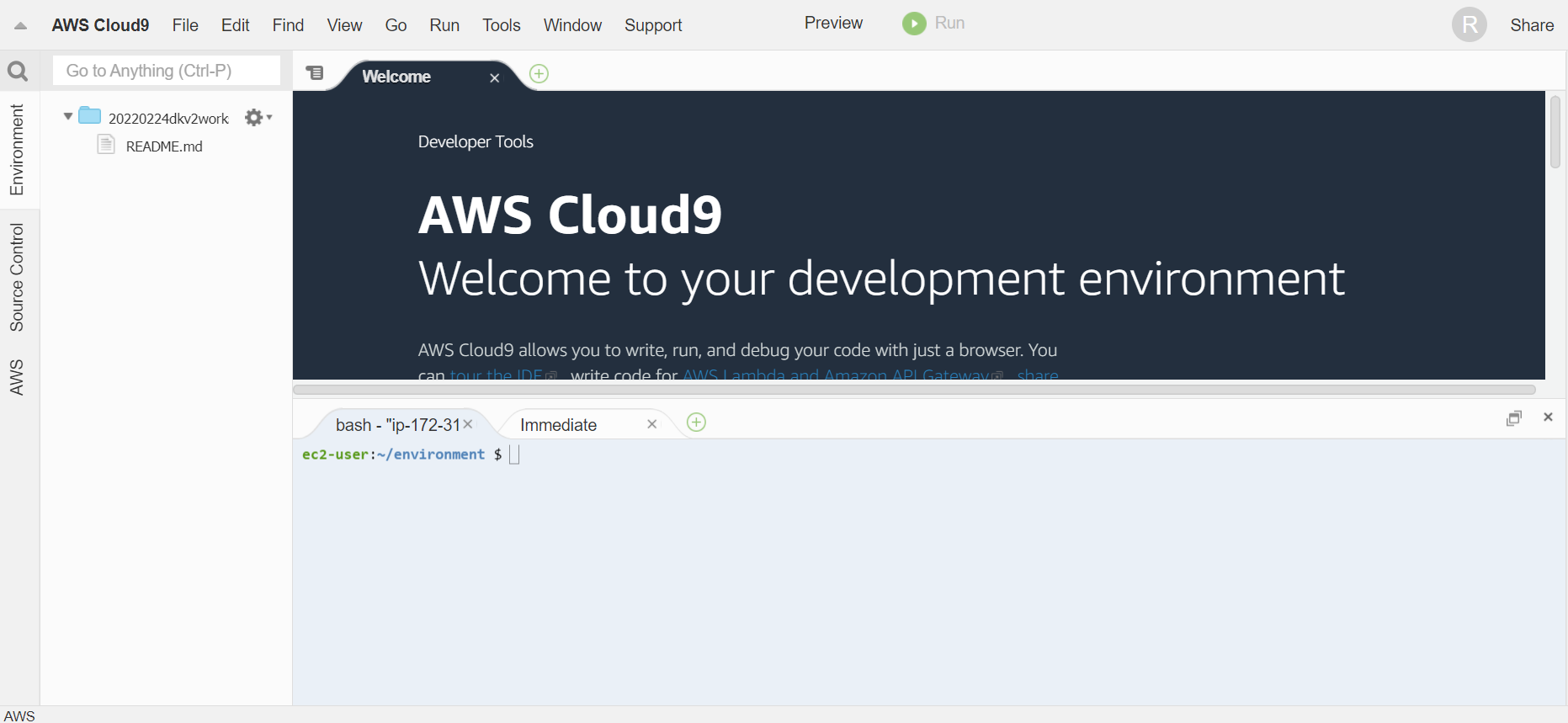
[Cloud9]の起動

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



1. [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
2. 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。
3. 以下の画面が起動すれば完了です。

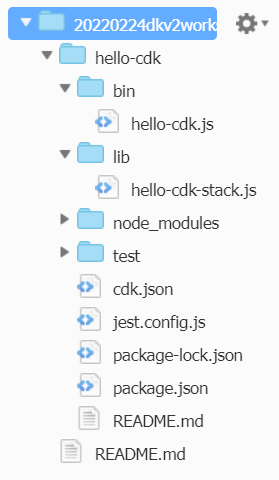


[CDK v2の動作確認]

1. [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](https://docs.aws.amazon.com/ja_jp/cdk/v2/guide/getting_started.html#getting_started_install)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。
2. [commands.txt]の1番を実行し、作業ディレクトリを作成します
3. [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。



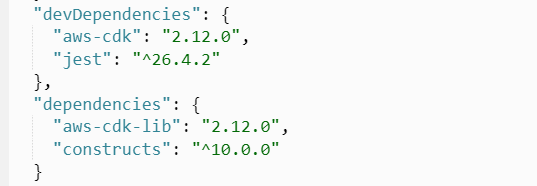
1. 以下のようにいくつかのフォルダやファイルができています。



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。

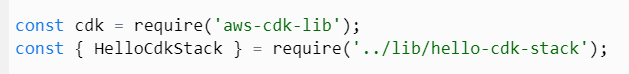


現在AWS IoT Core等未対応のAWSサービスも存在しており、その場合明示的にコンストラクタというものを読み込むことで追加リソースを認識させることが可能です。[aws-cdk-lib]には順次安定版が統合された形でマージされていきます。コミュニティで作成された外部コンストラクタは以下のサイトから入手が可能です。

<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。



[/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。

1. [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）

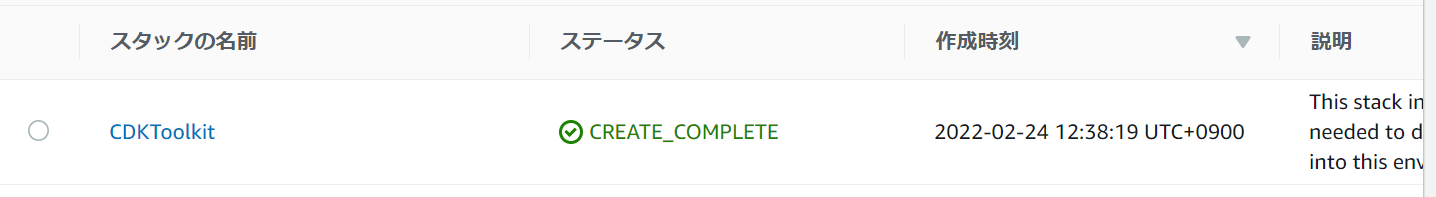
このスクリプトは以下を行います。

・'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

1. [cdk synth]を実行します
2. 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。
3. [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。



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。

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

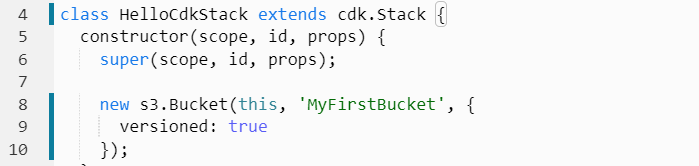


1. S3バケットをマネージメントコンソールで見ると2つ出来ていることがわかります。

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。



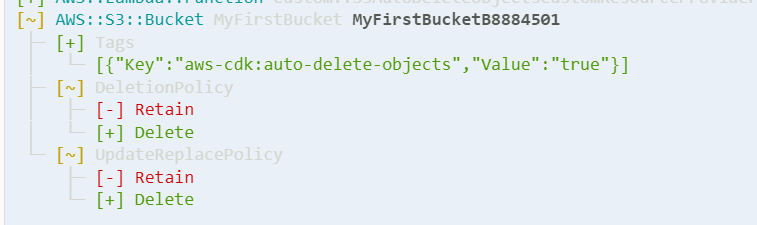
生成されたバケット名に文字列が修飾されていることがわかります。[lib/hello-cdk.js]の以下の部分により値が修飾されています。



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>

1. 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。
2. [cdk diff]を実行します。以下のように変更点が出力されます



1. 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。



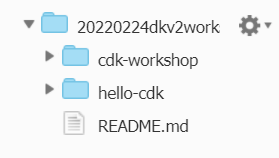
スタックをクリックし[テンプレート]タブを押すと、実行されているYaml CloudFormationテンプレートを確認できます。

1. 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。
2. 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）

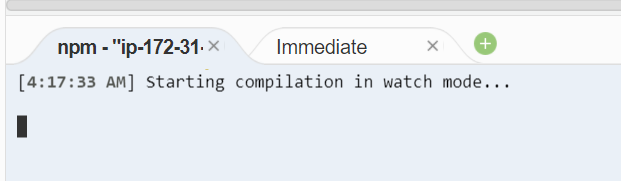
[TypeScriptによるLambdaの作成]

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。

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



1. 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。
2. 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。
3. 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）



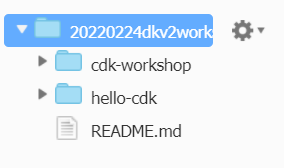
1. [lib/cdk-workshop-stack.ts]を見てみると、先程s3を指定していた個所に、デフォルトでSNS,SQSが設定されていることがわかります。



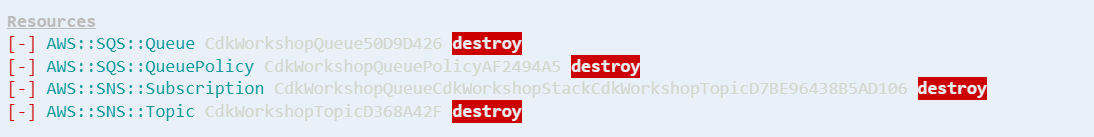
1. [cdk synth]を実行します。生成されるCloudFormationテンプレートとCDV v2 設定ファイルを見比べてみてください。（オリジナルの英語版シナリオページは、CFnテンプレートの説明がCDK v1ベースになっているので注意してください）
2. サイド[cdk bootstrap]を行います。（project毎にbootstrapを行う必要はありませんが、今回言語を変更したため必要となります）
3. [cdk deploy]を実行します。実行を行うか確認されますので[y + Enter]をおします。以下の通り新しいCFnスタックが生成されていますので、完了まで待ちます。



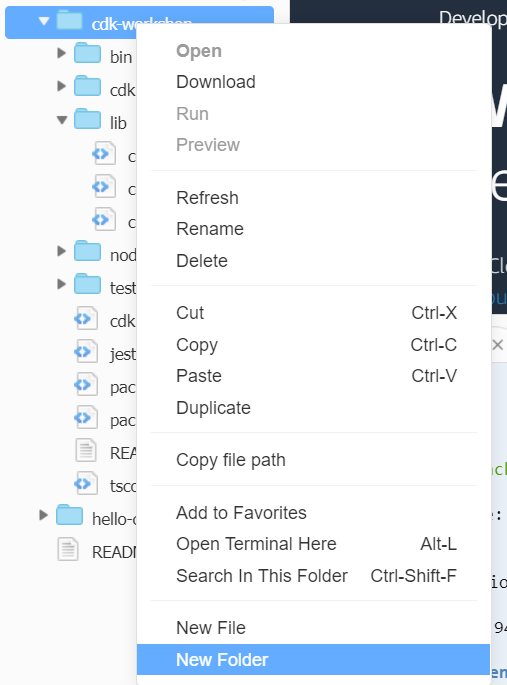
1. 実行が完了したら、SQS / SNS ができていることを確認してください。念のためStep1で作成した環境のS3バケットが作成されていないことを確認して下さい、フォルダ単位で別々のプロジェクトとして動作していることがわかります。



1. [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。
2. [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。



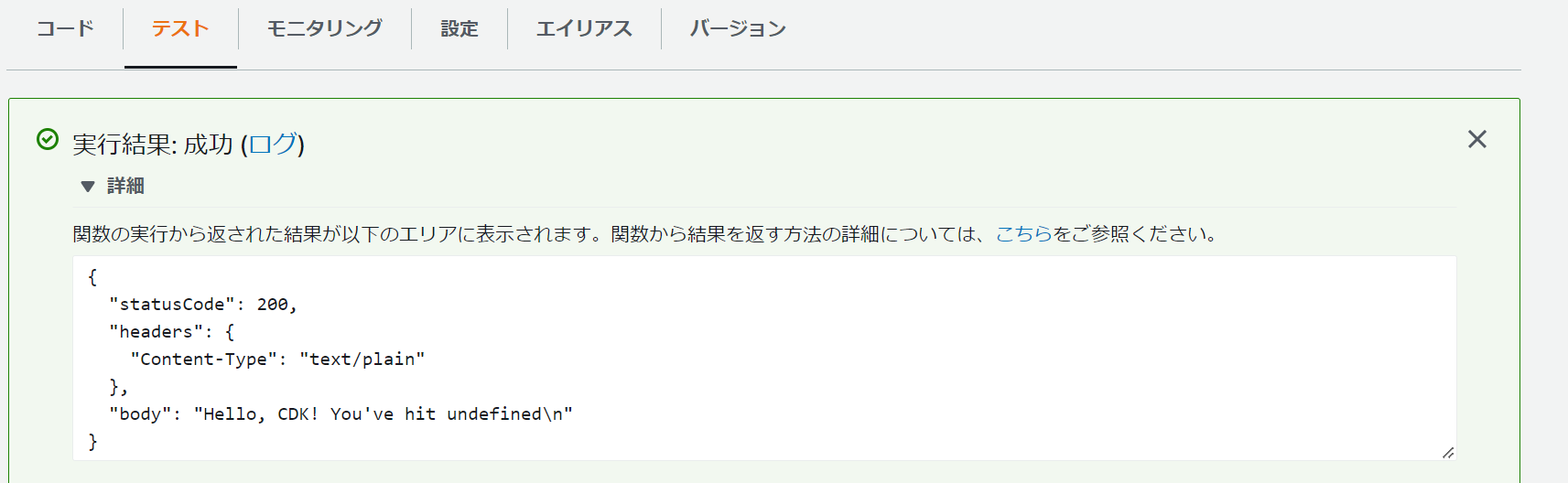
1. [cdk deploy]を実行します。実質今回のdeployは、最初に作成したものを削除するコマンドと同意義になります。
2. ではここからlambda関数を作成していきます。まずbinやlibと同じ階層に[lambda]というフォルダを作成します



1. コマンド8番の内容をコピペして、[hello.js]を作成します。
2. 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
3. [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。

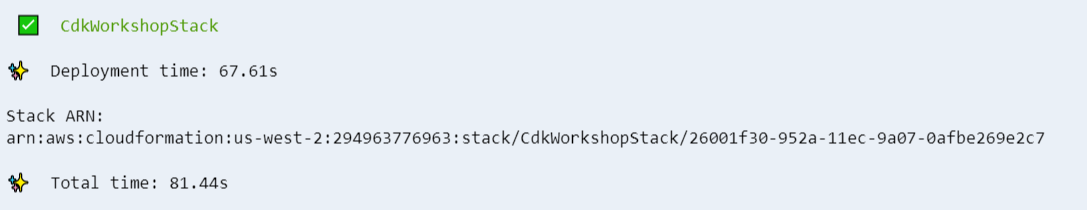


1. [cdk deploy]で実行します。確認が求められますので[y + Enter]をおします
2. Lambda関数ができていますので、クリックし、テストタブからテスト実行をしてみてください。



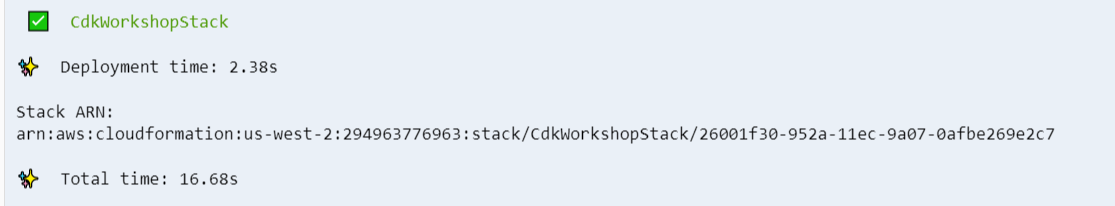
[Hot Swapを用いたより素早いDeploy]

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。



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

1. [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）
2. 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。



1. 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。
2. [cdk diff]で差分を確認します
3. [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
4. 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。

