CDK for Terraformの概要と特徴

ビヨンド社内勉強会#36

2022/09/30 株式会社ビヨンド 寺岡 佑樹

CDK for Terraformとは

- **CDK(Cloud Development Kit)のTerraform版**
  - **CDK = 使い慣れたプログラミング言語を使用してクラウドアプリケーションリソースを定義するためのオープンソースのソフトウェア (プログラミング言語でIaCできる)**
- **HashicorpとAWS CDKチームが共同開発した**
- **2022/08/12にGAになった**

CDK for Terraformの特徴

- CDKの実行環境としてTerraformを利用できる
  - = Terraformのラッパーツール(AWS CDKはCloudFormationを利用している)
- 馴染みのあるプログラミング言語を用いて記述できる
  - Typescript / Python / Java / C# / Go
  - AWS CDKとほぼ同じ書き味でセットアップできる (=HCLを書かなくても良い)
- マルチプロバイダに対応
  - AWS以外のクラウドプラットフォームの構築
  - Datadogなどのモニタリングサービスの設定
- AWS CDK と Terraformのハイブリッド

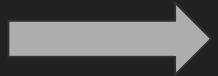
初期設定

- 前提条件
  - Terraform CLI(v1.1以上)
  - Node.jsとnpm(v16以上)
- インストール
  - npm install --global cdktf-cli@latest
- 初期化
  - cdktf init --template=go --local
  - 指定した言語に必要なファイル一式を用意してくれる

コード記述

```
provider "aws" {
 region = "ap-northeast-1"
}

resource "aws_instance" "compute" {
 ami           = "ami-078296f82eb463377"
 instance_type = "t3.micro"
}
```



- HCLを意識することなくGolangで記述できる
- コードの行数的に言えば HCLの方が短い

```
package main

import (
    "github.com/aws/constructs-go/constructs/v10"
    "github.com/hashicorp/terraform-cdk-go/cdktf"
    "github.com/aws/jsii-runtime-go"
    "cdk.tf/go/stack/generated/hashicorp/aws"
    "cdk.tf/go/stack/generated/hashicorp/aws/ec2"
)

func NewMyStack(scope constructs.Construct, id string) cdktf.TerraformStack {
    stack := cdktf.NewTerraformStack(scope, &id)

    aws.NewAwsProvider(stack, jsii.String("aws"), &aws.AwsProviderConfig{
        Region: jsii.String("ap-northeast-1"),
    })

    // The code that defines your stack goes here
    ec2.NewInstance(stack, jsii.String("compute"), &ec2.InstanceConfig{
        Ami:          jsii.String("ami-078296f82eb463377"),
        InstanceType: jsii.String("t3.micro"),
    })

    return stack
}

func main() {
    app := cdktf.NewApp(nil)

    NewMyStack(app, "cdk-for-terraform")

    app.Synth()
}
```

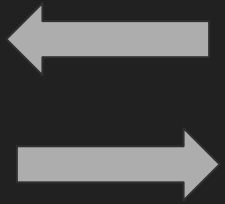
記述内容の反映

- cdktfの実行
  - cdktf deploy
  - 実行時の出力もterraform plan / applyと同様
  - Terraform & HCLに慣れている人だとわかりやすい
    - AWS CDKに慣れている人は実行エラーになると辛いかも
    - Terraformの使い方を覚えておくのがベスト

コードの相互運用

```
provider "aws" {
 region = "ap-northeast-1"
}

resource "aws_instance" "compute" {
 ami           = "ami-078296f82eb463377"
 instance_type = "t3.micro"
}
```



- HCL ⇔ CDKTFを変換するコマンドがある
  - cdktf synth
  - cdktf convert
- HCLをCDKTFに統合したい場合に有用

```
package main

import (
    "github.com/aws/constructs-go/constructs/v10"
    "github.com/hashicorp/terraform-cdk-go/cdktf"
    "github.com/aws/jsii-runtime-go"
    "cdk.tf/go/stack/generated/hashicorp/aws"
    "cdk.tf/go/stack/generated/hashicorp/aws/ec2"
)

func NewMyStack(scope constructs.Construct, id string) cdktf.TerraformStack {
    stack := cdktf.NewTerraformStack(scope, &id)

    aws.NewAwsProvider(stack, jsii.String("aws"), &aws.AwsProviderConfig{
        Region: jsii.String("ap-northeast-1"),
    })

    // The code that defines your stack goes here
    ec2.NewInstance(stack, jsii.String("compute"), &ec2.InstanceConfig{
        Ami:          jsii.String("ami-078296f82eb463377"),
        InstanceType: jsii.String("t3.micro"),
    })

    return stack
}

func main() {
    app := cdktf.NewApp(nil)

    NewMyStack(app, "cdk-for-terraform")

    app.Synth()
}
```

AWS CDKとの比較

```
func NewMyStack(scope constructs.Construct, id string)
cdktf.TerraformStack {
    stack := cdktf.NewTerraformStack(scope, &id)

    aws.NewAwsProvider(stack, jsii.String("aws"), &aws.AwsProviderConfig{
        Region: jsii.String("ap-northeast-1"),
    })

    // The code that defines your stack goes here
    ec2.NewInstance(stack, jsii.String("compute"), &ec2.InstanceConfig{
        Ami:          jsii.String("ami-078296f82eb463377"),
        InstanceType: jsii.String("t3.micro"),
    })

    return stack
}
```

- 両方ともGolangで記述できる
  - 同じコードを適用できない場面がある
  - AWS CDKで利用できるメソッドがある
- Terraformは暗黙的なデフォルト値が存在する
  - VPCを指定しない場合はデフォルトの VPCを利用
  - AWS CDKは明示的に指定しないとエラーになる
  - その点でコード行数的にいくと CDKTFの方が短い

```
type AwsCdkStackProps struct {
    awscdk.StackProps
}

func NewAwsCdkStack(scope constructs.Construct, id string, props
*AwsCdkStackProps) awscdk.Stack {
    var sprops awscdk.StackProps
    if props != nil {
        sprops = props.StackProps
    }
    stack := awscdk.NewStack(scope, &id, &sprops)

    // The code that defines your stack goes here

    vpc := awsec2.Vpc_FromLookup(stack, jsii.String("default"),
&awsec2.VpcLookupOptions{
        IsDefault: jsii.Bool(true),
    })

    awsec2.NewInstance(stack, jsii.String("compute"),
&awsec2.InstanceProps{
        MachineImage:
awsec2.NewAmazonLinuxImage(&awsec2.AmazonLinuxImageProps{
            Generation: awsec2.AmazonLinuxGeneration_AMAZON_LINUX_2,
        }),
        InstanceType: awsec2.NewInstanceType(jsii.String("t3.micro")),
        Vpc: vpc,
    })

    return stack
}
```

マルチプロバイダ

```
func NewMyStack(scope constructs.Construct, id string) cdktf.TerraformStack {
    stack := cdktf.NewTerraformStack(scope, &id)

    // AWS
    aws.NewAwsProvider(stack, jsii.String("aws"), &aws.AwsProviderConfig{
        Region: jsii.String("ap-northeast-1"),
    })

    ec2.NewInstance(stack, jsii.String("compute"), &ec2.InstanceConfig{
        Ami:          jsii.String("ami-078296f82eb463377"),
        InstanceType: jsii.String("t3.micro"),
    })

    // Google
    google.NewGoogleProvider(stack, jsii.String("google"), &google.GoogleProviderConfig{
        Zone:    jsii.String("asia-northeast1"),
        Project: jsii.String("cdktf-test"),
    })

    google.NewContainerCluster(stack, jsii.String("cluster"), &google.ContainerClusterConfig{
        Name:                  jsii.String("cluster"),
        RemoveDefaultNodePool: jsii.Bool(true),
        InitialNodeCount:      jsii.Number(1),
    })

    return stack
}
```

- **複数のクラウドプラットフォームを操作できる**
  - **AWS CDKではこれができない (AWS Only)**

利用シーン

- 初期構築
  - HCLではなくアプリ開発と同様の言語を利用して IaCをしたい
    - CDKTFの開発前にこのニーズがあった模様
  - バックエンド開発をしている方にはとても便利だと思われる
  - アプリ開発 / インフラが良い意味で分担できている場合
    - Terraform + HCLで事足りる部分も多そう(どちらを使うかを選択できる余地がある)
- 運用
  - マルチクラウドにおけるインフラの機能拡張
    - AWS CDKを元々使っていたがGCPのサービスも利用したい
      - Terraform + HCLに加えてCDKTFという選択肢が出来た
      - CDKのみでIaCを完結できる
        - Terraform + HCLで書いていたものをCDKTFに統合

懸念点

- CDKTF用のConstructがまだまだ少ない
  - 特にAWS CDKの L3 コンストラクト(Patterns) 相当のものがほぼない
    - L3 コンストラクト = 抽象度が最も高い (ECS使うなら大体この構成だよね)
    - 利用することでAWSのベストプラクティスに沿った構成が構築できる
  - 基本1から記述していくしかない
- AWS CDK用のConstructを使う方法が辛い
  - 通常では使えないため AWS Adapterと呼ばれるものをコード内に組み込む必要がある
  - AWS Adapter自体がまだプレビュー版
- 参考情報が少ない
  - あったとしても大体 Typescript
  - CDKTF用のConstructが充実してくるともっと便利になりそう

終わり