SRA の技術情報誌
GS LetterNeo
Vol.157 2021 年 8 月

プログラミングでインフラ構築 ~AWS CDK で始める IaC~

土田 拓実

DX 技術本部 DX 技術開発室

はじめに

AWS CDK(AWS Cloud Development Kit)[1]は AWS が提供している AWS リソースを構築するフレームワークです。AWS CDK を用いることでプログラミングライクに AWS リソースの構築が行えます。流れとしては、ユーザが作成したソースコードから CloudFormation の template を生成・Stack 化を行い、AWS 上にデプロイします。

本稿では AWS CDK を用いて TypeScript で簡単な構築を行ってみたいと思います。AWS CDK を使用した IaC のインフラ構築の概要をつかんでいただければと思います。

※一部課金が発生します。想定外の請求が発生しないよう、消し忘れ等ご注意ください。

環境構築

AWS CDK を利用する場合、前提として Node.js の Ver.10.13.0 以降が必要です。各自ダウンロード[2]を行ってください。また、AWS CLI も必要となります。インストール後、以下のコマンドからクレデンシャルとリージョンの設定を行ってください。

```
> aws configure
```

[1] https://aws.amazon.com/jp/cdk/
[2] https://nodejs.org/ja/download/

尚、EC2 インスタンス上で実行を行う場合は EC2 に適切なロールをアタッチすることでクレデンシャルの入力を省くことができます。アクセスキーID・シークレットアクセスキーの発行ができない場合は EC2 上で実行してみてください。
余談ですが、アクセスキーID・シークレットアクセスキーを扱う場合は絶対に**ハードコーディングしない**で下さい。ハードコーディングしたソースを GitHub などに上げてしまった場合、AWS アカウント流出の危険があります。
AWS CDK と TypeScript のインストールを行います。Node.js の実行環境がある中で以下のコマンドを実行してください。

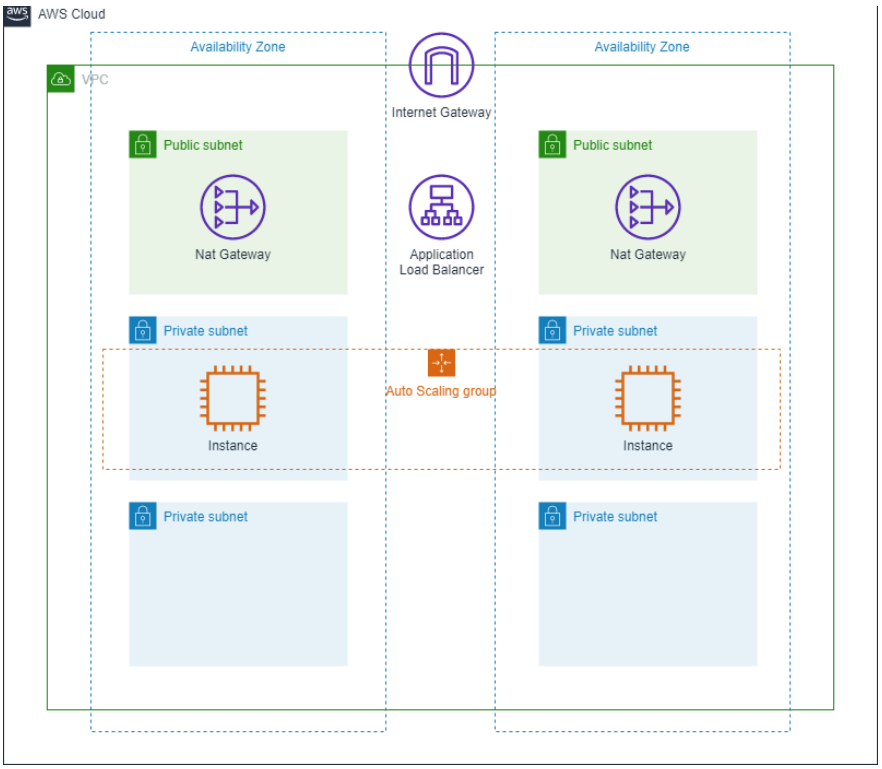
```
> npm install -g aws-cdk
> npm install -g typescript
```

最後にプロジェクトフォルダを作成し、init します。

```
> mkdir test-cdk
> cd test-cdk
> cdk init app --language typescript
```

作成する

今回は以下のような、よくある構成を作成します。



まずは、VPC、Subnetなどのネットワーク周りから作成します。次のコマンドで、必要なモジュールをプロジェクトにインポートします。

```
> npm install @aws-cdk/aws-ec2 @aws-cdk/aws-autoscaling @aws-cdk/aws-elasticloadbalancingv2
```

以下のように「lib/cdk-stack.ts」に変更を加えます。

```
import * as cdk from '@aws-cdk/core';
import * as ec2 from "@aws-cdk/aws-ec2";
import * as autoscaling from '@aws-cdk/aws-autoscaling';
import * as elb from '@aws-cdk/aws-elasticloadbalancingv2';


export class CdkStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here
    const vpc = new ec2.Vpc(this,"test-vpc",{
      cidr:"10.0.0.0/16",
      natGateways:2,
      maxAzs:2,
      subnetConfiguration:[
        {
          cidrMask:24,
          name:'Pubic',
          subnetType: ec2.SubnetType.PUBLIC
        },{
          cidrMask:24,
          name:'Private',
          subnetType: ec2.SubnetType.PRIVATE
        },{
          cidrMask:24,
          name:'Isolated',
          subnetType: ec2.SubnetType.ISOLATED
        }
      ]
    })
  }
}
```

12 行目からネットワークの作成を行っています。VPC の Cidr を決定し、NatGateway の数、AZ の数を決めます。

19,23,27 行では Subnet のタイプを定義しており、PUBLIC は「InternetGateway と関連付けされた Subnet」、PRIVATE は「NatGateway と関連付けされた Subnet」、ISOLATED は「関連付けがない Subnet」となります。
この定義の場合、必要な InternetGateway やルートテーブルなどのリソースは自動的に作成されます。
したがって、これだけで構成におけるネットワーク部分が定義できたことになります。
試しに template を確認後、一度デプロイしてみましょう。※数分かかります。
template 確認

```
> cdk synth
```

デプロイ

```
> cdk deploy
```

マネジメントコンソールなどで確認すると、MultiAZ になっていること・InternetGateway,NatGateway が作成されていること・ルートテーブルが作成されており、ルーティングが適切であることが確認できると思います。
次に、ApplicationLoadBalancer と EC2 の SecurityGroup・起動時に実行するスクリプトを定義します。

先ほどの VPC 定義の下部にプログラムを追記してください。

```
    const albSecurityGroup = new ec2.SecurityGroup(this,"alb-sg",{
      vpc: vpc
    });
    albSecurityGroup.addIngressRule(ec2.Peer.anyIpv4(),ec2.Port.tcp(80))

    const ec2SecurityGroup=new ec2.SecurityGroup(this,"ec2-sg",{
      vpc: vpc
    });
    ec2SecurityGroup.addIngressRule(albSecurityGroup,ec2.Port.tcp(80))

    const userData = ec2.UserData.forLinux({ shebang: '#!/bin/bash -xe' })
    userData.addCommands(
      'sudo yum update -y',
      'sudo yum -y install httpd',
      'echo "<html><head><title>CDKTest</title></head><body>Hello,AWS
CDK</body></html>" > /var/www/html/index.html',
        'sudo service httpd start')
```

1~4 行目では ALB にアタッチする SecurityGroup を定義しています。0.0.0.0/0 から 80 番ポートへのアクセスを受け入れる設定になっています。
6~9 行目では EC2 にアタッチする SecurityGroup を定義しています。ALB からの 80 番ポートへのアクセスのみ受け入れる設定になっています。
11 行目以降の UserData では、EC2 インスタンス起動時に実行するコマンドを定義しています。今回は WebServer を想定して Apache を起動しています。

次に、AutoScalingGroup と ALB の定義を行います。

```
const autoScalingGroup = new autoscaling.AutoScalingGroup(this, 'ec2-
asg', {
  vpc,
  instanceType: ec2.InstanceType.of(ec2.InstanceClass.T2, ec2.InstanceSize.
MICRO),
  machineImage: new ec2.AmazonLinuxImage(),
  desiredCapacity: 2,
  userData: userData,
  maxCapacity: 3,
  securityGroup: ec2SecurityGroup,
  associatePublicIpAddress: false,
  vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE }
});

const applicationLoadBalancer = new elb.ApplicationLoadBalancer(this, 'alb'
, {
  vpc,
  internetFacing: true,
  vpcSubnets:  {subnetType: ec2.SubnetType.PUBLIC},
  securityGroup: albSecurityGroup,
  });
```

1~11 行目では AutoScaling の定義を行っています。AutoScaling で起動するインスタンスタイプ・起動する最小数/最大数・パブリック IP をアタッチするかなどの設定を記載しています。
13 行目以降は ALB の定義を行っています。ALB は外部向けであること・PublicSubnet に配置すること等を記載しています。

最後に ALB の Target の定義を行います。

```
const targetGroup = new elb.ApplicationTargetGroup(this, 'tg', {
    protocol: elb.ApplicationProtocol.HTTP,
    targetGroupName: 'alb-tg',
    vpc,
  });

const listener = applicationLoadBalancer.addListener('Listener', {
    port: 80,
    open: true,
  });

listener.addTargets('ApplicationFleet', {
    port: 80,
    targets: [autoScalingGroup]
  });
```

7~10 行目では、リスナーの設定を行っています。今回は 80 番 Port を空けておきます。
12~14 行目では、ターゲットの設定を行っています。ALB に対し、80 番へのアクセスを AutoScaling(EC2)に 80 番で流すという設定になっています。

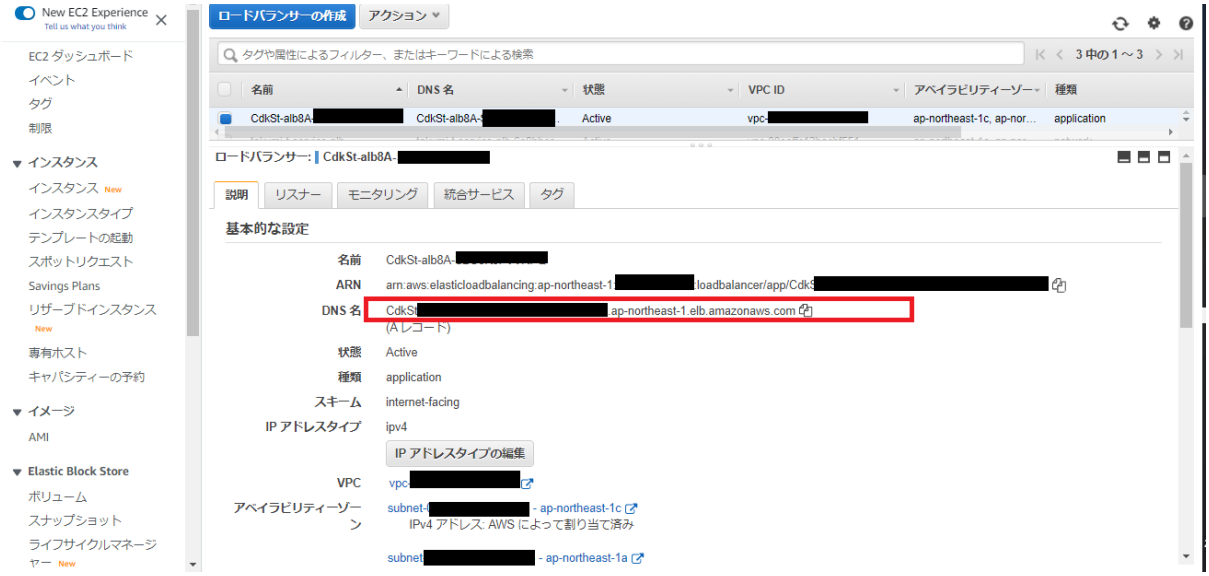
設定が完了したので、diff を確認後、デプロイをしてみましょう。

```
> cdk diff
```

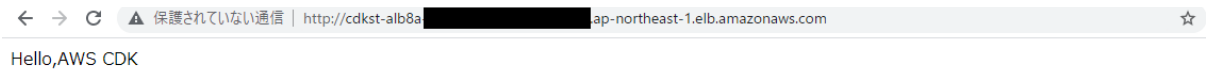
diff コマンドを打つと、新たに作成、削除されるリソースを確認することができます。デプロイ前に確認しましょう。

```
> cdk deploy
```

マネコンで確認すると、これまで作成したリソースが表示されます。
試しに、ALB のエンドポイントにアクセスしてみます。マネジメントコンソールから ALB のコンソールへアクセスし、エンドポイントを確認します。



テストページが表示され、EC2 へのルーティングが適切に行われていることがわかります。



Hello,AWS CDK

最後に以下のコマンドでリソースを削除します。

```
> cdk destroy
```

おわりに

AWS CDK を利用して簡単な構成を構築しました。プログラミングライクにインフラ構築する面白さが少しでも伝わればと思います。本稿では紹介しきれませんでしたが、AWS CDK はクロススタック参照が行えたり、IDE のメリットの補完機能を使えたり楽になる部分が多々あります。ただ、少ない記述量でかける分抽象度が高くデフォルト値を意識する必要がある側面もあるので、ドキュメント[3]を確認しつつ構築を行ってください。

[3] https://docs.aws.amazon.com/cdk/api/latest/docs/aws-construct-library.html

GSLetterNeo Vol.157
2021 年 8 月 20 日発行
発行者　株式会社 SRA 先端技術研究所

編集者　熊澤努　方 学芬
バックナンバー http://www.sra.co.jp/gsletter
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8

夢を。Yawaraka Innovation
やわらかいのべーしょん