Panduan Developer

AWS Cloud Development Kit (AWS CDK) v2

Versi 2

AWS Cloud Development Kit (AWS CDK) v2: Panduan Developer

Table of Contents

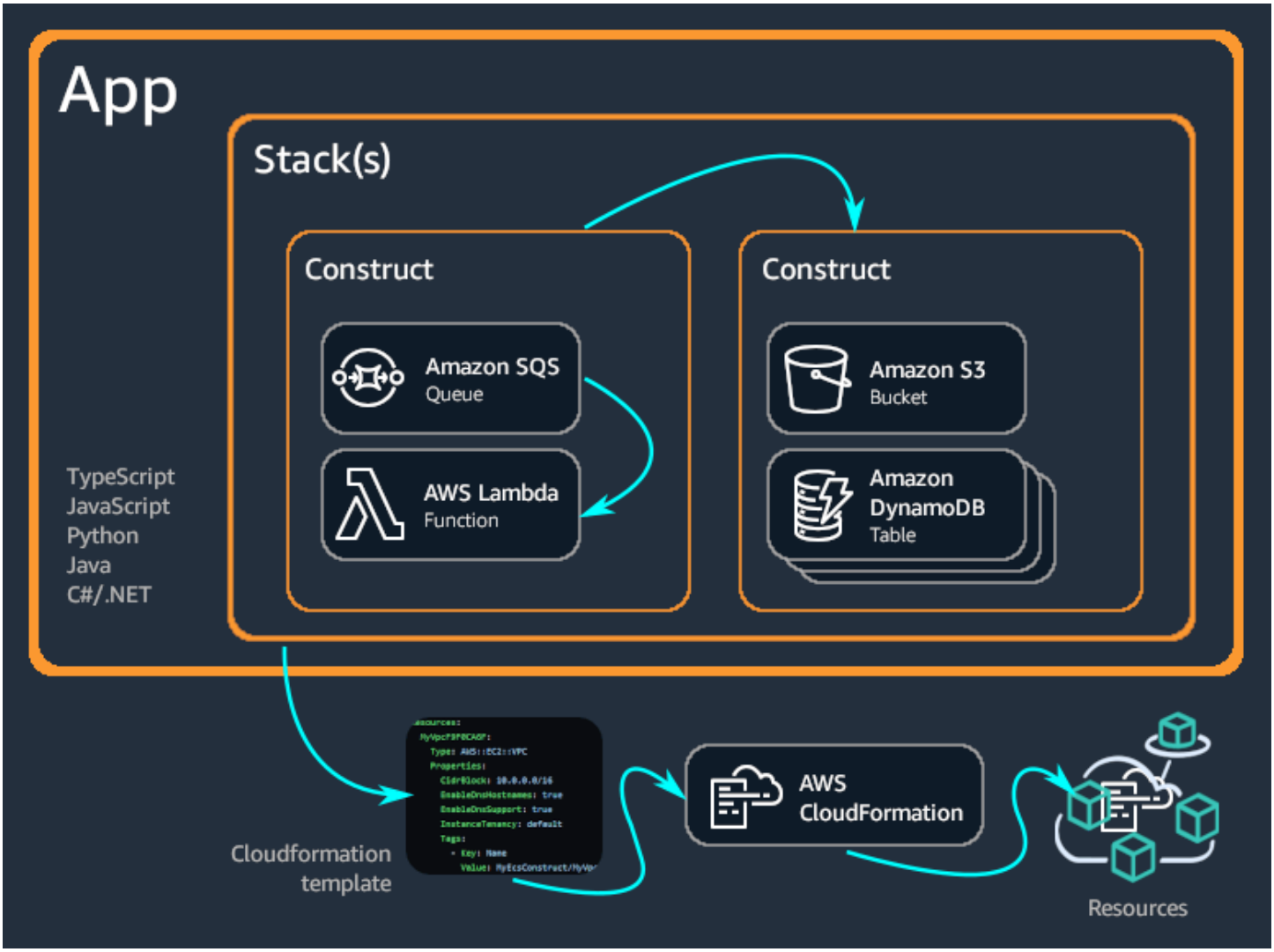
Apa itu AWS CDK?

AWS Cloud Development Kit (AWS CDK) Ini adalah kerangka pengembangan perangkat lunak open-source untuk mendefinisikan infrastruktur cloud dalam kode dan menyediakannya. AWS CloudFormation

AWS CDK Terdiri dari dua bagian utama:

- AWS CDK Construct Library - Kumpulan kode modular dan dapat digunakan kembali yang telah ditulis sebelumnya, yang disebut konstruksi, yang dapat Anda gunakan, modifikasi, dan integrasikan untuk mengembangkan infrastruktur Anda dengan cepat. Tujuan dari AWS CDK Construct Library adalah untuk mengurangi kompleksitas yang diperlukan untuk mendefinisikan dan mengintegrasikan AWS layanan bersama-sama saat membangun aplikasi. AWS
- AWS CDK Toolkit — Alat baris perintah untuk berinteraksi dengan aplikasi CDK. Gunakan AWS CDK Toolkit untuk membuat, mengelola, dan menyebarkan proyek Anda AWS CDK .

AWS CDK DukunganTypeScript,, JavaScriptPython,Java,C#/.Net, danGo. Anda dapat menggunakan salah satu bahasa pemrograman yang didukung ini untuk menentukan komponen cloud yang dapat digunakan kembali yang dikenal sebagai konstruksi. Anda menyusun ini bersama-sama ke tumpukan dan aplikasi. Kemudian, Anda menyebarkan aplikasi CDK Anda AWS CloudFormation untuk menyediakan atau memperbarui sumber daya Anda.



Topik

- Manfaat dari AWS CDK
- Contoh dari AWS CDK
- AWS CDK fitur
- Langkah selanjutnya
- Pelajari selengkapnya

Manfaat dari AWS CDK

Gunakan AWS CDK untuk mengembangkan aplikasi yang andal, terukur, dan hemat biaya di cloud dengan kekuatan ekspresif yang cukup besar dari bahasa pemrograman. Pendekatan ini menghasilkan banyak manfaat, termasuk:

Kembangkan dan kelola infrastruktur Anda sebagai kode (IAc)

Praktikkan infrastruktur sebagai kode untuk membuat, menyebarkan, dan memelihara infrastruktur dengan cara yang terprogram, deskriptif, dan deklaratif. Dengan IAc, Anda memperlakukan infrastruktur dengan cara yang sama seperti pengembang memperlakukan kode. Ini menghasilkan pendekatan yang terukur dan terstruktur untuk mengelola infrastruktur. Untuk mempelajari lebih lanjut tentang IAc, lihat Infrastruktur sebagai kode di Pengantar DevOps pada AWS Whitepaper.

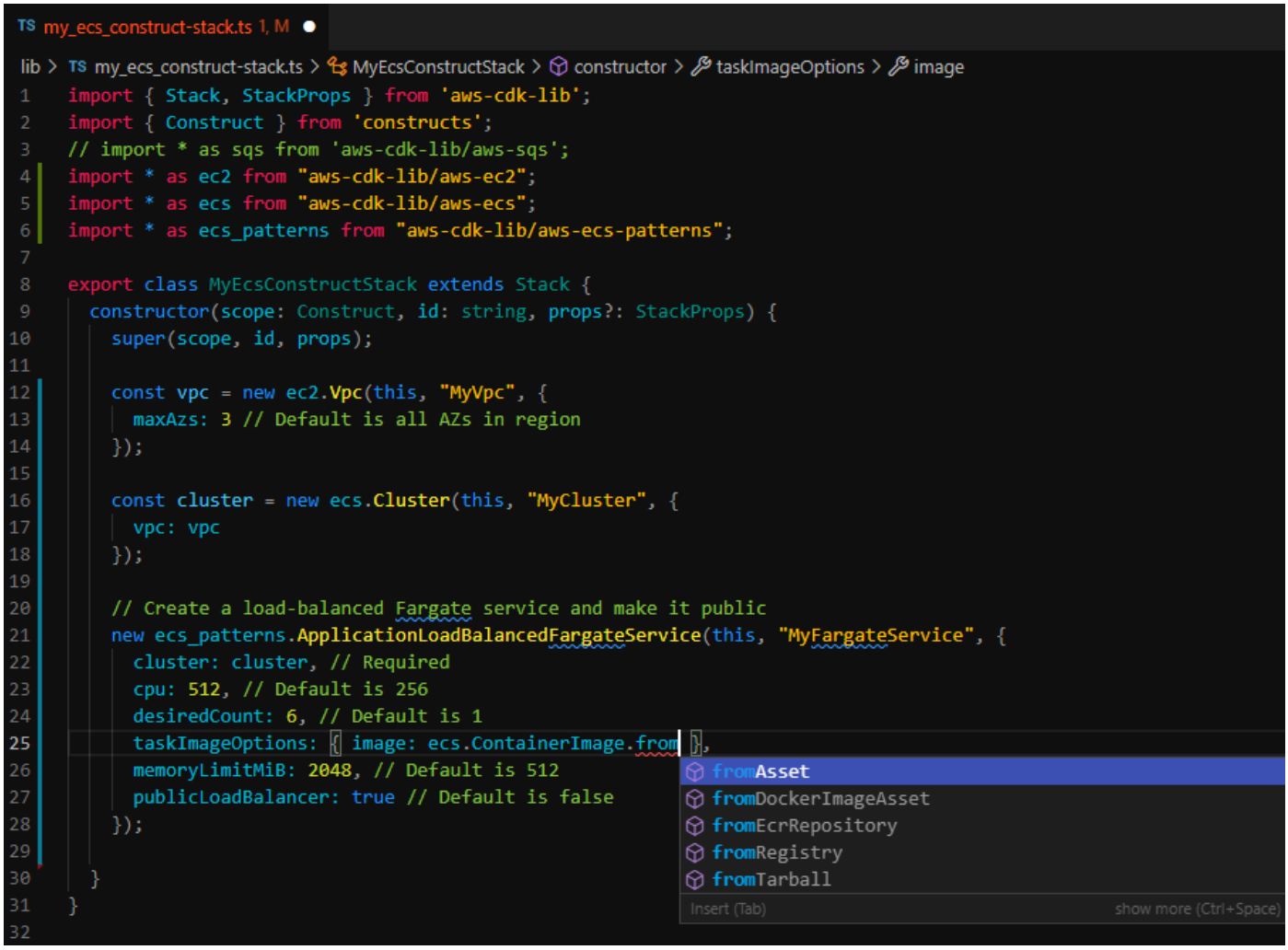
Dengan ini AWS CDK, Anda dapat menempatkan infrastruktur, kode aplikasi, dan konfigurasi Anda di satu tempat, memastikan bahwa Anda memiliki sistem yang lengkap dan dapat diterapkan di cloud di setiap tonggak sejarah. Gunakan praktik terbaik rekayasa perangkat lunak seperti tinjauan kode, pengujian unit, dan kontrol sumber untuk membuat infrastruktur Anda lebih kuat.

Tentukan infrastruktur cloud Anda menggunakan bahasa pemrograman tujuan umum

Dengan itu AWS CDK, Anda dapat menggunakan salah satu bahasa pemrograman berikut untuk menentukan infrastruktur cloud Anda:TypeScript,JavaScript,Python,Java,C#/.Net, danGo. Pilih bahasa pilihan Anda dan gunakan elemen pemrograman seperti parameter, kondisional, loop, komposisi, dan pewarisan untuk menentukan hasil yang diinginkan dari infrastruktur Anda.

Gunakan bahasa pemrograman yang sama untuk menentukan infrastruktur dan logika aplikasi Anda.

Dapatkan manfaat dari pengembangan infrastruktur di IDE pilihan Anda (Integrated Development Environment), seperti penyorotan sintaks dan penyelesaian kode cerdas.



Menyebarkan infrastruktur melalui AWS CloudFormation

AWS CDK terintegrasi dengan AWS CloudFormation untuk menyebarkan dan menyediakan infrastruktur Anda. AWS AWS CloudFormation adalah dikelola Layanan AWS yang menawarkan dukungan ekstensif konfigurasi sumber daya dan properti untuk penyediaan layanan di. AWS Dengan AWS CloudFormation, Anda dapat melakukan penerapan infrastruktur secara dapat diprediksi dan berulang kali, dengan kesalahan rollback. Jika Anda sudah terbiasa dengan AWS CloudFormation, Anda tidak perlu mempelajari layanan manajemen IAC baru ketika memulai dengan. AWS CDK

Mulai mengembangkan aplikasi Anda dengan cepat dengan konstruksi

Kembangkan lebih cepat dengan menggunakan dan berbagi komponen yang dapat digunakan kembali yang disebut konstruksi. Gunakan konstruksi tingkat rendah untuk menentukan AWS CloudFormation sumber daya individu dan propertinya. Gunakan konstruksi tingkat tinggi untuk mendefinisikan komponen aplikasi yang lebih besar dengan cepat, dengan default yang masuk

akal dan aman untuk AWS sumber daya Anda, mendefinisikan lebih banyak infrastruktur dengan kode yang lebih sedikit.

Buat konstruksi Anda sendiri yang disesuaikan untuk kasus penggunaan unik Anda dan bagikan di seluruh organisasi Anda atau bahkan dengan publik.

Contoh dari AWS CDK

Berikut ini adalah contoh menggunakan AWS CDK Constructs Library untuk membuat layanan Amazon Elastic Container Service (Amazon ECS) Container Service dengan tipe peluncuran. AWS Fargate (Fargate) Untuk detail lebih lanjut tentang contoh ini, lihatthe section called “ECS”.

TypeScript

```
export class MyEcsConstructStack extends Stack {
  constructor(scope: App, id: string, props?: StackProps) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService",
 {
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-
sample") },
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is false
    });
  }
}
```

JavaScript

```
class MyEcsConstructStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService",
 {
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-
sample") },
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is false
    });
  }
}

module.exports = { MyEcsConstructStack }
```

Python

```
class MyEcsConstructStack(Stack):

    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        vpc = ec2.Vpc(self, "MyVpc", max_azs=3)     # default is all AZs in region

        cluster = ecs.Cluster(self, "MyCluster", vpc=vpc)

        ecs_patterns.ApplicationLoadBalancedFargateService(self, "MyFargateService",
            cluster=cluster,            # Required
```

```
            cpu=512,                    # Default is 256
            desired_count=6,            # Default is 1
            task_image_options=ecs_patterns.ApplicationLoadBalancedTaskImageOptions(
                image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample")),
            memory_limit_mib=2048,      # Default is 512
            public_load_balancer=True)  # Default is False
```

Java

```
public class MyEcsConstructStack extends Stack {

    public MyEcsConstructStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyEcsConstructStack(final Construct scope, final String id,
            StackProps props) {
        super(scope, id, props);

        Vpc vpc = Vpc.Builder.create(this, "MyVpc").maxAzs(3).build();

        Cluster cluster = Cluster.Builder.create(this, "MyCluster")
                .vpc(vpc).build();

        ApplicationLoadBalancedFargateService.Builder.create(this,
 "MyFargateService")
                .cluster(cluster)
                .cpu(512)
                .desiredCount(6)
                .taskImageOptions(
                       ApplicationLoadBalancedTaskImageOptions.builder()
                               .image(ContainerImage
                                       .fromRegistry("amazon/amazon-ecs-sample"))
                               .build()).memoryLimitMiB(2048)
                .publicLoadBalancer(true).build();
    }
}
```

C#

```
public class MyEcsConstructStack : Stack
{
```

```
    public MyEcsConstructStack(Construct scope, string id, IStackProps props=null) :
 base(scope, id, props)
    {
        var vpc = new Vpc(this, "MyVpc", new VpcProps
        {
            MaxAzs = 3
        });

        var cluster = new Cluster(this, "MyCluster", new ClusterProps
        {
            Vpc = vpc
        });

        new ApplicationLoadBalancedFargateService(this, "MyFargateService",
            new ApplicationLoadBalancedFargateServiceProps
        {
            Cluster = cluster,
            Cpu = 512,
            DesiredCount = 6,
            TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
            {
                Image = ContainerImage.FromRegistry("amazon/amazon-ecs-sample")
            },
            MemoryLimitMiB = 2048,
            PublicLoadBalancer = true,
        });
    }
}
```

Go

```
func NewMyEcsConstructStack(scope constructs.Construct, id string, props
 *MyEcsConstructStackProps) awscdk.Stack {

 var sprops awscdk.StackProps

 if props != nil {
  sprops = props.StackProps
 }

 stack := awscdk.NewStack(scope, &id, &sprops)

 vpc := awsec2.NewVpc(stack, jsii.String("MyVpc"), &awsec2.VpcProps{
```

```
  MaxAzs: jsii.Number(3), // Default is all AZs in region
 })

 cluster := awsecs.NewCluster(stack, jsii.String("MyCluster"), &awsecs.ClusterProps{
  Vpc: vpc,
 })

 awsecspatterns.NewApplicationLoadBalancedFargateService(stack,
 jsii.String("MyFargateService"),
  &awsecspatterns.ApplicationLoadBalancedFargateServiceProps{
   Cluster:        cluster,           // required
   Cpu:            jsii.Number(512),  // default is 256
   DesiredCount:   jsii.Number(5),    // default is 1
   MemoryLimitMiB: jsii.Number(2048), // Default is 512
   TaskImageOptions: &awsecspatterns.ApplicationLoadBalancedTaskImageOptions{
    Image: awsecs.ContainerImage_FromRegistry(jsii.String("amazon/amazon-ecs-
sample"), nil),
   },
   PublicLoadBalancer: jsii.Bool(true), // Default is false
  })

 return stack

}
```

Kelas ini menghasilkan AWS CloudFormation template lebih dari 500 baris. Menerapkan AWS CDK aplikasi menghasilkan lebih dari 50 sumber daya dari jenis berikut.

- AWS: :EC2: :EIP
- AWS::EC2::InternetGateway
- AWS::EC2::NatGateway
- AWS::EC2::Route
- AWS::EC2::RouteTable
- AWS::EC2::SecurityGroup
- AWS::EC2::Subnet
- AWS::EC2::SubnetRouteTableAssociation
- AWS: :EC2: :VPC GatewayAttachment
- AWS: :EC2: :VPC

- AWS::ECS::Cluster
- AWS::ECS::Service
- AWS::ECS::TaskDefinition
- AWS::ElasticLoadBalancingV2::Listener
- AWS::ElasticLoadBalancingV2::LoadBalancer
- AWS::ElasticLoadBalancingV2::TargetGroup
- AWS::IAM::Policy
- AWS::IAM::Role
- AWS::Logs::LogGroup

AWS CDK fitur

AWS CDKGitHubRepositori

Untuk AWS CDKGitHub repositori resmi, lihat aws-cdk. Di sini, Anda dapat mengirimkan masalah, melihat lisensi kami, melacak rilis, dan banyak lagi.

Karena open-source, tim mendorong Anda untuk berkontribusi menjadikannya alat yang lebih baik. AWS CDK Untuk detailnya, lihat Berkontribusi pada AWS Cloud Development Kit (AWS CDK).

Referensi AWS CDK API

AWS CDK Construct Library menyediakan API untuk menentukan aplikasi CDK Anda dan menambahkan konstruksi CDK ke aplikasi. Untuk informasi lebih lanjut, lihat Referensi API AWS CDK.

Model Pemrograman Konstruksi

Construct Programming Model (CPM) memperluas konsep di balik domain tambahan AWS CDK . Alat lain yang menggunakan CPM meliputi:

- CDK untuk Terraform (CdKTF)
- CDK para Kubernetes (CDK8s)
- Proyek, untuk membangun konfigurasi proyek

Hub Konstruksi

Construct Hub adalah registri online tempat Anda dapat menemukan, menerbitkan, dan berbagi pustaka sumber terbuka AWS CDK .

Langkah selanjutnya

Untuk memulai dengan menggunakan AWS CDK, lihatMemulai dengan AWS CDK.

Pelajari selengkapnya

Untuk terus belajar tentang hal ini AWS CDK, lihat yang berikut ini:

- AWS CDK konsep— Konsep dan istilah penting untuk AWS CDK.
- AWS CDK Workshop — Lokakarya langsung untuk mempelajari dan menggunakan. AWS CDK
- AWS CDK Pola — Koleksi sumber terbuka pola arsitektur AWS tanpa server, dibangun untuk para ahli. AWS CDK AWS
- AWS CDK contoh kode — GitHub repositori proyek contoh AWS CDK .
- cdk.dev — Hub berbasis komunitas untuk, termasuk ruang kerja komunitas. AWS CDKSlack
- Awesome CDK — GitHub repositori yang berisi daftar proyek AWS CDK open-source, panduan, blog, dan sumber daya lainnya yang dikurasi.
- AWS Konstruksi Solusi — Pola infrastruktur konfigurasi sebagai kode (IAc) yang diperiksa, yang dapat dengan mudah dirakit menjadi aplikasi siap produksi.
- AWS Alat Pengembang Blog - Posting blog disaring untuk AWS CDK.
- AWS CDK on Stack Overflow — Pertanyaan yang ditandai dengan aws-cdk on. Stack Overflow
- AWS CDK tutorial untuk AWS Cloud9 — Tutorial tentang menggunakan AWS CDK dengan lingkungan AWS Cloud9 pengembangan.

Untuk mempelajari lebih lanjut tentang topik terkait dengan AWS CDK, lihat berikut ini:

- AWS CloudFormation Konsep — Karena AWS CDK dibangun untuk bekerja dengan AWS CloudFormation, kami sarankan Anda belajar dan memahami AWS CloudFormation konsep-konsep kunci.
- AWS Glosarium — Definisi istilah kunci yang digunakan di seluruh. AWS

Untuk mempelajari lebih lanjut tentang alat yang terkait dengan AWS CDK yang dapat digunakan untuk menyederhanakan pengembangan dan penyebaran aplikasi tanpa server, lihat berikut ini:

- AWS Serverless Application Model— Alat pengembang open-source yang menyederhanakan dan meningkatkan pengalaman membangun dan menjalankan aplikasi tanpa server. AWS
- AWSChalice— Kerangka kerja untuk menulis aplikasi tanpa server di. Python

AWS CDK konsep

Pelajari konsep inti di balik AWS Cloud Development Kit (AWS CDK).

AWS CDK dan IaC

AWS CDK Ini adalah kerangka kerja sumber terbuka yang dapat Anda gunakan untuk mengelola AWS infrastruktur Anda menggunakan kode. Pendekatan ini dikenal sebagai infrastruktur sebagai code (IaC). Dengan mengelola dan menyediakan infrastruktur Anda sebagai kode, Anda memperlakukan infrastruktur Anda dengan cara yang sama seperti pengembang memperlakukan kode. Ini memberikan banyak manfaat, seperti kontrol versi dan skalabilitas. Untuk mempelajari lebih lanjut tentang IAc, lihat Apa itu Infrastruktur sebagai Kode?

AWS CDK dan AWS CloudFormation

AWS CDK Ini terintegrasi erat dengan AWS CloudFormation. AWS CloudFormation adalah layanan yang dikelola sepenuhnya yang dapat Anda gunakan untuk mengelola dan menyediakan infrastruktur Anda AWS. Dengan AWS CloudFormation, Anda menentukan infrastruktur Anda dalam template dan AWS CloudFormation menerapkannya. AWS CloudFormation Layanan kemudian menyediakan infrastruktur Anda sesuai dengan konfigurasi yang ditentukan pada template Anda.

AWS CloudFormation template bersifat deklaratif, artinya mereka mendeklarasikan status atau hasil yang diinginkan dari infrastruktur Anda. Menggunakan JSON atau YAMB, Anda mendeklarasikan AWS infrastruktur Anda dengan mendefinisikan AWS sumber daya dan properti. Sumber daya mewakili banyak layanan AWS dan properti mewakili konfigurasi yang Anda inginkan dari layanan tersebut. Saat Anda menerapkan templat AWS CloudFormation, sumber daya dan properti yang dikonfigurasikan akan disediakan seperti yang dijelaskan pada templat Anda.

Dengan AWS CDK, Anda dapat mengelola infrastruktur Anda secara imperatif, menggunakan bahasa pemrograman tujuan umum. Alih-alih hanya mendefinisikan keadaan yang diinginkan secara deklaratif, Anda dapat menentukan logika atau urutan yang diperlukan untuk mencapai keadaan yang diinginkan. Misalnya, Anda dapat menggunakan `if` pernyataan atau loop bersyarat yang menentukan cara mencapai status akhir yang diinginkan untuk infrastruktur Anda.

Infrastruktur yang AWS CDK dibuat dengan akhirnya diterjemahkan, atau disintesis ke dalam AWS CloudFormation template dan digunakan menggunakan layanan. AWS CloudFormation Jadi

sementara AWS CDK menawarkan pendekatan yang berbeda untuk menciptakan infrastruktur Anda, Anda masih menerima manfaat dari AWS CloudFormation, seperti dukungan konfigurasi AWS sumber daya yang luas dan proses penyebaran yang kuat.

Untuk mempelajari lebih lanjut tentang AWS CloudFormation, lihat Apa itu AWS CloudFormation? dalam AWS CloudFormation User Guide.

AWS CDK dan abstraksi

Dengan AWS CloudFormation, Anda harus menentukan setiap detail tentang bagaimana sumber daya Anda dikonfigurasi. Ini memberikan manfaat memiliki kontrol penuh atas infrastruktur Anda. Namun, ini mengharuskan Anda untuk mempelajari, memahami, dan membuat templat tangguh yang berisi detail konfigurasi sumber daya dan hubungan antar sumber daya, seperti izin dan interaksi berbasis peristiwa.

Dengan AWS CDK, Anda dapat memiliki kontrol yang sama atas konfigurasi sumber daya Anda. Namun, AWS CDK juga menawarkan abstraksi yang kuat, yang dapat mempercepat dan menyederhanakan proses pengembangan infrastruktur. Misalnya, konstruksi AWS CDK include yang menyediakan konfigurasi default yang masuk akal dan metode pembantu yang menghasilkan kode boilerplate untuk Anda. Ini AWS CDK juga menawarkan alat, seperti AWS CDK Command Line Interface (AWS CDK CLI), yang melakukan tindakan manajemen infrastruktur untuk Anda.

Pelajari lebih lanjut tentang AWS CDK konsep inti

Berinteraksi dengan AWS CDK

Saat menggunakan dengan AWS CDK, Anda terutama akan berinteraksi dengan AWS Construct Library dan. AWS CDK CLI

Berkembang dengan AWS CDK

AWS CDK Dapat ditulis dalam bahasa pemrograman yang didukung. Anda mulai dengan proyek CDK, yang berisi struktur folder dan file, termasuk aset. Dalam proyek, Anda membuat aplikasi CDK. Di dalam aplikasi, Anda menentukan tumpukan, yang secara langsung mewakili CloudFormation tumpukan. Dalam tumpukan, Anda menentukan AWS sumber daya dan properti Anda menggunakan konstruksi.

Menyebarkan dengan AWS CDK

Anda menyebarkan aplikasi CDK ke lingkungan AWS . Sebelum menerapkan, Anda harus melakukan bootstrap satu kali untuk mempersiapkan lingkungan Anda.

Pelajari selengkapnya

Untuk mempelajari lebih lanjut tentang konsep AWS CDK inti, lihat topik di bagian ini.

Bahasa pemrograman yang didukung

Ini AWS Cloud Development Kit (AWS CDK) memiliki dukungan kelas satu untuk bahasa pemrograman tujuan umum berikut:

- TypeScript
- JavaScript
- Python
- Java
- C#
- Go

Bahasa lain JVM dan .NET CLR bahasa juga dapat digunakan secara teori, tetapi kami tidak menawarkan dukungan resmi saat ini.

> **Note**
>
> Panduan ini saat ini tidak menyertakan instruksi atau contoh kode Go selain darithe section called “Di Go”.

AWS CDK Ini dikembangkan dalam satu bahasa,TypeScript. Untuk mendukung bahasa lain, AWS CDK menggunakan alat yang disebut JSIIuntuk menghasilkan binding bahasa.

Kami berusaha menawarkan konvensi biasa setiap bahasa untuk membuat pengembangan dengan AWS CDK sealami dan seintuitif mungkin. Misalnya, kami mendistribusikan modul AWS Construct Library menggunakan repositori standar bahasa pilihan Anda, dan Anda menginstalnya

menggunakan pengelola paket standar bahasa tersebut. Metode dan properti juga diberi nama menggunakan pola penamaan yang direkomendasikan bahasa Anda.

Berikut ini adalah beberapa contoh kode:

TypeScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
  versioned: true,
  websiteRedirect: {hostName: 'aws.amazon.com'}});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
  versioned: true,
  websiteRedirect: {hostName: 'aws.amazon.com'}});
```

Python

```
bucket = s3.Bucket("MyBucket", bucket_name="my-bucket", versioned=True,
            website_redirect=s3.RedirectTarget(host_name="aws.amazon.com"))
```

Java

```
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .bucketName("my-bucket")
                      .versioned(true)
                      .websiteRedirect(new RedirectTarget.Builder()
                          .hostName("aws.amazon.com").build())
                      .build();
```

C#

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
                      BucketName = "my-bucket",
                      Versioned  = true,
                      WebsiteRedirect = new RedirectTarget {
                              HostName = "aws.amazon.com"
```

```
                        }});
```

Go

```
bucket := awss3.NewBucket(scope, jsii.String("MyBucket"), &awss3.BucketProps {
 BucketName: jsii.String("my-bucket"),
 Versioned: jsii.Bool(true),
 WebsiteRedirect: &awss3.RedirectTarget {
  HostName: jsii.String("aws.amazon.com"),
 },
})
```

> **Note**
>
> Cuplikan kode ini ditujukan hanya untuk ilustrasi. Mereka tidak lengkap dan tidak akan berjalan sebagaimana adanya.

Perpustakaan AWS Konstruksi didistribusikan menggunakan alat manajemen paket standar masing-masing bahasa, termasukNPM,, PyPiMaven, danNuGet. Kami juga menyediakan versi Referensi AWS CDK API untuk setiap bahasa.

Untuk membantu Anda menggunakan bahasa pilihan Anda, panduan ini mencakup topik berikut untuk bahasa yang didukung: AWS CDK

- the section called “Di TypeScript”
- the section called “Di JavaScript”
- the section called “Dengan Python”
- the section called “Di Jawa”
- the section called “Dalam C #”
- the section called “Di Go”

TypeScriptadalah bahasa pertama yang didukung oleh AWS CDK, dan sebagian besar kode AWS CDK contoh ditulis dalamTypeScript. Panduan ini mencakup topik khusus untuk menunjukkan cara mengadaptasi TypeScript AWS CDK kode untuk digunakan dengan bahasa lain yang didukung. Untuk informasi selengkapnya, lihat AWS CDK Membandingkan TypeScript dengan bahasa lain.

AWS CDK proyek

AWS Cloud Development Kit (AWS CDK) Proyek mewakili file dan folder yang berisi kode CDK Anda. Konten akan bervariasi berdasarkan bahasa pemrograman Anda.

Anda dapat membuat AWS CDK proyek Anda secara manual atau dengan AWS CDK perintah Command Line Interface (AWS CDK CLI)`cdk init`. Dalam topik ini, kita akan merujuk pada struktur proyek dan konvensi penamaan file dan folder yang dibuat oleh AWS CDK CLI. Anda dapat menyesuaikan dan mengatur proyek CDK Anda agar sesuai dengan kebutuhan Anda.

> **Note**
>
> Struktur proyek yang dibuat oleh AWS CDK CLI dapat bervariasi antar versi dari waktu ke waktu.

Topik

File dan folder universal

.git

Jika Anda telah `git` menginstal, AWS CDK CLI secara otomatis menginisialisasi Git repositori untuk proyek Anda. `.git`Direktori berisi informasi tentang repositori.

.gitignore

File teks yang digunakan oleh Git untuk menentukan file dan folder untuk diabaikan.

README.md

File teks yang memberi Anda panduan dasar dan informasi penting untuk mengelola AWS CDK proyek Anda. Ubah file ini seperlunya untuk mendokumentasikan informasi penting mengenai proyek CDK Anda.

cdk.json

File konfigurasi untuk file AWS CDK. File ini memberikan instruksi AWS CDK CLI tentang cara menjalankan aplikasi Anda.

File dan folder khusus bahasa

File dan folder berikut unik untuk setiap bahasa pemrograman yang didukung.

TypeScript

Berikut ini adalah contoh proyek yang dibuat dalam `my-cdk-ts-project` direktori menggunakan `cdk init --language typescript` perintah:

```
my-cdk-ts-project
### .git
### .gitignore
### .npmignore
### README.md
### bin
#   ### my-cdk-ts-project.ts
### cdk.json
### jest.config.js
### lib
#   ### my-cdk-ts-project-stack.ts
### node_modules
### package-lock.json
### package.json
### test
#   ### my-cdk-ts-project.test.ts
### tsconfig.json
```

.npmignore

File yang menentukan file dan folder mana yang harus diabaikan saat menerbitkan paket ke npm registri. File ini mirip dengan `.gitignore`, tetapi khusus untuk npm paket.

tempat sampah/ .ts my-cdk-ts-project

File aplikasi mendefinisikan aplikasi CDK Anda. Proyek CDK dapat berisi satu atau lebih file aplikasi. File aplikasi disimpan di `bin` folder.

Berikut ini adalah contoh file aplikasi dasar yang mendefinisikan aplikasi CDK:

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { MyCdkTsProjectStack } from '../lib/my-cdk-ts-project-stack';
```

```
const app = new cdk.App();
new MyCdkTsProjectStack(app, 'MyCdkTsProjectStack');
```

jest.config.js

File konfigurasi untukJest. Jestadalah kerangka JavaScript pengujian yang populer.

lib/ my-cdk-ts-project -stack.ts

File stack mendefinisikan tumpukan CDK Anda. Dalam tumpukan Anda, Anda mendefinisikan AWS sumber daya dan properti menggunakan konstruksi.

Berikut ini adalah contoh file stack dasar yang mendefinisikan tumpukan CDK:

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';

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

  // code that defines your resources and properties go here
 }
}
```

node_modules

Folder umum dalam Node.js proyek yang berisi dependensi untuk proyek Anda.

package-lock.json

File metadata yang bekerja dengan `package.json` file untuk mengelola versi dependensi.

package.json

File metadata yang biasa digunakan dalam Node.js proyek. File ini berisi informasi tentang proyek CDK Anda seperti nama proyek, definisi skrip, dependensi, dan informasi tingkat proyek impor lainnya.

uji/ .test.ts my-cdk-ts-project

Folder pengujian dibuat untuk mengatur pengujian untuk proyek CDK Anda. File uji sampel juga dibuat.

Anda dapat menulis tes TypeScript dan menggunakan Jest untuk mengkompilasi TypeScript kode Anda sebelum menjalankan tes.

tsconfig.json

File konfigurasi yang digunakan dalam TypeScript proyek yang menentukan opsi kompiler dan pengaturan proyek.

JavaScript

Berikut ini adalah contoh proyek yang dibuat dalam `my-cdk-js-project` direktori menggunakan `cdk init --language javascript` perintah:

```
my-cdk-js-project
### .git
### .gitignore
### .npmignore
### README.md
### bin
#   ### my-cdk-js-project.js
### cdk.json
### jest.config.js
### lib
#   ### my-cdk-js-project-stack.js
### node_modules
### package-lock.json
### package.json
### test
    ### my-cdk-js-project.test.js
```

.npmignore

File yang menentukan file dan folder mana yang harus diabaikan saat menerbitkan paket ke npm registri. File ini mirip dengan `.gitignore`, tetapi khusus untuk npm paket.

tempat sampah/.js my-cdk-js-project

File aplikasi mendefinisikan aplikasi CDK Anda. Proyek CDK dapat berisi satu atau lebih file aplikasi. File aplikasi disimpan di `bin` folder.

Berikut ini adalah contoh file aplikasi dasar yang mendefinisikan aplikasi CDK:

```
#!/usr/bin/env node
```

```
const cdk = require('aws-cdk-lib');
const { MyCdkJsProjectStack } = require('../lib/my-cdk-js-project-stack');

const app = new cdk.App();
new MyCdkJsProjectStack(app, 'MyCdkJsProjectStack');
```

jest.config.js

File konfigurasi untukJest. Jestadalah kerangka JavaScript pengujian yang populer.

lib/ -stack.js my-cdk-js-project

File stack mendefinisikan tumpukan CDK Anda. Dalam tumpukan Anda, Anda mendefinisikan AWS sumber daya dan properti menggunakan konstruksi.

Berikut ini adalah contoh file stack dasar yang mendefinisikan tumpukan CDK:

```
const { Stack, Duration } = require('aws-cdk-lib');

class MyCdkJsProjectStack extends Stack {
 constructor(scope, id, props) {
  super(scope, id, props);

  // code that defines your resources and properties go here
 }
}

module.exports = { MyCdkJsProjectStack }
```

node_modules

Folder umum dalam Node.js proyek yang berisi dependensi untuk proyek Anda.

package-lock.json

File metadata yang bekerja dengan `package.json` file untuk mengelola versi dependensi.

package.json

File metadata yang biasa digunakan dalam Node.js proyek. File ini berisi informasi tentang proyek CDK Anda seperti nama proyek, definisi skrip, dependensi, dan informasi tingkat proyek impor lainnya.

uji/ .test.js my-cdk-js-project

Folder pengujian dibuat untuk mengatur pengujian untuk proyek CDK Anda. File uji sampel juga dibuat.

Anda dapat menulis tes JavaScript dan menggunakan Jest untuk mengkompilasi JavaScript kode Anda sebelum menjalankan tes.

Python

Berikut ini adalah contoh proyek yang dibuat dalam `my-cdk-py-project` direktori menggunakan `cdk init --language python` perintah:

```
my-cdk-py-project
### .git
### .gitignore
### .venv
### README.md
### app.py
### cdk.json
### my_cdk_py_project
#   ### __init__.py
#   ### my_cdk_py_project_stack.py
### requirements-dev.txt
### requirements.txt
### source.bat
### tests
    ### __init__.py
    ### unit
```

.venv

CDK CLI secara otomatis menciptakan lingkungan virtual untuk proyek Anda. `.venv`Direktori mengacu pada lingkungan virtual ini.

app.py

File aplikasi mendefinisikan aplikasi CDK Anda. Proyek CDK dapat berisi satu atau lebih file aplikasi.

Berikut ini adalah contoh file aplikasi dasar yang mendefinisikan aplikasi CDK:

```
#!/usr/bin/env python3
```

```
import os

import aws_cdk as cdk

from my_cdk_py_project.my_cdk_py_project_stack import MyCdkPyProjectStack

app = cdk.App()
MyCdkPyProjectStack(app, "MyCdkPyProjectStack")

app.synth()
```

my_cdk_py_project

Direktori yang berisi file tumpukan Anda. CDK CLI membuat yang berikut di sini:

- `__init__.py` — File definisi Python paket kosong.
- `my_cdk_py_project`— File yang mendefinisikan tumpukan CDK Anda. Anda kemudian mendefinisikan AWS sumber daya dan properti dalam tumpukan menggunakan konstruksi.

Berikut ini adalah contoh file stack:

```
from aws_cdk import Stack

from constructs import Construct

class MyCdkPyProjectStack(Stack):
 def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
  super().__init__(scope, construct_id, **kwargs)

  # code that defines your resources and properties go here
```

requirements-dev.txt

File mirip dengan`requirements.txt`, tetapi digunakan untuk mengelola dependensi khusus untuk tujuan pengembangan daripada produksi.

requirements.txt

File umum yang digunakan dalam Python proyek untuk menentukan dan mengelola dependensi proyek.

source.bat

Batch file untuk Windows itu digunakan untuk mengatur lingkungan Python virtual.

tes

Direktori yang berisi pengujian untuk proyek CDK Anda.

Berikut ini adalah contoh dari unit test:

```
import aws_cdk as core
import aws_cdk.assertions as assertions

from my_cdk_py_project.my_cdk_py_project_stack import MyCdkPyProjectStack

def test_sqs_queue_created():
 app = core.App()
 stack = MyCdkPyProjectStack(app, "my-cdk-py-project")
 template = assertions.Template.from_stack(stack)

 template.has_resource_properties("AWS::SQS::Queue", {
  "VisibilityTimeout": 300
 })
```

Java

Berikut ini adalah contoh proyek yang dibuat dalam `my-cdk-java-project` direktori menggunakan `cdk init --language java` perintah:

```
my-cdk-java-project
### .git
### .gitignore
### README.md
### cdk.json
### pom.xml
### src
    ### main
    ### test
```

pom.xml

File yang berisi informasi konfigurasi dan metadata tentang proyek CDK Anda. File ini adalah bagian dariMaven.

src/utama

Direktori yang berisi file aplikasi dan tumpukan Anda.

Berikut ini adalah contoh file aplikasi:

```
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.Environment;
import software.amazon.awscdk.StackProps;

import java.util.Arrays;

public class MyCdkJavaProjectApp {
 public static void main(final String[] args) {
  App app = new App();

  new MyCdkJavaProjectStack(app, "MyCdkJavaProjectStack", StackProps.builder()
   .build());

  app.synth();
 }
}
```

Berikut ini adalah contoh file stack:

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

public class MyCdkJavaProjectStack extends Stack {
 public MyCdkJavaProjectStack(final Construct scope, final String id) {
  this(scope, id, null);
 }

 public MyCdkJavaProjectStack(final Construct scope, final String id, final
 StackProps props) {
  super(scope, id, props);

  // code that defines your resources and properties go here
 }
}
```

src/tes

Direktori yang berisi file pengujian Anda. Berikut ini adalah contohnya:

```
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.assertions.Template;
import java.io.IOException;

import java.util.HashMap;

import org.junit.jupiter.api.Test;

public class MyCdkJavaProjectTest {

 @Test
 public void testStack() throws IOException {
  App app = new App();
  MyCdkJavaProjectStack stack = new MyCdkJavaProjectStack(app, "test");

  Template template = Template.fromStack(stack);

  template.hasResourceProperties("AWS::SQS::Queue", new HashMap<String, Number>()
 {{
   put("VisibilityTimeout", 300);
  }});
 }
}
```

C#

Berikut ini adalah contoh proyek yang dibuat dalam `my-cdk-csharp-project` direktori menggunakan `cdk init --language csharp` perintah:

```
my-cdk-csharp-project
### .git
### .gitignore
### README.md
### cdk.json
### src
    ### MyCdkCsharpProject
```

```
    ### MyCdkCsharpProject.sln
```

src/ MyCdkCsharpProject

Direktori yang berisi file aplikasi dan tumpukan Anda.

Berikut ini adalah contoh file aplikasi:

```
using Amazon.CDK;
using System;
using System.Collections.Generic;
using System.Linq;

namespace MyCdkCsharpProject
{
 sealed class Program
 {
  public static void Main(string[] args)
  {
   var app = new App();
   new MyCdkCsharpProjectStack(app, "MyCdkCsharpProjectStack", new StackProps{});
   app.Synth();
  }
 }
}
```

Berikut ini adalah contoh file stack:

```
using Amazon.CDK;
using Constructs;

namespace MyCdkCsharpProject
{
 public class MyCdkCsharpProjectStack : Stack
 {
  internal MyCdkCsharpProjectStack(Construct scope, string id, IStackProps props
 = null) : base(scope, id, props)
  {
   // code that defines your resources and properties go here
  }
 }
}
```

Direktori ini juga berisi yang berikut:

- `GlobalSuppressions.cs`— File yang digunakan untuk menekan peringatan atau kesalahan kompiler tertentu di seluruh proyek Anda.
- `.csproj`— File berbasis XML yang digunakan untuk menentukan pengaturan proyek, dependensi, dan membangun konfigurasi.

src/ MyCdkCsharpProject .sln

Microsoft Visual Studio Solution Filedigunakan untuk mengatur dan mengelola proyek terkait.

Go

Berikut ini adalah contoh proyek yang dibuat dalam `my-cdk-go-project` direktori menggunakan `cdk init --language go` perintah:

```
my-cdk-go-project
### .git
### .gitignore
### README.md
### cdk.json
### go.mod
### my-cdk-go-project.go
### my-cdk-go-project_test.go
```

go.mod

File yang berisi informasi modul dan digunakan untuk mengelola dependensi dan pembuatan versi untuk proyek Anda. Go

my-cdk-go-project.pergi

File yang mendefinisikan aplikasi dan tumpukan CDK Anda.

Berikut ini adalah contohnya:

```
package main
import (
 "github.com/aws/aws-cdk-go/awscdk/v2"
 "github.com/aws/constructs-go/constructs/v10"
 "github.com/aws/jsii-runtime-go"
)
```

```
type MyCdkGoProjectStackProps struct {
 awscdk.StackProps
}

func NewMyCdkGoProjectStack(scope constructs.Construct, id string, props
 *MyCdkGoProjectStackProps) awscdk.Stack {
 var sprops awscdk.StackProps
 if props != nil {
  sprops = props.StackProps
 }
 stack := awscdk.NewStack(scope, &id, &sprops)
 // The code that defines your resources and properties go here

  return stack
}

func main() {
 defer jsii.Close()
 app := awscdk.NewApp(nil)
 NewMyCdkGoProjectStack(app, "MyCdkGoProjectStack", &MyCdkGoProjectStackProps{
  awscdk.StackProps{
   Env: env(),
  },
 })
 app.Synth(nil)
}

func env() *awscdk.Environment {

 return nil
}
```

my-cdk-go-project_test.go

File yang mendefinisikan uji sampel.

Berikut ini adalah contohnya:

```
package main

import (
 "testing"

 "github.com/aws/aws-cdk-go/awscdk/v2"
```

```
 "github.com/aws/aws-cdk-go/awscdk/v2/assertions"
 "github.com/aws/jsii-runtime-go"
)

func TestMyCdkGoProjectStack(t *testing.T) {

 // GIVEN
 app := awscdk.NewApp(nil)

 // WHEN
 stack := NewMyCdkGoProjectStack(app, "MyStack", nil)

 // THEN
 template := assertions.Template_FromStack(stack, nil)
 template.HasResourceProperties(jsii.String("AWS::SQS::Queue"),
 map[string]interface{}{
  "VisibilityTimeout": 300,
 })
}
```

AWS CDK aplikasi

AWS Cloud Development Kit (AWS CDK) Aplikasi atau aplikasi adalah kumpulan dari satu atau lebih tumpukan CDK. Tumpukan adalah kumpulan dari satu atau lebih konstruksi, yang mendefinisikan AWS sumber daya dan properti. Oleh karena itu, pengelompokan keseluruhan tumpukan dan konstruksi Anda dikenal sebagai aplikasi CDK Anda.

Topik

- Mendefinisikan aplikasi
- Pohon konstruksi
- Siklus hidup aplikasi

Mendefinisikan aplikasi

Anda membuat aplikasi dengan mendefinisikan instance aplikasi dalam file aplikasi project Anda. Untuk melakukan ini, Anda mengimpor dan menggunakan Appkonstruksi dari Construct AWS Library. AppKonstruk tidak memerlukan argumen inisialisasi apa pun. Ini adalah satu-satunya konstruksi yang dapat digunakan sebagai root.

`Stack`Kelas `App` dan dari Perpustakaan AWS Konstruksi adalah konstruksi yang unik. Dibandingkan dengan konstruksi lain, mereka tidak mengonfigurasi AWS sumber daya sendiri. Sebaliknya, mereka digunakan untuk menyediakan konteks untuk konstruksi Anda yang lain. Semua konstruksi yang mewakili AWS sumber daya harus didefinisikan, secara langsung atau tidak langsung, dalam lingkup konstruksi. `Stack` `Stack`konstruksi didefinisikan dalam lingkup `App` konstruksi.

Aplikasi kemudian disintesis untuk membuat AWS CloudFormation template untuk tumpukan Anda. Berikut ini adalah contohnya:

TypeScript

```
const app = new App();
new MyFirstStack(app, 'hello-cdk');
app.synth();
```

JavaScript

```
const app = new App();
new MyFirstStack(app, 'hello-cdk');
app.synth();
```

Python

```
app = App()
MyFirstStack(app, "hello-cdk")
app.synth()
```

Java

```
App app = new App();
new MyFirstStack(app, "hello-cdk");
app.synth();
```

C#

```
var app = new App();
new MyFirstStack(app, "hello-cdk");
app.Synth();
```

Go

```
app := awscdk.NewApp(nil)

MyFirstStack(app, "MyFirstStack", &MyFirstStackProps{
  awscdk.StackProps{
    Env: env(),
  },
})

app.Synth(nil)
```

Tumpukan dalam satu aplikasi dapat dengan mudah merujuk ke sumber daya dan properti masing-masing. AWS CDK Menyimpulkan dependensi antar tumpukan sehingga dapat digunakan dalam urutan yang benar. Anda dapat menerapkan salah satu atau semua tumpukan dalam aplikasi dengan satu `cdk deploy` perintah.

Pohon konstruksi

Konstruksi didefinisikan di dalam konstruksi lain menggunakan `scope` argumen yang diteruskan ke setiap konstruksi, dengan `App` kelas sebagai root. Dengan cara ini, AWS CDK aplikasi mendefinisikan hierarki konstruksi yang dikenal sebagai pohon konstruksi.

Akar pohon ini adalah aplikasi Anda, yang merupakan instance dari `App` kelas. Di dalam aplikasi, Anda membuat instance satu atau lebih tumpukan. Di dalam tumpukan, Anda membuat instance konstruksi, yang mungkin membuat instance sumber daya atau konstruksi lain, dan seterusnya di bawah pohon.

Konstruksi selalu didefinisikan secara eksplisit dalam lingkup konstruksi lain, yang menciptakan hubungan antar konstruksi. Hampir selalu, Anda harus meneruskan `this` (dengan Python,`self`) sebagai ruang lingkup, yang menunjukkan bahwa konstruksi baru adalah anak dari konstruksi saat ini. Pola yang dimaksud adalah Anda mendapatkan konstruksi Anda `Construct`, lalu membuat instance konstruksi yang digunakannya dalam konstruktornya.

Melewati ruang lingkup secara eksplisit memungkinkan setiap konstruksi untuk menambahkan dirinya ke pohon, dengan perilaku ini sepenuhnya terkandung dalam kelas dasar. `Construct` Ia bekerja dengan cara yang sama dalam setiap bahasa yang didukung oleh AWS CDK dan tidak memerlukan kustomisasi tambahan.

> **Important**
>
> Secara teknis, dimungkinkan untuk melewati beberapa ruang lingkup selain `this` saat membuat instance konstruksi. Anda dapat menambahkan konstruksi di mana saja di pohon, atau bahkan di tumpukan lain di aplikasi yang sama. Misalnya, Anda bisa menulis fungsi gaya mixin yang menambahkan konstruksi ke lingkup yang diteruskan sebagai argumen. Kesulitan praktis di sini adalah Anda tidak dapat dengan mudah memastikan bahwa ID yang Anda pilih untuk konstruksi Anda unik dalam lingkup orang lain. Praktik ini juga membuat kode Anda lebih sulit untuk dipahami, dipelihara, dan digunakan kembali. Hampir selalu lebih baik untuk menemukan cara untuk mengekspresikan niat Anda tanpa harus menyalahgunakan argumen. `scope`

AWS CDK Menggunakan ID dari semua konstruksi di jalur dari akar pohon ke setiap konstruksi anak untuk menghasilkan ID unik yang diperlukan oleh. AWS CloudFormation Pendekatan ini berarti bahwa ID konstruksi hanya perlu unik dalam cakupannya, bukan di dalam seluruh tumpukan seperti di asli AWS CloudFormation. Namun, jika Anda memindahkan konstruksi ke cakupan yang berbeda, ID unik tumpukan yang dihasilkan akan berubah, dan tidak AWS CloudFormation akan menganggapnya sebagai sumber daya yang sama.

Pohon konstruksi terpisah dari konstruksi yang Anda tentukan dalam kode Anda AWS CDK . Namun, ini dapat diakses melalui `node` atribut konstruksi apa pun, yang merupakan referensi ke simpul yang mewakili konstruksi itu di pohon. Setiap node adalah sebuah Nodeinstance, atribut yang menyediakan akses ke akar pohon dan ke lingkup induk node dan anak-anak.

1. `node.children`— Anak-anak langsung dari konstruksi.
2. `node.id`— Pengidentifikasi konstruksi dalam ruang lingkupnya.
3. `node.path`— Jalur lengkap konstruksi termasuk ID semua orang tuanya.
4. `node.root`— Akar pohon konstruksi (aplikasi).
5. `node.scope`— Lingkup (induk) dari konstruksi, atau undefined jika node adalah root.
6. `node.scopes`— Semua orang tua dari konstruksi, sampai ke akar.
7. `node.uniqueId`— Pengidentifikasi alfanumerik unik untuk konstruksi ini di dalam pohon (secara default, dihasilkan dari `node.path` dan hash).

Pohon konstruksi mendefinisikan urutan implisit di mana konstruksi disintesis ke sumber daya dalam template akhir. AWS CloudFormation Dimana satu sumber daya harus dibuat sebelum yang lain,

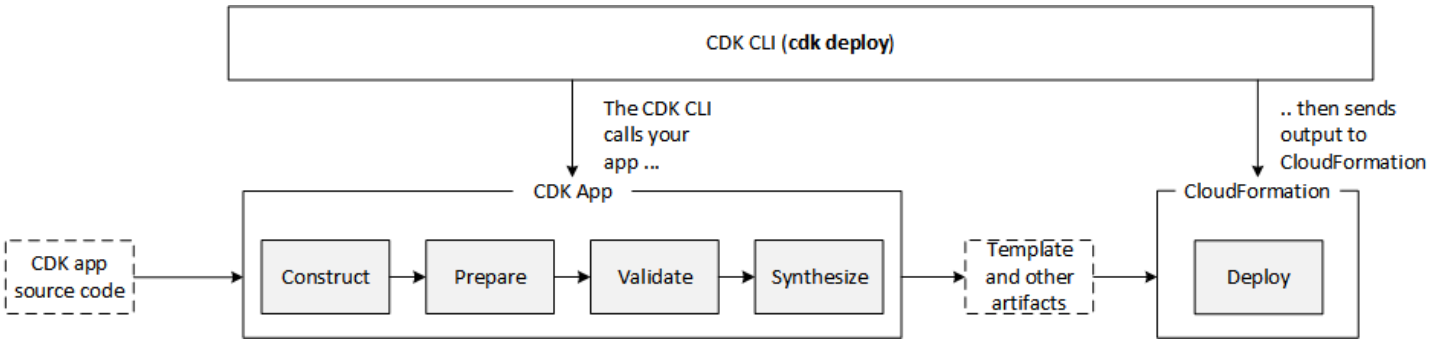
AWS CloudFormation atau AWS Construct Library umumnya menyimpulkan ketergantungan. Mereka kemudian memastikan bahwa sumber daya dibuat dalam urutan yang benar.

Anda juga dapat menambahkan ketergantungan eksplisit antara dua node dengan menggunakan. `node.addDependency()` Untuk informasi selengkapnya, lihat Dependensi di Referensi AWS CDK API.

AWS CDK Ini menyediakan cara sederhana untuk mengunjungi setiap node di pohon konstruksi dan melakukan operasi pada masing-masing node. Untuk informasi selengkapnya, lihat the section called “Aspek”.

Siklus hidup aplikasi

Saat Anda menerapkan aplikasi CDK, fase berikut akan berlangsung. Ini dikenal sebagai siklus hidup aplikasi:



AWS CDK Aplikasi melewati fase berikut dalam siklus hidupnya.

- Konstruksi (atau Inisialisasi) - Kode Anda membuat instance semua konstruksi yang ditentukan dan kemudian menautkannya bersama-sama. Pada tahap ini, semua konstruksi (aplikasi, tumpukan, dan konstruksi anak mereka) dipakai dan rantai konstruktor dijalankan. Sebagian besar kode aplikasi Anda dijalankan pada tahap ini.
- Persiapan — Semua konstruksi yang telah menerapkan `prepare` metode berpartisipasi dalam putaran akhir modifikasi, untuk mengatur keadaan akhir mereka. Fase persiapan terjadi secara otomatis. Sebagai pengguna, Anda tidak melihat umpan balik dari fase ini. Jarang perlu menggunakan kait “siapkan”, dan umumnya tidak disarankan. Berhati-hatilah saat memutasi pohon konstruksi selama fase ini, karena urutan operasi dapat memengaruhi perilaku.
- Validasi — Semua konstruksi yang telah menerapkan `validate` metode dapat memvalidasi diri mereka sendiri untuk memastikan bahwa mereka berada dalam keadaan yang akan diterapkan dengan benar. Anda akan mendapatkan pemberitahuan tentang kegagalan validasi yang terjadi selama fase ini. Umumnya, kami merekomendasikan untuk melakukan validasi sesegera mungkin

(biasanya segera setelah Anda mendapatkan beberapa masukan) dan melempar pengecualian sedini mungkin. Melakukan validasi lebih awal meningkatkan keandalan karena jejak tumpukan akan lebih akurat, dan memastikan bahwa kode Anda dapat terus dijalankan dengan aman.

- Sintesis — Ini adalah tahap akhir dari eksekusi AWS CDK aplikasi Anda. Ini dipicu oleh panggilan keapp.synth(), dan melintasi pohon konstruksi dan memanggil `synthesize` metode pada semua konstruksi. Konstruksi yang mengimplementasikan `synthesize` dapat berpartisipasi dalam sintesis dan memancarkan artefak penyebaran ke perakitan cloud yang dihasilkan. Artefak ini termasuk AWS CloudFormation template, bundel AWS Lambda aplikasi, aset file dan Docker gambar, dan artefak penyebaran lainnya. the section called “Rakitan awan”menjelaskan output dari fase ini. Dalam kebanyakan kasus, Anda tidak perlu menerapkan `synthesize` metode ini.
- Deployment — Pada fase ini, AWS CDK CLI mengambil perakitan cloud artefak penyebaran yang dihasilkan oleh fase sintesis dan menyebarkannya ke lingkungan. AWS Ini mengunggah aset ke Amazon S3 dan Amazon ECR, atau ke mana pun mereka harus pergi. Kemudian, ia memulai AWS CloudFormation penyebaran untuk menyebarkan aplikasi dan membuat sumber daya.

Pada saat fase AWS CloudFormation penerapan dimulai, AWS CDK aplikasi Anda telah selesai dan keluar. Ini memiliki implikasi sebagai berikut:

- AWS CDK Aplikasi tidak dapat merespons peristiwa yang terjadi selama penerapan, seperti sumber daya yang sedang dibuat atau seluruh penyelesaian penerapan. Untuk menjalankan kode selama fase penerapan, Anda harus menyuntikkannya ke AWS CloudFormation template sebagai sumber daya khusus. Untuk informasi selengkapnya tentang menambahkan resource kustom ke aplikasi Anda, lihat AWS CloudFormation modul, atau contoh sumber daya khusus.
- AWS CDK Aplikasi mungkin harus bekerja dengan nilai yang tidak dapat diketahui pada saat dijalankan. Misalnya, jika AWS CDK aplikasi mendefinisikan bucket Amazon S3 dengan nama yang dibuat secara otomatis, dan Anda mengambil atribut `bucket.bucketName` (Pythonbucket_name:), nilai tersebut bukanlah nama bucket yang diterapkan. Sebaliknya, Anda mendapatkan `Token` nilai. Untuk menentukan apakah nilai tertentu tersedia, panggil `cdk.isUnresolved(value)` (Python:`is_unresolved`). Lihat the section called “Token” untuk detail.

Rakitan awan

Panggilan ke `app.synth()` adalah apa yang memberitahu AWS CDK untuk mensintesis perakitan cloud dari sebuah aplikasi. Biasanya Anda tidak berinteraksi langsung dengan rakitan cloud. Mereka adalah file yang menyertakan semua yang diperlukan untuk menerapkan aplikasi Anda ke lingkungan

cloud. Misalnya, ini menyertakan AWS CloudFormation template untuk setiap tumpukan di aplikasi Anda. Ini juga menyertakan salinan aset file atau gambar Docker apa pun yang Anda referensikan di aplikasi Anda.

Lihat spesifikasi perakitan cloud untuk detail tentang cara rakitan cloud diformat.

Untuk berinteraksi dengan rakitan cloud yang dibuat AWS CDK aplikasi Anda, Anda biasanya menggunakan file AWS CDK CLI. Namun, alat apa pun yang dapat membaca format perakitan cloud dapat digunakan untuk menerapkan aplikasi Anda.

Menjalankan aplikasi Anda

CDK CLI perlu tahu cara menjalankan AWS CDK aplikasi Anda. Jika Anda membuat project dari template menggunakan `cdk init` perintah, `cdk.json` file aplikasi Anda menyertakan `app` kunci. Kunci ini menentukan perintah yang diperlukan untuk bahasa yang digunakan aplikasi. Jika bahasa Anda memerlukan kompilasi, baris perintah melakukan langkah ini sebelum menjalankan aplikasi, jadi Anda tidak bisa lupa melakukannya.

TypeScript

```
{
  "app": "npx ts-node --prefer-ts-exts bin/my-app.ts"
}
```

JavaScript

```
{
  "app": "node bin/my-app.js"
}
```

Python

```
{
    "app": "python app.py"
}
```

Java

```
{
  "app": "mvn -e -q compile exec:java"
```

```
}
```

C#

```
{
  "app": "dotnet run -p src/MyApp/MyApp.csproj"
}
```

Go

```
{
  "app": "go mod download && go run my-app.go"
}
```

Jika Anda tidak membuat proyek menggunakan CDKCLI, atau jika Anda ingin mengganti baris perintah yang diberikan`cdk.json`, Anda dapat menggunakan --app opsi saat mengeluarkan perintah. `cdk`

```
$ cdk --app 'executable' cdk-command ...
```

Bagian perintah *`yang dapat dieksekusi`* menunjukkan perintah yang harus dijalankan untuk menjalankan aplikasi CDK Anda. Gunakan tanda kutip seperti yang ditunjukkan, karena perintah tersebut berisi spasi. *`Perintah cdk`* adalah subperintah seperti synth atau deploy yang memberi tahu CDK CLI apa yang ingin Anda lakukan dengan aplikasi Anda. Ikuti ini dengan opsi tambahan apa pun yang diperlukan untuk subperintah itu.

Ini juga AWS CDK CLI dapat berinteraksi langsung dengan rakitan cloud yang sudah disintesis. Untuk melakukan itu, lewati direktori tempat perakitan cloud disimpan--app. Contoh berikut mencantumkan tumpukan yang ditentukan dalam perakitan cloud yang disimpan di bawah`./my-cloud-assembly`.

```
$ cdk --app ./my-cloud-assembly ls
```

Tumpukan

AWS Cloud Development Kit (AWS CDK) Stack adalah kumpulan dari satu atau lebih konstruksi, yang mendefinisikan AWS sumber daya. Setiap tumpukan CDK mewakili AWS CloudFormation tumpukan di aplikasi CDK Anda. Pada penerapan, konstruksi dalam tumpukan disediakan sebagai

satu unit, yang disebut tumpukan. AWS CloudFormation Untuk mempelajari lebih lanjut tentang AWS CloudFormation tumpukan, lihat Bekerja dengan tumpukan di AWS CloudFormation Panduan Pengguna.

Karena tumpukan CDK diimplementasikan melalui AWS CloudFormation tumpukan, AWS CloudFormation kuota dan batasan berlaku. Untuk mempelajari lebih lanjut, lihat AWS CloudFormation kuota.

Topik

Mendefinisikan tumpukan

Tumpukan didefinisikan dalam konteks aplikasi. Anda mendefinisikan tumpukan menggunakan `Stack` kelas dari AWS Construct Library. Tumpukan dapat didefinisikan dengan salah satu cara berikut:

- Langsung dalam lingkup aplikasi.
- Secara tidak langsung oleh konstruksi apa pun di dalam pohon.

Contoh berikut mendefinisikan aplikasi CDK yang berisi dua tumpukan:

TypeScript

```
const app = new App();

new MyFirstStack(app, 'stack1');
new MySecondStack(app, 'stack2');

app.synth();
```

JavaScript

```
const app = new App();

new MyFirstStack(app, 'stack1');
new MySecondStack(app, 'stack2');
```

```
app.synth();
```

Python

```
app = App()

MyFirstStack(app, 'stack1')
MySecondStack(app, 'stack2')

app.synth()
```

Java

```
App app = new App();

new MyFirstStack(app, "stack1");
new MySecondStack(app, "stack2");

app.synth();
```

C#

```
var app = new App();

new MyFirstStack(app, "stack1");
new MySecondStack(app, "stack2");

app.Synth();
```

Contoh berikut adalah pola umum untuk mendefinisikan tumpukan pada file terpisah. Di sini, kita memperluas atau mewarisi `Stack` kelas dan mendefinisikan konstruktor yang menerima`scope`,, `id` dan. `props` Kemudian, kita memanggil konstruktor `Stack` kelas dasar menggunakan `super` dengan yang diterima`scope`,`id`, dan. `props`

TypeScript

```
class HelloCdkStack extends Stack {
  constructor(scope: App, id: string, props?: StackProps) {
    super(scope, id, props);
```

```
    //...
  }
}
```

JavaScript

```
class HelloCdkStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    //...
  }
}
```

Python

```
class HelloCdkStack(Stack):

    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # ...
```

Java

```
public class HelloCdkStack extends Stack {
    public HelloCdkStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloCdkStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        // ...
    }
}
```

C#

```
public class HelloCdkStack : Stack
{
```

```
    public HelloCdkStack(Construct scope, string id, IStackProps props=null) :
 base(scope, id, props)
    {
        //...
    }
}
```

Go

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

  return stack
}
```

Contoh berikut mendeklarasikan kelas tumpukan bernama `MyFirstStack` yang menyertakan satu bucket Amazon S3.

TypeScript

```
class MyFirstStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket');
  }
}
```

JavaScript

```
class MyFirstStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket');
  }
```

```
}
```

Python

```
class MyFirstStack(Stack):

    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        s3.Bucket(self, "MyFirstBucket")
```

Java

```
public class MyFirstStack extends Stack {
    public MyFirstStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyFirstStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        new Bucket(this, "MyFirstBucket");
    }
}
```

C#

```
public class MyFirstStack : Stack
{
    public MyFirstStack(Stack scope, string id, StackProps props = null) :
 base(scope, id, props)
    {
        new Bucket(this, "MyFirstBucket");
    }
}
```

Go

```
func MyFirstStack(scope constructs.Construct, id string, props *MyFirstStackProps)
 awscdk.Stack {
  var sprops awscdk.StackProps
```

```
  if props != nil {
    sprops = props.StackProps
  }
  stack := awscdk.NewStack(scope, &id, &sprops)

  s3.NewBucket(stack, jsii.String("MyFirstBucket"), &s3.BucketProps{})
  return stack
}
```

Namun, kode ini hanya mendeklarasikan tumpukan. Agar tumpukan benar-benar disintesis menjadi AWS CloudFormation template dan digunakan, itu harus dipakai. Dan, seperti semua konstruksi CDK, itu harus dipakai dalam beberapa konteks. Konteksnya `App` adalah itu.

Jika Anda menggunakan template AWS CDK pengembangan standar, tumpukan Anda akan dipakai dalam file yang sama tempat Anda membuat instance objek. `App`

TypeScript

File yang dinamai setelah proyek Anda (misalnya,`hello-cdk.ts`) di `bin` folder proyek Anda.

JavaScript

File yang dinamai setelah proyek Anda (misalnya,`hello-cdk.js`) di `bin` folder proyek Anda.

Python

File `app.py` di direktori utama proyek Anda.

Java

File bernama*`ProjectName`*`App.java`, misalnya`HelloCdkApp.java`, bersarang jauh di bawah `src/main` direktori.

C#

File bernama `Program.cs` di bawah`src\`*`ProjectName`*, misalnya`src\HelloCdk\Program.cs`.

API tumpukan

Objek Stack menyediakan API kaya, termasuk yang berikut ini:

- `Stack.of(construct)`— Sebuah metode statis yang mengembalikan Stack di mana konstruksi didefinisikan. Ini berguna jika Anda perlu berinteraksi dengan tumpukan dari dalam konstruksi

yang dapat digunakan kembali. Panggilan gagal jika tumpukan tidak dapat ditemukan dalam ruang lingkup.

- `stack.stackName`(Python:`stack_name`) — Mengembalikan nama fisik tumpukan. Seperti disebutkan sebelumnya, semua AWS CDK tumpukan memiliki nama fisik yang AWS CDK dapat diselesaikan selama sintesis.
- `stack.region`dan `stack.account` — Kembalikan AWS Wilayah dan akun, masing-masing, ke mana tumpukan ini akan digunakan. Properti ini mengembalikan salah satu dari yang berikut:
  - Akun atau Wilayah secara eksplisit ditentukan saat tumpukan ditentukan
  - Token yang disandikan string yang menyelesaikan parameter AWS CloudFormation semu untuk akun dan Wilayah untuk menunjukkan bahwa tumpukan ini adalah lingkungan agnostik

  Untuk informasi tentang bagaimana lingkungan ditentukan untuk tumpukan, lihatthe section called “Lingkungan”.
- `stack.addDependency(stack)`(Python: `stack.add_dependency(stack)` — Dapat digunakan untuk secara eksplisit mendefinisikan urutan ketergantungan antara dua tumpukan. Urutan ini dihormati oleh cdk deploy perintah saat menerapkan beberapa tumpukan sekaligus.
- `stack.tags`— Mengembalikan TagManageryang dapat Anda gunakan untuk menambah atau menghapus tag tingkat tumpukan. Manajer tag ini menandai semua sumber daya di dalam tumpukan, dan juga menandai tumpukan itu sendiri saat dibuat AWS CloudFormation.
- `stack.partition`, `stack.urlSuffix` (Python:`url_suffix`), (`stack.stackId`Python:`stack_id`), dan (`stack.notificationArn`Python:`notification_arn`) — Kembalikan token yang menyelesaikan parameter semu masing-masing AWS CloudFormation , seperti. `{ "Ref": "AWS::Partition" }` Token ini dikaitkan dengan objek tumpukan tertentu sehingga AWS CDK kerangka kerja dapat mengidentifikasi referensi cross-stack.
- `stack.availabilityZones`(Python:`availability_zones`) — Mengembalikan set Availability Zones yang tersedia di lingkungan tempat tumpukan ini digunakan. Untuk tumpukan agnostik lingkungan, ini selalu mengembalikan array dengan dua Availability Zones. Untuk tumpukan khusus lingkungan, AWS CDK kueri lingkungan dan menampilkan set persis Availability Zone yang tersedia di Wilayah yang Anda tentukan.
- `stack.parseArn(arn)`dan `stack.formatArn(comps)` (Python:`parse_arn`,`format_arn`) - Dapat digunakan untuk bekerja dengan Amazon Resource Names (ARN).
- `stack.toJsonString(obj)`(Python:`to_json_string`) - Dapat digunakan untuk memformat objek arbitrer sebagai string JSON yang dapat disematkan dalam template. AWS CloudFormation

Objek dapat menyertakan token, atribut, dan referensi, yang hanya diselesaikan selama penerapan.

- `stack.templateOptions`(Python:`template_options`) — Gunakan untuk menentukan opsi AWS CloudFormation template, seperti Transform, Description, dan Metadata, untuk tumpukan Anda.

Menggunakan tumpukan

Untuk mencantumkan semua tumpukan di aplikasi CDK, gunakan perintah. cdk ls Contoh sebelumnya akan menampilkan yang berikut:

```
stack1
stack2
```

Tumpukan digunakan sebagai bagian dari AWS CloudFormation tumpukan ke lingkungan AWS . Lingkungan mencakup spesifik Akun AWS dan Wilayah AWS.

Saat Anda menjalankan cdk synth perintah untuk aplikasi dengan beberapa tumpukan, rakitan cloud menyertakan templat terpisah untuk setiap instance tumpukan. Bahkan jika dua tumpukan adalah contoh dari kelas yang sama, AWS CDK memancarkannya sebagai dua templat individual.

Anda dapat mensintesis setiap template dengan menentukan nama tumpukan dalam perintah. cdk synth Contoh berikut mensintesis template untuk stack1.

```
$ cdk synth stack1
```

Pendekatan ini secara konseptual berbeda dari bagaimana AWS CloudFormation template biasanya digunakan, di mana template dapat digunakan beberapa kali dan diparameterisasi melalui parameter.AWS CloudFormation Meskipun AWS CloudFormation parameter dapat didefinisikan dalam AWS CDK, mereka umumnya tidak disarankan karena AWS CloudFormation parameter diselesaikan hanya selama penerapan. Ini berarti Anda tidak dapat menentukan nilainya dalam kode Anda.

Misalnya, untuk menyertakan sumber daya secara kondisional di aplikasi berdasarkan nilai parameter, Anda harus menyiapkan AWS CloudFormation kondisi dan menandai sumber daya dengannya. Ini AWS CDK mengambil pendekatan di mana templat konkret diselesaikan pada waktu sintesis. Oleh karena itu, Anda dapat menggunakan pernyataan if untuk memeriksa nilai untuk menentukan apakah sumber daya harus didefinisikan atau beberapa perilaku harus diterapkan.

> **Note**
>
> AWS CDK Ini memberikan resolusi sebanyak mungkin selama waktu sintesis untuk memungkinkan penggunaan bahasa pemrograman Anda secara idiomatik dan alami.

Seperti konstruksi lainnya, tumpukan dapat disusun bersama menjadi beberapa kelompok. Kode berikut menunjukkan contoh layanan yang terdiri dari tiga tumpukan: bidang kontrol, bidang data, dan tumpukan pemantauan. Konstruksi layanan didefinisikan dua kali: sekali untuk lingkungan beta dan sekali untuk lingkungan produksi.

TypeScript

```
import { App, Stack } from 'aws-cdk-lib';
import { Construct } from 'constructs';

interface EnvProps {
  prod: boolean;
}

// imagine these stacks declare a bunch of related resources
class ControlPlane extends Stack {}
class DataPlane extends Stack {}
class Monitoring extends Stack {}

class MyService extends Construct {

  constructor(scope: Construct, id: string, props?: EnvProps) {

    super(scope, id);

    // we might use the prod argument to change how the service is configured
    new ControlPlane(this, "cp");
    new DataPlane(this, "data");
    new Monitoring(this, "mon");  }
}

const app = new App();
new MyService(app, "beta");
new MyService(app, "prod", { prod: true });
```

```
app.synth();
```

JavaScript

```
const { App, Stack } = require('aws-cdk-lib');
const { Construct } = require('constructs');

// imagine these stacks declare a bunch of related resources
class ControlPlane extends Stack {}
class DataPlane extends Stack {}
class Monitoring extends Stack {}

class MyService extends Construct {

  constructor(scope, id, props) {

    super(scope, id);

    // we might use the prod argument to change how the service is configured
    new ControlPlane(this, "cp");
    new DataPlane(this, "data");
    new Monitoring(this, "mon");
  }
}

const app = new App();
new MyService(app, "beta");
new MyService(app, "prod", { prod: true });

app.synth();
```

Python

```
from aws_cdk import App, Stack
from constructs import Construct

# imagine these stacks declare a bunch of related resources
class ControlPlane(Stack): pass
class DataPlane(Stack): pass
class Monitoring(Stack): pass

class MyService(Construct):
```

```
  def __init__(self, scope: Construct, id: str, *, prod=False):

    super().__init__(scope, id)

    # we might use the prod argument to change how the service is configured
    ControlPlane(self, "cp")
    DataPlane(self, "data")
    Monitoring(self, "mon")

app = App();
MyService(app, "beta")
MyService(app, "prod", prod=True)

app.synth()
```

Java

```
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.Stack;
import software.constructs.Construct;

public class MyApp {

    // imagine these stacks declare a bunch of related resources
    static class ControlPlane extends Stack {
        ControlPlane(Construct scope, String id) {
            super(scope, id);
        }
    }

    static class DataPlane extends Stack {
        DataPlane(Construct scope, String id) {
            super(scope, id);
        }
    }

    static class Monitoring extends Stack {
        Monitoring(Construct scope, String id) {
            super(scope, id);
        }
    }
```

```
    static class MyService extends Construct {
        MyService(Construct scope, String id) {
            this(scope, id, false);
        }

        MyService(Construct scope, String id, boolean prod) {
            super(scope, id);

            // we might use the prod argument to change how the service is
 configured
            new ControlPlane(this, "cp");
            new DataPlane(this, "data");
            new Monitoring(this, "mon");
        }
    }

    public static void main(final String argv[]) {
        App app = new App();

        new MyService(app, "beta");
        new MyService(app, "prod", true);

        app.synth();
    }
}
```

C#

```
using Amazon.CDK;
using Constructs;

// imagine these stacks declare a bunch of related resources
public class ControlPlane : Stack {
    public ControlPlane(Construct scope, string id=null) : base(scope, id) { }
}

public class DataPlane : Stack {
    public DataPlane(Construct scope, string id=null) : base(scope, id) { }
}

public class Monitoring : Stack
{
```

```
    public Monitoring(Construct scope, string id=null) : base(scope, id) { }
}

public class MyService : Construct
{
    public MyService(Construct scope, string id, Boolean prod=false) : base(scope,
 id)
    {
        // we might use the prod argument to change how the service is configured
        new ControlPlane(this, "cp");
        new DataPlane(this, "data");
        new Monitoring(this, "mon");
    }
}

class Program
{
    static void Main(string[] args)
    {

        var app = new App();
        new MyService(app, "beta");
        new MyService(app, "prod", prod: true);
        app.Synth();
    }
}
```

AWS CDK Aplikasi ini akhirnya terdiri dari enam tumpukan, tiga untuk setiap lingkungan:

```
$ cdk ls

betacpDA8372D3
betadataE23DB2BA
betamon632BD457
prodcp187264CE
proddataF7378CE5
prodmon631A1083
```

Nama fisik AWS CloudFormation tumpukan secara otomatis ditentukan oleh AWS CDK berdasarkan jalur konstruksi tumpukan di pohon. Secara default, nama tumpukan berasal dari ID konstruksi `Stack`

objek. Namun, Anda dapat menentukan nama eksplisit dengan menggunakan `stackName` prop (dalam Python,`stack_name`), sebagai berikut.

TypeScript

```
new MyStack(this, 'not:a:stack:name', { stackName: 'this-is-stack-name' });
```

JavaScript

```
new MyStack(this, 'not:a:stack:name', { stackName: 'this-is-stack-name' });
```

Python

```
MyStack(self, "not:a:stack:name", stack_name="this-is-stack-name")
```

Java

```
new MyStack(this, "not:a:stack:name", StackProps.builder()
    .StackName("this-is-stack-name").build());
```

C#

```
new MyStack(this, "not:a:stack:name", new StackProps
{
    StackName = "this-is-stack-name"
});
```

Tumpukan nested

NestedStackKonstruk ini menawarkan jalan di sekitar batas AWS CloudFormation 500 sumber daya untuk tumpukan. Tumpukan bersarang dihitung sebagai hanya satu sumber daya di tumpukan yang berisi itu. Namun, dapat berisi hingga 500 sumber daya, termasuk tumpukan bersarang tambahan.

Ruang lingkup tumpukan bersarang harus berupa `Stack` atau `NestedStack` konstruksi. Tumpukan bersarang tidak perlu dideklarasikan secara leksikal di dalam tumpukan induknya. Anda hanya perlu meneruskan tumpukan induk sebagai parameter pertama (`scope`) saat membuat instance tumpukan bersarang. Selain pembatasan ini, mendefinisikan konstruksi dalam tumpukan bersarang bekerja persis sama seperti di tumpukan biasa.

Pada waktu sintesis, tumpukan bersarang disintesis ke AWS CloudFormation templatnya sendiri, yang diunggah ke bucket AWS CDK pementasan saat penerapan. Tumpukan bersarang terikat pada tumpukan induknya dan tidak diperlakukan sebagai artefak penerapan independen. Mereka tidak terdaftar oleh`cdk list`, dan mereka tidak dapat digunakan oleh`cdk deploy`.

Referensi antara tumpukan induk dan tumpukan bersarang secara otomatis diterjemahkan ke parameter tumpukan dan output dalam AWS CloudFormation templat yang dihasilkan, seperti halnya referensi tumpukan silang.

> **Warning**
>
> Perubahan postur keamanan tidak ditampilkan sebelum penerapan untuk tumpukan bersarang. Informasi ini hanya ditampilkan untuk tumpukan tingkat atas.

Membangun

Konstruksi adalah blok bangunan dasar AWS Cloud Development Kit (AWS CDK) aplikasi. Konstruk adalah komponen dalam aplikasi Anda yang mewakili satu atau lebih AWS CloudFormation sumber daya dan konfigurasinya. Anda membangun aplikasi Anda, sepotong demi sepotong, dengan mengimpor dan mengonfigurasi konstruksi.

Konstruksi adalah kelas yang Anda impor ke aplikasi CDK Anda. Konstruksi tersedia dari AWS Construct Library. Anda juga dapat membuat dan mendistribusikan konstruksi Anda sendiri, atau menggunakan konstruksi yang dibuat oleh pengembang pihak ketiga.

Konstruksi adalah bagian dari Construct Programming Model (CPM). Mereka tersedia untuk digunakan dengan alat lain seperti CDK for Terraform (CdKTF), CDK for Kubernetes (Cdk8s), dan. Projen

Topik

- AWS Membangun Perpustakaan
- Mendefinisikan konstruksi
- Bekerja dengan konstruksi
- Bekerja dengan konstruksi pihak ketiga
- Pelajari selengkapnya

AWS Membangun Perpustakaan

Perpustakaan AWS Construct berisi koleksi konstruksi yang dikembangkan dan dikelola oleh. AWS Ini diatur ke dalam berbagai modul yang berisi konstruksi yang mewakili semua sumber daya yang tersedia di AWS. Untuk informasi referensi, lihat Referensi AWS CDK API.

Paket CDK utama disebut`aws-cdk-lib`, dan berisi sebagian besar AWS Construct Library. Ini juga berisi kelas dasar seperti `Stack` dan`App`.

Nama paket sebenarnya dari paket CDK utama bervariasi menurut bahasa.

TypeScript

Menginstal	`npm install aws-cdk-lib`
Import	`import * as cdk from 'aws-cdk-lib';`

JavaScript

Menginstal	`npm install aws-cdk-lib`
Import	`const cdk = require('aws-cdk-lib');`

Python

Menginstal	python -m pip install aws-cdk-lib
Import	`import aws_cdk as cdk`

Java

Di`pom.xml`, tambahkan	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`

Import	`import software.amazon.awscdk.App;`

C#

Menginstal	dotnet add package Amazon.CDK.Lib
Import	`using Amazon.CDK;`

Go

Menginstal	go get github.com/aws/aws-cdk-go/awscdk/v2
Import	`import ( "github.com/aws/aws-cdk-go/awscdk/v2" )`

> **Note**
>
> Jika Anda membuat proyek CDK menggunakancdk init, Anda tidak perlu menginstal `aws-cdk-lib` secara manual.

The AWS Construct Library juga berisi `constructs` paket dengan kelas `Construct` dasar. Ini dalam paketnya sendiri karena digunakan oleh alat berbasis konstruksi lain selain AWS CDK, termasuk CDK untuk Terraform dan CDK untuk Kubernetes.

Banyak pihak ketiga juga telah menerbitkan konstruksi yang kompatibel dengan. AWS CDK Kunjungi Construct Hub untuk menjelajahi ekosistem mitra AWS CDK konstruksi.

Membangun tingkat

Konstruksi dari AWS Construct Library dikategorikan menjadi tiga tingkatan. Setiap tingkat menawarkan tingkat abstraksi yang meningkat. Semakin tinggi abstraksi, semakin mudah

dikonfigurasi, membutuhkan lebih sedikit keahlian. Semakin rendah abstraksi, semakin banyak kustomisasi yang tersedia, membutuhkan lebih banyak keahlian.

Konstruksi level 1 (L1)

Konstruksi L1, juga dikenal sebagai sumber daya CFN, adalah konstruksi tingkat terendah dan tidak menawarkan abstraksi. Setiap konstruksi L1 memetakan langsung ke satu AWS CloudFormation sumber daya. Dengan konstruksi L1, Anda mengimpor konstruksi yang mewakili sumber daya tertentu. AWS CloudFormation Anda kemudian menentukan properti sumber daya dalam instance konstruksi Anda.

Konstruksi L1 sangat bagus untuk digunakan ketika Anda terbiasa AWS CloudFormation dan membutuhkan kontrol penuh untuk mendefinisikan properti sumber daya Anda AWS .

Dalam AWS Construct Library, konstruksi L1 diberi nama dimulai dengan`Cfn`, diikuti oleh identifier untuk AWS CloudFormation sumber daya yang diwakilinya. Misalnya, `CfnBucket` konstruk adalah konstruksi L1 yang mewakili sumber daya. `AWS::S3::Bucket` AWS CloudFormation

Konstruksi L1 dihasilkan dari spesifikasi AWS CloudFormation sumber daya. Jika sumber daya ada di AWS CloudFormation, itu akan tersedia di AWS CDK sebagai konstruksi L1. Sumber daya atau properti baru mungkin membutuhkan waktu hingga satu minggu untuk tersedia di Perpustakaan AWS Konstruksi. Untuk informasi selengkapnya, lihat referensi jenis AWS sumber daya dan properti di Panduan AWS CloudFormation Pengguna.

Konstruksi level 2 (L2)

Konstruksi L2, juga dikenal sebagai konstruksi kurasi, dikembangkan dengan cermat oleh tim CDK dan biasanya merupakan jenis konstruksi yang paling banyak digunakan. Konstruksi L2 memetakan langsung ke AWS CloudFormation sumber daya tunggal, mirip dengan konstruksi L1. Dibandingkan dengan konstruksi L1, konstruksi L2 memberikan abstraksi tingkat yang lebih tinggi melalui API berbasis niat intuitif. Konstruksi L2 mencakup konfigurasi properti default yang masuk akal, kebijakan keamanan praktik terbaik, dan menghasilkan banyak kode boilerplate dan logika lem untuk Anda.

Konstruksi L2 juga menyediakan metode pembantu untuk sebagian besar sumber daya yang membuatnya lebih sederhana dan lebih cepat untuk menentukan properti, izin, interaksi berbasis peristiwa antara sumber daya, dan banyak lagi.

`s3.Bucket`Kelas ini adalah contoh konstruksi L2 untuk sumber daya bucket Amazon Simple Storage Service (Amazon S3).

Perpustakaan AWS Konstruksi berisi konstruksi L2 yang ditunjuk stabil dan siap untuk digunakan produksi. Untuk konstruksi L2 yang sedang dikembangkan, mereka ditunjuk sebagai eksperimental dan ditawarkan dalam modul terpisah.

Konstruksi level 3 (L3)

Konstruksi L3, juga dikenal sebagai pola, adalah abstraksi tingkat tertinggi. Setiap konstruksi L3 dapat berisi kumpulan sumber daya yang dikonfigurasi untuk bekerja sama untuk menyelesaikan tugas atau layanan tertentu dalam aplikasi Anda. Konstruksi L3 digunakan untuk membuat seluruh AWS arsitektur untuk kasus penggunaan tertentu dalam aplikasi Anda.

Untuk menyediakan desain sistem yang lengkap, atau bagian penting dari sistem yang lebih besar, konstruksi L3 menawarkan konfigurasi properti default yang berpendirian. Mereka dibangun di sekitar pendekatan tertentu untuk memecahkan masalah dan memberikan solusi. Dengan konstruksi L3, Anda dapat membuat dan mengonfigurasi beberapa sumber daya dengan cepat, dengan jumlah input dan kode paling sedikit.

`ecsPatterns.ApplicationLoadBalancedFargateService`Kelas adalah contoh konstruksi L3 yang mewakili layanan yang berjalan pada cluster Amazon Elastic Container AWS Fargate Service (Amazon ECS) Container Service (Amazon ECS) dan diawali oleh penyeimbang beban aplikasi.

Mirip dengan konstruksi L2, konstruksi L3 yang siap untuk penggunaan produksi disertakan dalam Construct Library. AWS Mereka yang sedang dikembangkan ditawarkan dalam modul terpisah.

Mendefinisikan konstruksi

Komposisi

Komposisi adalah pola kunci untuk mendefinisikan abstraksi tingkat yang lebih tinggi melalui konstruksi. Sebuah konstruksi tingkat tinggi dapat terdiri dari sejumlah konstruksi tingkat rendah. Dari perspektif bottom-up, Anda menggunakan konstruksi untuk mengatur AWS sumber daya individual yang ingin Anda terapkan. Anda menggunakan abstraksi apa pun yang nyaman untuk tujuan Anda, dengan level sebanyak yang Anda butuhkan.

Dengan komposisi, Anda menentukan komponen yang dapat digunakan kembali dan membagikannya seperti kode lainnya. Misalnya, tim dapat menentukan konstruksi yang mengimplementasikan praktik terbaik perusahaan untuk tabel Amazon DynamoDB, termasuk pencadangan, replikasi global, penskalaan otomatis, dan pemantauan. Tim dapat berbagi konstruksi secara internal dengan tim lain, atau secara publik.

Tim dapat menggunakan konstruksi seperti paket perpustakaan lainnya. Ketika pustaka diperbarui, pengembang mendapatkan akses ke perbaikan versi baru dan perbaikan bug, mirip dengan pustaka kode lainnya.

Inisialisasi

Konstruksi diimplementasikan di kelas yang memperluas kelas `Construct`dasar. Anda mendefinisikan konstruksi dengan membuat instance kelas. Semua konstruksi mengambil tiga parameter saat diinisialisasi:

- lingkup — Orang tua atau pemilik konstruksi. Ini bisa berupa tumpukan atau konstruksi lain. Lingkup menentukan tempat konstruk di pohon konstruksi. Anda biasanya harus melewati `this` (`self`finPython), yang mewakili objek saat ini, untuk ruang lingkup.
- id — Pengenal yang harus unik dalam lingkup. Identifier berfungsi sebagai namespace untuk semua yang didefinisikan dalam konstruksi. Ini digunakan untuk menghasilkan pengidentifikasi unik, seperti nama sumber daya dan ID AWS CloudFormation logis.

  Pengidentifikasi hanya perlu unik dalam ruang lingkup. Ini memungkinkan Anda membuat instance dan menggunakan kembali konstruksi tanpa memperhatikan konstruksi dan pengidentifikasi yang mungkin dikandungnya, dan memungkinkan pembuatan konstruksi menjadi abstraksi tingkat yang lebih tinggi. Selain itu, cakupan memungkinkan untuk merujuk ke kelompok konstruksi sekaligus. Contohnya termasuk untuk penandaan, atau menentukan di mana konstruksi akan digunakan.
- props — Satu set properti atau argumen kata kunci, tergantung pada bahasa, yang menentukan konfigurasi awal konstruksi. Konstruksi tingkat yang lebih tinggi memberikan lebih banyak default, dan jika semua elemen prop opsional, Anda dapat menghilangkan parameter props sepenuhnya.

Konfigurasi

Sebagian besar konstruksi menerima `props` sebagai argumen ketiga mereka (atau dalam Python, argumen kata kunci), kumpulan nama/nilai yang mendefinisikan konfigurasi konstruksi. Contoh berikut mendefinisikan bucket dengan enkripsi AWS Key Management Service (AWS KMS) dan hosting situs web statis diaktifkan. Karena tidak secara eksplisit menentukan kunci enkripsi, `Bucket` konstruksi mendefinisikan yang baru `kms.Key` dan mengaitkannya dengan bucket.

TypeScript

```
new s3.Bucket(this, 'MyEncryptedBucket', {
  encryption: s3.BucketEncryption.KMS,
```

```
  websiteIndexDocument: 'index.html'
});
```

JavaScript

```
new s3.Bucket(this, 'MyEncryptedBucket', {
  encryption: s3.BucketEncryption.KMS,
  websiteIndexDocument: 'index.html'
});
```

Python

```
s3.Bucket(self, "MyEncryptedBucket", encryption=s3.BucketEncryption.KMS,
    website_index_document="index.html")
```

Java

```
Bucket.Builder.create(this, "MyEncryptedBucket")
        .encryption(BucketEncryption.KMS_MANAGED)
        .websiteIndexDocument("index.html").build();
```

C#

```
new Bucket(this, "MyEncryptedBucket", new BucketProps
{
    Encryption = BucketEncryption.KMS_MANAGED,
    WebsiteIndexDocument = "index.html"
});
```

Go

```
 awss3.NewBucket(stack, jsii.String("MyEncryptedBucket"), &awss3.BucketProps{
  Encryption: awss3.BucketEncryption_KMS,
  WebsiteIndexDocument: jsii.String("index.html"),
 })
```

Berinteraksi dengan konstruksi

Konstruksi adalah kelas yang memperluas kelas Konstruksi dasar. Setelah Anda membuat instance konstruksi, objek konstruksi mengekspos satu set metode dan properti yang memungkinkan Anda berinteraksi dengan konstruksi dan meneruskannya sebagai referensi ke bagian lain dari sistem.

AWS CDK Kerangka kerja tidak membatasi API konstruksi. Penulis dapat menentukan API apa pun yang mereka inginkan. Namun, AWS konstruksi yang disertakan dengan AWS Construct Library, sepertis3.Bucket, mengikuti pedoman dan pola umum. Ini memberikan pengalaman yang konsisten di semua AWS sumber daya.

Sebagian besar AWS konstruksi memiliki seperangkat metode hibah yang dapat Anda gunakan untuk memberikan izin AWS Identity and Access Management (IAM) pada konstruksi tersebut ke prinsipal. Contoh berikut memberikan data-science izin grup IAM untuk membaca dari bucket Amazon S3. raw-data

TypeScript

```
const rawData = new s3.Bucket(this, 'raw-data');
const dataScience = new iam.Group(this, 'data-science');
rawData.grantRead(dataScience);
```

JavaScript

```
const rawData = new s3.Bucket(this, 'raw-data');
const dataScience = new iam.Group(this, 'data-science');
rawData.grantRead(dataScience);
```

Python

```
raw_data = s3.Bucket(self, 'raw-data')
data_science = iam.Group(self, 'data-science')
raw_data.grant_read(data_science)
```

Java

```
Bucket rawData = new Bucket(this, "raw-data");
Group dataScience = new Group(this, "data-science");
rawData.grantRead(dataScience);
```

C#

```
var rawData = new Bucket(this, "raw-data");
var dataScience = new Group(this, "data-science");
rawData.GrantRead(dataScience);
```

Go

```
 rawData := awss3.NewBucket(stack, jsii.String("raw-data"), nil)
 dataScience := awsiam.NewGroup(stack, jsii.String("data-science"), nil)
 rawData.GrantRead(dataScience, nil)
```

Pola umum lainnya adalah AWS konstruksi untuk mengatur salah satu atribut sumber daya dari data yang disediakan di tempat lain. Atribut dapat menyertakan Nama Sumber Daya Amazon (ARN), nama, atau URL.

Kode berikut mendefinisikan AWS Lambda fungsi dan mengaitkannya dengan antrian Amazon Simple Queue Service (Amazon SQS) melalui URL antrian dalam variabel lingkungan.

TypeScript

```
const jobsQueue = new sqs.Queue(this, 'jobs');
const createJobLambda = new lambda.Function(this, 'create-job', {
  runtime: lambda.Runtime.NODEJS_18_X,
  handler: 'index.handler',
  code: lambda.Code.fromAsset('./create-job-lambda-code'),
  environment: {
    QUEUE_URL: jobsQueue.queueUrl
  }
});
```

JavaScript

```
const jobsQueue = new sqs.Queue(this, 'jobs');
const createJobLambda = new lambda.Function(this, 'create-job', {
  runtime: lambda.Runtime.NODEJS_18_X,
  handler: 'index.handler',
  code: lambda.Code.fromAsset('./create-job-lambda-code'),
  environment: {
    QUEUE_URL: jobsQueue.queueUrl
  }
```

```
});
```

Python

```
jobs_queue = sqs.Queue(self, "jobs")
create_job_lambda = lambda_.Function(self, "create-job",
    runtime=lambda_.Runtime.NODEJS_18_X,
    handler="index.handler",
    code=lambda_.Code.from_asset("./create-job-lambda-code"),
    environment=dict(
        QUEUE_URL=jobs_queue.queue_url
    )
)
```

Java

```
final Queue jobsQueue = new Queue(this, "jobs");
Function createJobLambda = Function.Builder.create(this, "create-job")
                .handler("index.handler")
                .code(Code.fromAsset("./create-job-lambda-code"))
                .environment(java.util.Map.of(   // Map.of is Java 9 or later
                    "QUEUE_URL", jobsQueue.getQueueUrl())
                .build();
```

C#

```
var jobsQueue = new Queue(this, "jobs");
var createJobLambda = new Function(this, "create-job", new FunctionProps
{
    Runtime = Runtime.NODEJS_18_X,
    Handler = "index.handler",
    Code = Code.FromAsset(@".\create-job-lambda-code"),
    Environment = new Dictionary<string, string>
    {
        ["QUEUE_URL"] = jobsQueue.QueueUrl
    }
});
```

Go

```
 createJobLambda := awslambda.NewFunction(stack, jsii.String("create-job"),
 &awslambda.FunctionProps{
```

```
  Runtime: awslambda.Runtime_NODEJS_18_X(),
  Handler: jsii.String("index.handler"),
  Code:    awslambda.Code_FromAsset(jsii.String(".\\create-job-lambda-code"), nil),
  Environment: &map[string]*string{
   "QUEUE_URL": jsii.String(*jobsQueue.QueueUrl()),
  },
 })
```

Untuk informasi tentang pola API yang paling umum di AWS Construct Library, lihatthe section called "Sumber daya".

Aplikasi dan konstruksi tumpukan

`Stack`Kelas `App` dan dari AWS Construct Library adalah konstruksi yang unik. Dibandingkan dengan konstruksi lain, mereka tidak mengonfigurasi AWS sumber daya sendiri. Sebaliknya, mereka digunakan untuk menyediakan konteks untuk konstruksi Anda yang lain. Semua konstruksi yang mewakili AWS sumber daya harus didefinisikan, secara langsung atau tidak langsung, dalam lingkup konstruksi. `Stack` `Stack`konstruksi didefinisikan dalam lingkup App konstruk.

Untuk mempelajari lebih lanjut tentang aplikasi CDK, lihatAWS CDK aplikasi. Untuk mempelajari lebih lanjut tentang tumpukan CDK, lihat. Tumpukan

Contoh berikut mendefinisikan aplikasi dengan satu tumpukan. Di dalam tumpukan, konstruksi L2 digunakan untuk mengonfigurasi sumber daya bucket Amazon S3.

TypeScript

```
import { App, Stack, StackProps } from 'aws-cdk-lib';
import * as s3 from 'aws-cdk-lib/aws-s3';

class HelloCdkStack extends Stack {
  constructor(scope: App, id: string, props?: StackProps) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}

const app = new App();
```

```
new HelloCdkStack(app, "HelloCdkStack");
```

JavaScript

```
const { App , Stack } = require('aws-cdk-lib');
const s3 = require('aws-cdk-lib/aws-s3');

class HelloCdkStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}

const app = new App();
new HelloCdkStack(app, "HelloCdkStack");
```

Python

```
from aws_cdk import App, Stack
import aws_cdk.aws_s3 as s3
from constructs import Construct

class HelloCdkStack(Stack):

    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        s3.Bucket(self, "MyFirstBucket", versioned=True)

app = App()
HelloCdkStack(app, "HelloCdkStack")
```

Java

Tumpukan didefinisikan dalam `HelloCdkStack.java` file:

```
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
```

```
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.s3.*;

public class HelloCdkStack extends Stack {
    public HelloCdkStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloCdkStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        Bucket.Builder.create(this, "MyFirstBucket")
            .versioned(true).build();
    }
}
```

Aplikasi didefinisikan dalam `HelloCdkApp.java` file:

```
import software.amazon.awscdk.App;
import software.amazon.awscdk.StackProps;

public class HelloCdkApp {
    public static void main(final String[] args) {
        App app = new App();

        new HelloCdkStack(app, "HelloCdkStack", StackProps.builder()
                .build());

        app.synth();
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;

namespace HelloCdkApp
{
    internal static class Program
    {
        public static void Main(string[] args)
```

```
        {
            var app = new App();
            new HelloCdkStack(app, "HelloCdkStack");
            app.Synth();
        }
    }

    public class HelloCdkStack : Stack
    {
        public HelloCdkStack(Construct scope, string id, IStackProps props=null) :
 base(scope, id, props)
        {
            new Bucket(this, "MyFirstBucket", new BucketProps { Versioned = true });
        }
    }
}
```

Go

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

 awss3.NewBucket(stack, jsii.String("MyFirstBucket"), &awss3.BucketProps{
  Versioned: jsii.Bool(true),
 })

 return stack
}
```

Bekerja dengan konstruksi

Bekerja dengan konstruksi L1

L1 membangun peta langsung ke sumber daya individu AWS CloudFormation . Anda harus menyediakan konfigurasi sumber daya yang diperlukan.

Dalam contoh ini, kita membuat `bucket` objek menggunakan konstruksi `CfnBucket` L1:

TypeScript

```
const bucket = new s3.CfnBucket(this, "MyBucket", {
  bucketName: "MyBucket"
});
```

JavaScript

```
const bucket = new s3.CfnBucket(this, "MyBucket", {
  bucketName: "MyBucket"
});
```

Python

```
bucket = s3.CfnBucket(self, "MyBucket", bucket_name="MyBucket")
```

Java

```
CfnBucket bucket = new CfnBucket.Builder().bucketName("MyBucket").build();
```

C#

```
var bucket = new CfnBucket(this, "MyBucket", new CfnBucketProps
{
    BucketName= "MyBucket"
});
```

Go

```
 awss3.NewCfnBucket(stack, jsii.String("MyBucket"), &awss3.CfnBucketProps{
  BucketName: jsii.String("MyBucket"),
 })
```

Properti konstruksi yang bukan Boolean sederhana, string, angka, atau kontainer ditangani secara berbeda dalam bahasa yang didukung.

TypeScript

```
const bucket = new s3.CfnBucket(this, "MyBucket", {
```

```
  bucketName: "MyBucket",
  corsConfiguration: {
    corsRules: [{
          allowedOrigins: ["*"],
          allowedMethods: ["GET"]
    }]
  }
});
```

JavaScript

```
const bucket = new s3.CfnBucket(this, "MyBucket", {
  bucketName: "MyBucket",
  corsConfiguration: {
    corsRules: [{
          allowedOrigins: ["*"],
          allowedMethods: ["GET"]
    }]
  }
});
```

Python

Dalam Python, properti ini diwakili oleh tipe yang didefinisikan sebagai kelas dalam dari konstruksi L1. Misalnya, properti opsional `cors_configuration` dari sebuah `CfnBucket` memerlukan pembungkus tipe`CfnBucket.CorsConfigurationProperty`. Di sini kita mendefinisikan `cors_configuration` sebuah `CfnBucket` contoh.

```
bucket = CfnBucket(self, "MyBucket", bucket_name="MyBucket",
    cors_configuration=CfnBucket.CorsConfigurationProperty(
        cors_rules=[CfnBucket.CorsRuleProperty(
            allowed_origins=["*"],
            allowed_methods=["GET"]
        )]
    )
)
```

Java

Di Jawa, properti ini diwakili oleh tipe yang didefinisikan sebagai kelas dalam dari konstruksi L1. Misalnya, properti opsional `corsConfiguration` dari sebuah `CfnBucket` memerlukan

pembungkus tipe`CfnBucket.CorsConfigurationProperty`. Di sini kita mendefinisikan `corsConfiguration` sebuah `CfnBucket` contoh.

```
CfnBucket bucket = CfnBucket.Builder.create(this, "MyBucket")
                        .bucketName("MyBucket")
                        .corsConfiguration(new
 CfnBucket.CorsConfigurationProperty.Builder()
                            .corsRules(Arrays.asList(new
 CfnBucket.CorsRuleProperty.Builder()
                                .allowedOrigins(Arrays.asList("*"))
                                .allowedMethods(Arrays.asList("GET"))
                                .build()))
                            .build())
                        .build();
```

C#

Dalam C #, properti ini diwakili oleh tipe yang didefinisikan sebagai kelas dalam dari konstruksi L1. Misalnya, properti opsional `CorsConfiguration` dari sebuah `CfnBucket` memerlukan pembungkus tipe`CfnBucket.CorsConfigurationProperty`. Di sini kita mendefinisikan `CorsConfiguration` sebuah `CfnBucket` contoh.

```
var bucket = new CfnBucket(this, "MyBucket", new CfnBucketProps
{
    BucketName = "MyBucket",
    CorsConfiguration = new CfnBucket.CorsConfigurationProperty
    {
        CorsRules = new object[] {
            new CfnBucket.CorsRuleProperty
            {
                AllowedOrigins = new string[] { "*" },
                AllowedMethods = new string[] { "GET" },
            }
        }
    }
});
```

Go

Di Go, jenis ini diberi nama menggunakan nama konstruksi L1, garis bawah, dan nama properti. Misalnya, properti opsional `CorsConfiguration` dari sebuah `CfnBucket` memerlukan

pembungkus tipe`CfnBucket_CorsConfigurationProperty`. Di sini kita mendefinisikan `CorsConfiguration` sebuah `CfnBucket` contoh.

```
awss3.NewCfnBucket(stack, jsii.String("MyBucket"), &awss3.CfnBucketProps{
 BucketName: jsii.String("MyBucket"),
 CorsConfiguration: &awss3.CfnBucket_CorsConfigurationProperty{
  CorsRules: []awss3.CorsRule{
   awss3.CorsRule{
    AllowedOrigins: jsii.Strings("*"),
    AllowedMethods: &[]awss3.HttpMethods{"GET"},
   },
  },
 },
})
```

Important

Anda tidak dapat menggunakan tipe properti L2 dengan konstruksi L1, atau sebaliknya. Saat bekerja dengan konstruksi L1, selalu gunakan tipe yang ditentukan untuk konstruksi L1 yang Anda gunakan. Jangan gunakan tipe dari konstruksi L1 lainnya (beberapa mungkin memiliki nama yang sama, tetapi tipenya bukan tipe yang sama).
Beberapa referensi API khusus bahasa kami saat ini memiliki kesalahan di jalur ke tipe properti L1, atau tidak mendokumentasikan kelas ini sama sekali. Kami berharap untuk segera memperbaikinya. Sementara itu, ingatlah bahwa tipe seperti itu selalu merupakan kelas dalam dari konstruksi L1 yang digunakan.

Bekerja dengan konstruksi L2

Dalam contoh berikut, kita mendefinisikan bucket Amazon S3 dengan membuat objek dari konstruksi `Bucket`L2:

TypeScript

```
import * as s3 from 'aws-cdk-lib/aws-s3';

// "this" is HelloCdkStack
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true
```

```
});
```

JavaScript

```
const s3 = require('aws-cdk-lib/aws-s3');

// "this" is HelloCdkStack
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true
});
```

Python

```
import aws_cdk.aws_s3 as s3

# "self" is HelloCdkStack
s3.Bucket(self, "MyFirstBucket", versioned=True)
```

Java

```
import software.amazon.awscdk.services.s3.*;

public class HelloCdkStack extends Stack {
    public HelloCdkStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloCdkStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        Bucket.Builder.create(this, "MyFirstBucket")
                .versioned(true).build();
    }
}
```

C#

```
using Amazon.CDK.AWS.S3;

// "this" is HelloCdkStack
new Bucket(this, "MyFirstBucket", new BucketProps
```

```
{
    Versioned = true
});
```

Go

```
import (
 "github.com/aws/aws-cdk-go/awscdk/v2/awss3"
 "github.com/aws/jsii-runtime-go"
)

// stack is HelloCdkStack
awss3.NewBucket(stack, jsii.String("MyFirstBucket"), &awss3.BucketProps{
  Versioned: jsii.Bool(true),
 })>
```

`MyFirstBucket`bukan nama ember yang AWS CloudFormation menciptakan. Ini adalah pengidentifikasi logis yang diberikan ke konstruksi baru dalam konteks aplikasi CDK Anda. Nilai PhysicalName akan digunakan untuk memberi nama sumber daya. AWS CloudFormation

Bekerja dengan konstruksi pihak ketiga

Construct Hub adalah sumber daya untuk membantu Anda menemukan konstruksi tambahan dari AWS, pihak ketiga, dan komunitas CDK sumber terbuka.

Menulis konstruksi Anda sendiri

Selain menggunakan konstruksi yang ada, Anda juga dapat menulis konstruksi Anda sendiri dan membiarkan siapa pun menggunakannya di aplikasi mereka. Semua konstruksi sama dalam. AWS CDK Konstruksi dari AWS Construct Library diperlakukan sama seperti contstruct dari pustaka pihak ketiga yang diterbitkan melaluiNPM,, atau. Maven PyPI Konstruksi yang dipublikasikan ke repositori paket internal perusahaan Anda juga diperlakukan dengan cara yang sama.

Untuk mendeklarasikan konstruksi baru, buat kelas yang memperluas kelas dasar Construct, dalam `constructs` paket, lalu ikuti pola untuk argumen penginisialisasi.

Contoh berikut menunjukkan cara mendeklarasikan konstruksi yang mewakili bucket Amazon S3. Bucket S3 mengirimkan notifikasi Amazon Simple Notification Service (Amazon SNS) setiap kali seseorang mengunggah file ke dalamnya.

TypeScript

```
export interface NotifyingBucketProps {
  prefix?: string;
}

export class NotifyingBucket extends Construct {
  constructor(scope: Construct, id: string, props: NotifyingBucketProps = {}) {
    super(scope, id);
    const bucket = new s3.Bucket(this, 'bucket');
    const topic = new sns.Topic(this, 'topic');
    bucket.addObjectCreatedNotification(new s3notify.SnsDestination(topic),
      { prefix: props.prefix });
  }
}
```

JavaScript

```
class NotifyingBucket extends Construct {
  constructor(scope, id, props = {}) {
    super(scope, id);
    const bucket = new s3.Bucket(this, 'bucket');
    const topic = new sns.Topic(this, 'topic');
    bucket.addObjectCreatedNotification(new s3notify.SnsDestination(topic),
      { prefix: props.prefix });
  }
}

module.exports = { NotifyingBucket }
```

Python

```
class NotifyingBucket(Construct):

    def __init__(self, scope: Construct, id: str, *, prefix=None):
        super().__init__(scope, id)
        bucket = s3.Bucket(self, "bucket")
        topic = sns.Topic(self, "topic")
        bucket.add_object_created_notification(s3notify.SnsDestination(topic),
            s3.NotificationKeyFilter(prefix=prefix))
```

Java

```
public class NotifyingBucket extends Construct {

    public NotifyingBucket(final Construct scope, final String id) {
        this(scope, id, null, null);
    }

    public NotifyingBucket(final Construct scope, final String id, final BucketProps
 props) {
        this(scope, id, props, null);
    }

    public NotifyingBucket(final Construct scope, final String id, final String
 prefix) {
        this(scope, id, null, prefix);
    }

    public NotifyingBucket(final Construct scope, final String id, final BucketProps
 props, final String prefix) {
        super(scope, id);

        Bucket bucket = new Bucket(this, "bucket");
        Topic topic = new Topic(this, "topic");
        if (prefix != null)
            bucket.addObjectCreatedNotification(new SnsDestination(topic),
                NotificationKeyFilter.builder().prefix(prefix).build());
     }
}
```

C#

```
public class NotifyingBucketProps : BucketProps
{
    public string Prefix { get; set; }
}

public class NotifyingBucket : Construct
{
    public NotifyingBucket(Construct scope, string id, NotifyingBucketProps props =
 null) : base(scope, id)
    {
        var bucket = new Bucket(this, "bucket");
```

```
        var topic = new Topic(this, "topic");
        bucket.AddObjectCreatedNotification(new SnsDestination(topic), new
 NotificationKeyFilter
        {
            Prefix = props?.Prefix
        });
    }
}
```

Go

```
type NotifyingBucketProps struct {
 awss3.BucketProps
 Prefix *string
}

func NewNotifyingBucket(scope constructs.Construct, id *string, props
 *NotifyingBucketProps) awss3.Bucket {
 var bucket awss3.Bucket
 if props == nil {
  bucket = awss3.NewBucket(scope, jsii.String(*id+"Bucket"), nil)
 } else {
  bucket = awss3.NewBucket(scope, jsii.String(*id+"Bucket"), &props.BucketProps)
 }
 topic := awssns.NewTopic(scope, jsii.String(*id+"Topic"), nil)
 if props == nil {
  bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic))
 } else {
  bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic),
 &awss3.NotificationKeyFilter{
   Prefix: props.Prefix,
  })
 }
 return bucket
}
```

> **Note**
>
> `NotifyingBucket`Konstruksi kami mewarisi bukan dari `Bucket` melainkan dari. `Construct` Kami menggunakan komposisi, bukan pewarisan, untuk menggabungkan bucket

Amazon S3 dan topik Amazon SNS bersama-sama. Secara umum, komposisi lebih disukai daripada pewarisan ketika mengembangkan AWS CDK konstruksi.

`NotifyingBucket`Konstruktor memiliki tanda tangan konstruksi yang khas:`scope`,`id`, dan. `props` Argumen terakhir,`props`, adalah opsional (mendapat nilai default`{}`) karena semua alat peraga adalah opsional. (`Construct`Kelas dasar tidak mengambil `props` argumen.) Anda dapat menentukan instance konstruksi ini di aplikasi Anda tanpa`props`, misalnya:

TypeScript

```
new NotifyingBucket(this, 'MyNotifyingBucket');
```

JavaScript

```
new NotifyingBucket(this, 'MyNotifyingBucket');
```

Python

```
NotifyingBucket(self, "MyNotifyingBucket")
```

Java

```
new NotifyingBucket(this, "MyNotifyingBucket");
```

C#

```
new NotifyingBucket(this, "MyNotifyingBucket");
```

Go

```
NewNotifyingBucket(stack, jsii.String("MyNotifyingBucket"), nil)
```

Atau Anda dapat menggunakan `props` (di Java, parameter tambahan) untuk menentukan awalan jalur untuk difilter, misalnya:

TypeScript

```
new NotifyingBucket(this, 'MyNotifyingBucket', { prefix: 'images/' });
```

JavaScript

```
new NotifyingBucket(this, 'MyNotifyingBucket', { prefix: 'images/' });
```

Python

```
NotifyingBucket(self, "MyNotifyingBucket", prefix="images/")
```

Java

```
new NotifyingBucket(this, "MyNotifyingBucket", "/images");
```

C#

```
new NotifyingBucket(this, "MyNotifyingBucket", new NotifyingBucketProps
{
    Prefix = "/images"
});
```

Go

```
NewNotifyingBucket(stack, jsii.String("MyNotifyingBucket"), &NotifyingBucketProps{
 Prefix: jsii.String("images/"),
})
```

Biasanya, Anda juga ingin mengekspos beberapa properti atau metode pada konstruksi Anda. Tidak terlalu berguna untuk memiliki topik yang tersembunyi di balik konstruksi Anda, karena pengguna konstruksi Anda tidak dapat berlangganan. Menambahkan `topic` properti memungkinkan konsumen mengakses topik batin, seperti yang ditunjukkan pada contoh berikut:

TypeScript

```
export class NotifyingBucket extends Construct {
  public readonly topic: sns.Topic;
```

```
  constructor(scope: Construct, id: string, props: NotifyingBucketProps) {
    super(scope, id);
    const bucket = new s3.Bucket(this, 'bucket');
    this.topic = new sns.Topic(this, 'topic');
    bucket.addObjectCreatedNotification(new s3notify.SnsDestination(this.topic),
 { prefix: props.prefix });
  }
}
```

JavaScript

```
class NotifyingBucket extends Construct {

  constructor(scope, id, props) {
    super(scope, id);
    const bucket = new s3.Bucket(this, 'bucket');
    this.topic = new sns.Topic(this, 'topic');
    bucket.addObjectCreatedNotification(new s3notify.SnsDestination(this.topic),
 { prefix: props.prefix });
  }
}

module.exports = { NotifyingBucket };
```

Python

```
class NotifyingBucket(Construct):

    def __init__(self, scope: Construct, id: str, *, prefix=None, **kwargs):
        super().__init__(scope, id)
        bucket = s3.Bucket(self, "bucket")
        self.topic = sns.Topic(self, "topic")
        bucket.add_object_created_notification(s3notify.SnsDestination(self.topic),
            s3.NotificationKeyFilter(prefix=prefix))
```

Java

```
public class NotifyingBucket extends Construct {

    public Topic topic = null;

    public NotifyingBucket(final Construct scope, final String id) {
```

```
        this(scope, id, null, null);
    }

    public NotifyingBucket(final Construct scope, final String id, final BucketProps
 props) {
        this(scope, id, props, null);
    }

    public NotifyingBucket(final Construct scope, final String id, final String
 prefix) {
        this(scope, id, null, prefix);
    }

    public NotifyingBucket(final Construct scope, final String id, final BucketProps
 props, final String prefix) {
        super(scope, id);

        Bucket bucket = new Bucket(this, "bucket");
        topic = new Topic(this, "topic");
        if (prefix != null)
            bucket.addObjectCreatedNotification(new SnsDestination(topic),
                NotificationKeyFilter.builder().prefix(prefix).build());
     }
}
```

C#

```
public class NotifyingBucket : Construct
{
    public readonly Topic topic;

    public NotifyingBucket(Construct scope, string id, NotifyingBucketProps props =
 null) : base(scope, id)
    {
        var bucket = new Bucket(this, "bucket");
        topic = new Topic(this, "topic");
        bucket.AddObjectCreatedNotification(new SnsDestination(topic), new
 NotificationKeyFilter
        {
            Prefix = props?.Prefix
        });
    }
}
```

Go

Untuk melakukan ini di Go, kita perlu sedikit pipa tambahan. `NewNotifyingBucket`Fungsi asli kami mengembalikan file`awss3.Bucket`. Kita perlu memperluas `Bucket` untuk menyertakan `topic` anggota dengan membuat `NotifyingBucket` struct. Fungsi kita kemudian akan mengembalikan tipe ini.

```
type NotifyingBucket struct {
 awss3.Bucket
 topic awssns.Topic
}

func NewNotifyingBucket(scope constructs.Construct, id *string, props
 *NotifyingBucketProps) NotifyingBucket {
 var bucket awss3.Bucket
 if props == nil {
  bucket = awss3.NewBucket(scope, jsii.String(*id+"Bucket"), nil)
 } else {
  bucket = awss3.NewBucket(scope, jsii.String(*id+"Bucket"), &props.BucketProps)
 }
 topic := awssns.NewTopic(scope, jsii.String(*id+"Topic"), nil)
 if props == nil {
  bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic))
 } else {
  bucket.AddObjectCreatedNotification(awss3notifications.NewSnsDestination(topic),
 &awss3.NotificationKeyFilter{
   Prefix: props.Prefix,
  })
 }
 var nbucket NotifyingBucket
 nbucket.Bucket = bucket
 nbucket.topic = topic
 return nbucket
}
```

Sekarang, konsumen dapat berlangganan topik, misalnya:

TypeScript

```
const queue = new sqs.Queue(this, 'NewImagesQueue');
const images = new NotifyingBucket(this, '/images');
```

```
images.topic.addSubscription(new sns_sub.SqsSubscription(queue));
```

JavaScript

```
const queue = new sqs.Queue(this, 'NewImagesQueue');
const images = new NotifyingBucket(this, '/images');
images.topic.addSubscription(new sns_sub.SqsSubscription(queue));
```

Python

```
queue = sqs.Queue(self, "NewImagesQueue")
images = NotifyingBucket(self, prefix="Images")
images.topic.add_subscription(sns_sub.SqsSubscription(queue))
```

Java

```
NotifyingBucket images = new NotifyingBucket(this, "MyNotifyingBucket", "/images");
images.topic.addSubscription(new SqsSubscription(queue));
```

C#

```
var queue = new Queue(this, "NewImagesQueue");
var images = new NotifyingBucket(this, "MyNotifyingBucket", new NotifyingBucketProps
{
    Prefix = "/images"
});
images.topic.AddSubscription(new SqsSubscription(queue));
```

Go

```
 queue := awssqs.NewQueue(stack, jsii.String("NewImagesQueue"), nil)
 images := NewNotifyingBucket(stack, jsii.String("MyNotifyingBucket"),
 &NotifyingBucketProps{
  Prefix: jsii.String("/images"),
 })
 images.topic.AddSubscription(awssnssubscriptions.NewSqsSubscription(queue, nil))
```

Pelajari selengkapnya

Video berikut memberikan ikhtisar komprehensif tentang konstruksi CDK, dan menjelaskan bagaimana Anda dapat menggunakannya di aplikasi CDK Anda.

Konstruksi CDK Dijelaskan

Lingkungan

Lingkungan adalah target Akun AWS dan tumpukan Wilayah AWS itu digunakan. Semua tumpukan di aplikasi CDK Anda secara eksplisit atau implisit terkait dengan environment (). `env`

Topik

- Mengonfigurasi lingkungan
- Lingkungan bootstrap

Mengonfigurasi lingkungan

Untuk tumpukan produksi, sebaiknya Anda secara eksplisit menentukan lingkungan untuk setiap tumpukan di aplikasi menggunakan properti. `env` Contoh berikut menentukan lingkungan yang berbeda untuk dua tumpukan yang berbeda.

TypeScript

```
const envEU  = { account: '2383838383', region: 'eu-west-1' };
const envUSA = { account: '8373873873', region: 'us-west-2' };

new MyFirstStack(app, 'first-stack-us', { env: envUSA });
new MyFirstStack(app, 'first-stack-eu', { env: envEU });
```

JavaScript

```
const envEU  = { account: '2383838383', region: 'eu-west-1' };
const envUSA = { account: '8373873873', region: 'us-west-2' };

new MyFirstStack(app, 'first-stack-us', { env: envUSA });
new MyFirstStack(app, 'first-stack-eu', { env: envEU });
```

Python

```
env_EU = cdk.Environment(account="8373873873", region="eu-west-1")
env_USA = cdk.Environment(account="2383838383", region="us-west-2")

MyFirstStack(app, "first-stack-us", env=env_USA)
MyFirstStack(app, "first-stack-eu", env=env_EU)
```

Java

```
public class MyApp {

    // Helper method to build an environment
    static Environment makeEnv(String account, String region) {
        return Environment.builder()
                .account(account)
                .region(region)
                .build();
    }

    public static void main(final String argv[]) {
        App app = new App();

        Environment envEU = makeEnv("8373873873", "eu-west-1");
        Environment envUSA = makeEnv("2383838383", "us-west-2");

        new MyFirstStack(app, "first-stack-us", StackProps.builder()
                .env(envUSA).build());
        new MyFirstStack(app, "first-stack-eu", StackProps.builder()
                .env(envEU).build());

        app.synth();
    }
}
```

C#

```
Amazon.CDK.Environment makeEnv(string account, string region)
{
    return new Amazon.CDK.Environment
    {
        Account = account,
        Region = region
```

```
    };
}

var envEU = makeEnv(account: "8373873873", region: "eu-west-1");
var envUSA = makeEnv(account: "2383838383", region: "us-west-2");

new MyFirstStack(app, "first-stack-us", new StackProps { Env=envUSA });
new MyFirstStack(app, "first-stack-eu", new StackProps { Env=envEU });
```

Saat Anda membuat kode keras akun target dan Wilayah seperti yang ditunjukkan pada contoh sebelumnya, tumpukan selalu diterapkan ke akun dan Wilayah tertentu tersebut. Untuk membuat tumpukan dapat diterapkan ke target yang berbeda, tetapi untuk menentukan target pada waktu sintesis, tumpukan Anda dapat menggunakan dua variabel lingkungan yang disediakan oleh AWS CDK CLI: dan. `CDK_DEFAULT_ACCOUNT` `CDK_DEFAULT_REGION` Variabel ini diatur berdasarkan AWS profil yang ditentukan menggunakan --profile opsi, atau AWS profil default jika Anda tidak menentukannya.

Fragmen kode berikut menunjukkan cara mengakses akun dan Wilayah yang diteruskan dari AWS CDK CLI di tumpukan Anda.

TypeScript

Akses variabel lingkungan melalui `process` objek Node.

> **Note**
>
> Anda memerlukan `DefinitelyTyped` modul untuk `process` digunakan TypeScript. `cdk init`menginstal modul ini untuk Anda. Namun, Anda harus menginstal modul ini secara manual jika Anda bekerja dengan proyek yang dibuat sebelum ditambahkan, atau jika Anda tidak menyiapkan proyek Anda menggunakan`cdk init`.
>
> ```
> npm install @types/node
> ```

```
new MyDevStack(app, 'dev', {
  env: {
    account: process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEFAULT_REGION
```

```
}});
```

JavaScript

Akses variabel lingkungan melalui `process` objek Node.

```
new MyDevStack(app, 'dev', {
  env: {
    account: process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEFAULT_REGION
}});
```

Python

Gunakan `environ` kamus `os` modul untuk mengakses variabel lingkungan.

```
import os
MyDevStack(app, "dev", env=cdk.Environment(
    account=os.environ["CDK_DEFAULT_ACCOUNT"],
    region=os.environ["CDK_DEFAULT_REGION"]))
```

Java

Gunakan `System.getenv()` untuk mendapatkan nilai variabel lingkungan.

```
public class MyApp {

    // Helper method to build an environment
    static Environment makeEnv(String account, String region) {
        account = (account == null) ? System.getenv("CDK_DEFAULT_ACCOUNT") :
 account;
        region = (region == null) ? System.getenv("CDK_DEFAULT_REGION") : region;

        return Environment.builder()
                .account(account)
                .region(region)
                .build();
    }

    public static void main(final String argv[]) {
        App app = new App();
```

```
        Environment envEU = makeEnv(null, null);
        Environment envUSA = makeEnv(null, null);

        new MyDevStack(app, "first-stack-us", StackProps.builder()
                .env(envUSA).build());
        new MyDevStack(app, "first-stack-eu", StackProps.builder()
                .env(envEU).build());

        app.synth();
    }
}
```

C#

Gunakan `System.Environment.GetEnvironmentVariable()` untuk mendapatkan nilai variabel lingkungan.

```
Amazon.CDK.Environment makeEnv(string account=null, string region=null)
{
    return new Amazon.CDK.Environment
    {
        Account = account ??
 System.Environment.GetEnvironmentVariable("CDK_DEFAULT_ACCOUNT"),
        Region = region ??
 System.Environment.GetEnvironmentVariable("CDK_DEFAULT_REGION")
    };
}

new MyDevStack(app, "dev", new StackProps { Env = makeEnv() });
```

Tentukan Wilayah AWS menggunakan kode Wilayah. Untuk daftar, lihat Titik akhir Regional.

Perbedaan AWS CDK antara tidak menentukan `env` properti sama sekali dan menentukannya menggunakan dan. `CDK_DEFAULT_ACCOUNT` `CDK_DEFAULT_REGION` Yang pertama menyiratkan bahwa tumpukan harus mensintesis template agnostik lingkungan. Konstruksi yang didefinisikan dalam tumpukan seperti itu tidak dapat menggunakan informasi apa pun tentang lingkungannya. Misalnya, Anda tidak dapat menulis kode seperti `if (stack.region === 'us-east-1')` atau menggunakan fasilitas kerangka kerja seperti VPC.fromLookup (Python`from_lookup`:), yang perlu menanyakan akun Anda. AWS Fitur-fitur ini tidak berfungsi sama sekali sampai Anda menentukan lingkungan eksplisit; untuk menggunakannya, Anda harus menentukan`env`.

Saat Anda melewati lingkungan Anda menggunakan `CDK_DEFAULT_ACCOUNT` dan`CDK_DEFAULT_REGION`, tumpukan akan digunakan di akun dan Wilayah yang ditentukan oleh AWS CDK CLI pada saat sintesis. Ini memungkinkan kode yang bergantung pada lingkungan bekerja, tetapi juga berarti bahwa template yang disintesis dapat berbeda berdasarkan mesin, pengguna, atau sesi yang disintesis di bawahnya. Perilaku ini sering dapat diterima atau bahkan diinginkan selama pengembangan, tetapi mungkin akan menjadi anti-pola untuk penggunaan produksi.

Anda dapat mengatur `env` sesuka Anda, menggunakan ekspresi yang valid. Misalnya, Anda dapat menulis tumpukan Anda untuk mendukung dua variabel lingkungan tambahan agar Anda dapat mengganti akun dan Wilayah pada waktu sintesis. Kami akan memanggil ini `CDK_DEPLOY_ACCOUNT` dan `CDK_DEPLOY_REGION` di sini, tetapi Anda bisa memberi nama apa pun yang Anda suka, karena mereka tidak diatur oleh AWS CDK. Di lingkungan tumpukan berikut, variabel lingkungan alternatif digunakan jika disetel. Jika tidak disetel, mereka kembali ke lingkungan default yang disediakan oleh AWS CDK.

TypeScript

```
new MyDevStack(app, 'dev', {
  env: {
    account: process.env.CDK_DEPLOY_ACCOUNT || process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEPLOY_REGION || process.env.CDK_DEFAULT_REGION
}});
```

JavaScript

```
new MyDevStack(app, 'dev', {
  env: {
    account: process.env.CDK_DEPLOY_ACCOUNT || process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEPLOY_REGION || process.env.CDK_DEFAULT_REGION
}});
```

Python

```
MyDevStack(app, "dev", env=cdk.Environment(
    account=os.environ.get("CDK_DEPLOY_ACCOUNT", os.environ["CDK_DEFAULT_ACCOUNT"]),
    region=os.environ.get("CDK_DEPLOY_REGION", os.environ["CDK_DEFAULT_REGION"])
```

Java

```
public class MyApp {

    // Helper method to build an environment
    static Environment makeEnv(String account, String region) {
        account = (account == null) ? System.getenv("CDK_DEPLOY_ACCOUNT") : account;
        region = (region == null) ? System.getenv("CDK_DEPLOY_REGION") : region;
        account = (account == null) ? System.getenv("CDK_DEFAULT_ACCOUNT") :
 account;
        region = (region == null) ? System.getenv("CDK_DEFAULT_REGION") : region;

        return Environment.builder()
                .account(account)
                .region(region)
                .build();
    }

    public static void main(final String argv[]) {
        App app = new App();

        Environment envEU = makeEnv(null, null);
        Environment envUSA = makeEnv(null, null);

        new MyDevStack(app, "first-stack-us", StackProps.builder()
                .env(envUSA).build());
        new MyDevStack(app, "first-stack-eu", StackProps.builder()
                .env(envEU).build());

        app.synth();
    }
}
```

C#

```
Amazon.CDK.Environment makeEnv(string account=null, string region=null)
{
    return new Amazon.CDK.Environment
    {
        Account = account ??
            System.Environment.GetEnvironmentVariable("CDK_DEPLOY_ACCOUNT") ??
            System.Environment.GetEnvironmentVariable("CDK_DEFAULT_ACCOUNT"),
        Region = region ??
```

```
            System.Environment.GetEnvironmentVariable("CDK_DEPLOY_REGION") ??
            System.Environment.GetEnvironmentVariable("CDK_DEFAULT_REGION")
    };
}

new MyDevStack(app, "dev", new StackProps { Env = makeEnv() });
```

Dengan lingkungan tumpukan Anda dideklarasikan dengan cara ini, Anda dapat menulis skrip pendek atau file batch seperti berikut untuk mengatur variabel dari argumen baris perintah, lalu panggil`cdk deploy`. Argumen apa pun di luar dua yang pertama diteruskan ke `cdk deploy` dan dapat digunakan untuk menentukan opsi baris perintah atau tumpukan.

macOS/Linux

```
#!/usr/bin/env bash
if [[ $# -ge 2 ]]; then
    export CDK_DEPLOY_ACCOUNT=$1
    export CDK_DEPLOY_REGION=$2
    shift; shift
    npx cdk deploy "$@"
    exit $?
else
    echo 1>&2 "Provide account and region as first two args."
    echo 1>&2 "Additional args are passed through to cdk deploy."
    exit 1
fi
```

Simpan skrip sebagai`cdk-deploy-to.sh`, lalu jalankan `chmod +x cdk-deploy-to.sh` untuk membuatnya dapat dieksekusi.

Windows

```
@findstr /B /V @ %~dpnx0 > %~dpn0.ps1 && powershell -ExecutionPolicy Bypass
 %~dpn0.ps1 %*
@exit /B %ERRORLEVEL%
if ($args.length -ge 2) {
    $env:CDK_DEPLOY_ACCOUNT, $args = $args
    $env:CDK_DEPLOY_REGION,  $args = $args
    npx cdk deploy $args
    exit $lastExitCode
} else {
    [console]::error.writeline("Provide account and region as first two args.")
```

```
    [console]::error.writeline("Additional args are passed through to cdk deploy.")
    exit 1
}
```

Versi Windows skrip digunakan PowerShell untuk menyediakan fungsionalitas yang sama dengan versi macOS/Linux. Ini juga berisi instruksi untuk memungkinkannya dijalankan sebagai file batch sehingga dapat dengan mudah dipanggil dari baris perintah. Itu harus disimpan sebagai`cdk-deploy-to.bat`. File `cdk-deploy-to.ps1` akan dibuat ketika file batch dipanggil.

Kemudian Anda dapat menulis skrip tambahan yang memanggil skrip “deploy-to” untuk diterapkan ke lingkungan tertentu (bahkan beberapa lingkungan per skrip):

macOS/Linux

```
#!/usr/bin/env bash
# cdk-deploy-to-test.sh
./cdk-deploy-to.sh 123457689 us-east-1 "$@"
```

Windows

```
@echo off
rem cdk-deploy-to-test.bat
cdk-deploy-to 135792469 us-east-1 %*
```

Saat menerapkan ke beberapa lingkungan, pertimbangkan apakah Anda ingin melanjutkan penerapan ke lingkungan lain setelah penerapan gagal. Contoh berikut menghindari penerapan ke lingkungan produksi kedua jika yang pertama tidak berhasil.

macOS/Linux

```
#!/usr/bin/env bash
# cdk-deploy-to-prod.sh
./cdk-deploy-to.sh 135792468 us-west-1 "$@" || exit
./cdk-deploy-to.sh 246813579 eu-west-1 "$@"
```

Windows

```
@echo off
rem cdk-deploy-to-prod.bat
```

```
cdk-deploy-to 135792469 us-west-1 %* || exit /B
cdk-deploy-to 245813579 eu-west-1 %*
```

Pengembang masih dapat menggunakan `cdk deploy` perintah normal untuk menyebarkan ke AWS lingkungan mereka sendiri untuk pengembangan.

Jika Anda tidak menentukan lingkungan saat Anda membuat instance tumpukan, tumpukan dikatakan agnostik lingkungan. AWS CloudFormation template yang disintesis dari tumpukan tersebut akan mencoba menggunakan resolusi waktu penerapan pada atribut terkait lingkungan seperti,`stack.account`, `stack.region` dan (Python:). `stack.availabilityZones` `availability_zones`

Saat menggunakan cdk deploy untuk menyebarkan tumpukan agnostik lingkungan, AWS CDK CLI akan menggunakan profil yang ditentukan AWS CLI untuk menentukan di mana harus menggunakan. Jika tidak ada profil yang ditentukan, profil default digunakan. AWS CDK CLIBerikut ini protokol yang mirip dengan AWS CLI untuk menentukan AWS kredensil mana yang akan digunakan saat melakukan operasi di akun Anda AWS . Lihat the section called “AWS CDK Toolkit” untuk detail.

Dalam tumpukan agnostik lingkungan, konstruksi apa pun yang menggunakan Availability Zones akan melihat dua Availability Zone, memungkinkan tumpukan untuk diterapkan ke Wilayah mana pun.

Lingkungan bootstrap

Anda harus mem-bootstrap setiap lingkungan tempat Anda akan menyebarkan tumpukan CDK. Bootstrapping mempersiapkan lingkungan untuk penyebaran. Untuk mempelajari informasi lebih lanjut, lihat Bootstrapping.

Bootstrapping

Bootstrapping adalah proses mempersiapkan lingkungan untuk penyebaran. Bootstrapping adalah tindakan satu kali yang harus Anda lakukan untuk setiap lingkungan tempat Anda menyebarkan sumber daya.

Topik

- Perbedaan template bootstrap
- Stack synthesizer
- Menyesuaikan sintesis
- Kontrak template bootstrap
- Temuan Security Hub

Lingkungan bootstrap

> **Important**
>
> Anda mungkin dikenakan AWS biaya untuk data yang disimpan dalam sumber daya bootstrap.

Bootstrapping menyediakan sumber daya di lingkungan Anda seperti bucket Amazon Simple Storage Service (Amazon S3) untuk menyimpan file AWS Identity and Access Management dan peran (IAM) yang memberikan izin yang diperlukan untuk melakukan penerapan. Sumber daya ini disediakan dalam AWS CloudFormation tumpukan, yang disebut tumpukan bootstrap. Biasanya dinamai`CDKToolkit`. Seperti AWS CloudFormation tumpukan apa pun, itu akan muncul di AWS CloudFormation konsol lingkungan Anda setelah digunakan.

> **Note**
>
> CDK v2 menggunakan template bootstrap modern. Template lama dari CDK v1 tidak didukung di v2.

Lingkungan bersifat independen. Jika Anda ingin menerapkan ke beberapa lingkungan, setiap lingkungan harus di-bootstrap secara terpisah.

Jika Anda mencoba menerapkan aplikasi CDK ke lingkungan yang belum di-bootstrap, Anda akan menerima pesan kesalahan yang mengingatkan Anda untuk mem-bootstrap lingkungan.

Bootstrapping dengan CDK Pipelines

Jika Anda menggunakan CDK Pipelines untuk menyebarkan ke lingkungan akun lain, dan Anda menerima pesan seperti berikut:

```
Policy contains a statement with one or more invalid principals
```

Pesan kesalahan ini berarti bahwa peran IAM yang sesuai tidak ada di lingkungan lain. Penyebab yang paling mungkin adalah bahwa lingkungan belum di-bootstrap. Bootstrap lingkungan dan coba lagi.

> **Note**
>
> Jika lingkungan di-bootstrap, jangan hapus dan buat ulang tumpukan bootstrap lingkungan. Menghapus tumpukan bootstrap akan menghapus AWS sumber daya yang awalnya disediakan di lingkungan untuk mendukung penerapan CDK. Ini akan menyebabkan pipa berhenti bekerja. Sebagai gantinya, coba perbarui tumpukan bootstrap ke versi baru dengan menjalankan CLI `cdk bootstrap` perintah CDK lagi.

Cara bootstrap

Saat Anda mem-bootstrap lingkungan, AWS CloudFormation template diterapkan ke lingkungan tertentu. Template ini menyediakan sumber daya di akun Anda untuk mempersiapkan lingkungan Anda untuk penerapan.

Template bootstrap menerima parameter yang menyesuaikan beberapa aspek sumber daya bootstrap. Untuk informasi selengkapnya, lihat the section called “Menyesuaikan bootstrap”.

Anda dapat bootstrap dengan salah satu cara berikut:

- Gunakan AWS CDK CLI cdk bootstrap perintah. Ini adalah metode paling sederhana dan berfungsi dengan baik jika Anda hanya memiliki beberapa lingkungan untuk bootstrap.
- Menyebarkan template yang disediakan oleh AWS CDK CLI menggunakan alat AWS CloudFormation penyebaran lain. Ini memungkinkan Anda menggunakan AWS CloudFormation StackSets atau AWS Control Tower dan juga AWS CloudFormation konsol atau AWS CLI. Anda dapat membuat modifikasi kecil pada template sebelum penerapan. Pendekatan ini lebih fleksibel dan cocok untuk penyebaran skala besar.

Bukan kesalahan untuk mem-bootstrap lingkungan lebih dari sekali. Jika lingkungan yang Anda bootstrap telah di-bootstrap, tumpukan bootstrap akan ditingkatkan jika perlu. Kalau tidak, tidak akan terjadi apa-apa.

Bootstrapping dengan AWS CDKCLI

Gunakan `cdk bootstrap` perintah untuk bootstrap satu atau lebih AWS lingkungan.

Contoh berikut bootstraps dua lingkungan:

```
$ cdk bootstrap aws://ACCOUNT-NUMBER-1/REGION-1 aws://ACCOUNT-NUMBER-2/REGION-2 ...
```

Contoh berikut menunjukkan beberapa cara lingkungan bootstrap. Seperti yang ditunjukkan pada contoh kedua, `aws://` awalan adalah opsional saat menentukan lingkungan.

```
$ cdk bootstrap aws://123456789012/us-east-1
$ cdk bootstrap 123456789012/us-east-1 123456789012/us-west-1
```

Saat Anda menjalankancdk bootstrap, CDK CLI selalu mensintesis aplikasi CDK di direktori saat ini. Jika Anda tidak menentukan setidaknya satu lingkungan, CDK CLI akan mem-bootstrap semua lingkungan yang direferensikan dalam aplikasi.

Untuk tumpukan agnostik lingkungan, CDK CLI akan mencoba menentukan lingkungan dari sumber default. Ini bisa berupa lingkungan yang ditentukan menggunakan --profile opsi, dari variabel lingkungan, atau AWS CLI sumber default. Jika ditemukan, lingkungan kemudian di-bootstrap.

Misalnya, perintah berikut mensintesis AWS CDK aplikasi saat ini menggunakan `prod` AWS profil, lalu bootstrap lingkungannya.

```
$ cdk bootstrap --profile prod
```

Bootstrapping dari template AWS CloudFormation

Anda dapat mem-bootstrap lingkungan dengan mendapatkan dan menerapkan AWS CloudFormation template bootstrap.

Untuk mendapatkan salinan template ini dalam file`bootstrap-template.yaml`, jalankan perintah berikut:

macOS/Linux

```
$ cdk bootstrap --show-template > bootstrap-template.yaml
```

Windows

Pada Windows, PowerShell harus digunakan untuk melestarikan pengkodean template.

```
powershell "cdk bootstrap --show-template | Out-File -encoding utf8 bootstrap-
template.yaml"
```

Template juga tersedia di AWS CDK GitHub repositori.

Terapkan template ini menggunakan CDK CLI atau mekanisme penerapan pilihan Anda untuk template. AWS CloudFormation Untuk menerapkan menggunakan CDK CLI, jalankan. cdk bootstrap --template ***TEMPLATE_FILENAME*** Anda juga dapat menerapkannya menggunakan AWS CLI dengan menjalankan perintah di bawah ini, atau menyebarkan ke satu atau beberapa akun sekaligus menggunakan AWS CloudFormation Stack Sets.

macOS/Linux

```
aws cloudformation create-stack \
  --stack-name CDKToolkit \
  --template-body file://path/to/bootstrap-template.yaml \
  --capabilities CAPABILITY_NAMED_IAM \
  --region us-west-1
```

Windows

```
aws cloudformation create-stack ^
  --stack-name CDKToolkit ^
  --template-body file://path/to/bootstrap-template.yaml ^
  --capabilities CAPABILITY_NAMED_IAM ^
  --region us-west-1
```

Menyesuaikan bootstrap

Ada dua cara untuk menyesuaikan bootstrap sumber daya di lingkungan Anda:

- Gunakan parameter baris perintah dengan `cdk bootstrap` perintah. Ini memungkinkan Anda memodifikasi beberapa aspek template.
- Ubah template bootstrap default dan terapkan sendiri. Ini memberi Anda kontrol yang lebih lengkap atas sumber daya bootstrap.

Opsi baris perintah berikut, bila digunakan dengan CDK CLIcdk bootstrap, memberikan penyesuaian yang umum digunakan pada template bootstrap:

- - -bootstrap-bucket-name mengganti nama bucket Amazon S3. Mungkin memerlukan perubahan pada aplikasi CDK Anda (lihatthe section called “Stack synthesizer”).
- --bootstrap-kms-key-idmengganti AWS KMS kunci yang digunakan untuk mengenkripsi bucket S3.
- --cloudformation-execution-policiesmenentukan ARN kebijakan terkelola yang harus dilampirkan ke peran penerapan yang diasumsikan AWS CloudFormation selama penerapan tumpukan Anda. Secara default, tumpukan disebarkan dengan izin administrator penuh menggunakan kebijakan. `AdministratorAccess`

  ARN kebijakan harus diteruskan sebagai argumen string tunggal, dengan masing-masing ARN dipisahkan dengan koma. Sebagai contoh:

  ```
  --cloudformation-execution-policies "arn:aws:iam::aws:policy/
  AWSLambda_FullAccess,arn:aws:iam::aws:policy/AWSCodeDeployFullAccess".
  ```

  > **Important**
  >
  > Untuk menghindari kegagalan penerapan, pastikan kebijakan yang Anda tentukan cukup untuk penerapan apa pun yang akan Anda lakukan di lingkungan yang di-bootstrap.

- --qualifieradalah string yang ditambahkan ke nama semua sumber daya di tumpukan bootstrap. Kualifikasi memungkinkan Anda menghindari bentrokan nama sumber daya saat Anda menyediakan beberapa tumpukan bootstrap di lingkungan yang sama. Defaultnya adalah `hnb659fds` (nilai ini tidak memiliki signifikansi).

  Mengubah qualifier juga mengharuskan aplikasi CDK Anda meneruskan nilai yang diubah ke stack synthesizer. Untuk informasi selengkapnya, lihat the section called “Stack synthesizer”.
- --tagsmenambahkan satu atau lebih AWS CloudFormation tag ke tumpukan bootstrap.
- --trustmencantumkan AWS akun yang dapat diterapkan ke lingkungan yang sedang di-bootstrap.

  Gunakan tanda ini saat mem-bootstrap lingkungan tempat Pipeline CDK di lingkungan lain akan disebarkan. Akun yang melakukan bootstrapping selalu dipercaya.
- --trust-for-lookupdaftar AWS akun yang mungkin mencari informasi konteks dari lingkungan yang di-bootstrap.

Gunakan tanda ini untuk memberikan izin akun untuk mensintesis tumpukan yang akan diterapkan ke lingkungan, tanpa benar-benar memberi mereka izin untuk menerapkan tumpukan tersebut secara langsung.

- --termination-protectionmencegah tumpukan bootstrap dihapus. Untuk informasi selengkapnya, lihat Melindungi tumpukan agar tidak dihapus di Panduan AWS CloudFormation Pengguna.

> **Important**
>
> Template bootstrap modern secara efektif memberikan izin yang tersirat oleh `--cloudformation-execution-policies` ke AWS akun mana pun dalam daftar. `--trust` Secara default, ini memperluas izin untuk membaca dan menulis ke sumber daya apa pun di akun bootstrap. Pastikan untuk mengonfigurasi tumpukan bootstrap dengan kebijakan dan akun tepercaya yang nyaman bagi Anda.

Menyesuaikan template

Ketika Anda membutuhkan lebih banyak penyesuaian daripada yang CLI dapat disediakan CDK, Anda dapat memodifikasi template bootstrap agar sesuai dengan kebutuhan Anda. Pertama, Anda mendapatkan template menggunakan --show-template opsi. Berikut ini adalah contohnya:

```
$ cdk bootstrap --show-template
```

Setiap modifikasi yang Anda buat harus mematuhi kontrak template bootstrap. Untuk memastikan bahwa kustomisasi Anda tidak sengaja ditimpa nantinya oleh seseorang yang menjalankan cdk bootstrap menggunakan templat default, ubah nilai default parameter template. `BootstrapVariant` CDK CLI hanya akan mengizinkan penimpaan tumpukan bootstrap dengan template yang memiliki versi yang `BootstrapVariant` sama dan sama atau lebih tinggi dari template yang saat ini digunakan.

Anda kemudian dapat menerapkan template yang dimodifikasi seperti yang dijelaskan dalamthe section called “Bootstrapping dari template AWS CloudFormation”, atau menggunakancdk bootstrap --template.

```
$ cdk bootstrap --template bootstrap-template.yaml
```

Perbedaan template bootstrap

Seperti disebutkan sebelumnya, AWS CDK v1 mendukung dua template bootstrap, warisan dan modern. CDK v2 hanya mendukung template modern. Sebagai referensi, berikut adalah perbedaan tingkat tinggi antara kedua templat ini.

Fitur	Warisan (hanya v1)	Modern (v1 dan v2)
Penerapan lintas akun	Tidak diizinkan	Diizinkan
AWS CloudFormation Izin	Menyebarkan menggunakan izin pengguna saat ini (ditentukan oleh AWS profil, variabel lingkungan, dll.)	Menerapkan menggunakan izin yang ditentukan saat tumpukan bootstrap disediakan (misalnya, dengan menggunakan) `--trust`
Pembuatan Versi	Hanya satu versi stack bootstrap yang tersedia	Bootstrap stack berversi; sumber daya baru dapat ditambahkan di versi future, dan AWS CDK aplikasi dapat memerlukan versi minimum
Sumber daya *	Bucket Amazon S3	Bucket Amazon S3
		AWS KMS key
		Peran IAM
		Repositori Amazon ECR
		Parameter SSM untuk pembuatan versi
Penamaan sumber daya	Dihasilkan secara otomatis	Deterministik
Enkripsi ember	Kunci default	Kunci yang dikelola pelanggan

* Kami akan menambahkan sumber daya tambahan ke template bootstrap sesuai kebutuhan.

Lingkungan yang di-bootstrap menggunakan template lama harus ditingkatkan untuk menggunakan template modern untuk CDK v2 dengan melakukan bootstrapping ulang. Terapkan ulang semua AWS CDK aplikasi di lingkungan setidaknya sekali sebelum menghapus bucket lama.

Stack synthesizer

AWS CDK Aplikasi Anda perlu mengetahui tentang resource bootstrap yang tersedia agar berhasil mensintesis tumpukan yang dapat digunakan. Stack synthesizer adalah AWS CDK kelas yang mengontrol bagaimana template stack disintesis. Ini termasuk bagaimana ia menggunakan sumber daya bootstrap (misalnya, bagaimana mengacu pada aset yang disimpan dalam bucket bootstrap).

AWS CDK Synthesizer tumpukan bawaan disebut. `DefaultStackSynthesizer` Ini mencakup kemampuan untuk penyebaran lintas akun dan penerapan CDK Pipelines.

Anda dapat meneruskan synthesizer tumpukan ke tumpukan saat Anda membuat instance menggunakan properti. `synthesizer`

TypeScript

```
new MyStack(this, 'MyStack', {
  // stack properties
  synthesizer: new DefaultStackSynthesizer({
    // synthesizer properties
  }),
});
```

JavaScript

```
new MyStack(this, 'MyStack', {
  // stack properties
  synthesizer: new DefaultStackSynthesizer({
    // synthesizer properties
  }),
});
```

Python

```
MyStack(self, "MyStack",
    # stack properties
    synthesizer=DefaultStackSynthesizer(
        # synthesizer properties
```

```
))
```

Java

```
new MyStack(app, "MyStack", StackProps.builder()
    // stack properties
  .synthesizer(DefaultStackSynthesizer.Builder.create()
    // synthesizer properties
    .build())
  .build();
```

C#

```
new MyStack(app, "MyStack", new StackProps
// stack properties
{
    Synthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps
    {
        // synthesizer properties
    })
});
```

Jika Anda tidak menyediakan `synthesizer` properti, `DefaultStackSynthesizer` digunakan.

Menyesuaikan sintesis

Bergantung pada perubahan yang Anda buat pada template bootstrap, Anda mungkin juga perlu menyesuaikan sintesis. `DefaultStackSynthesizer`Dapat disesuaikan menggunakan properti yang dijelaskan sebagai berikut.

Jika tidak satu pun dari properti ini menyediakan penyesuaian yang Anda butuhkan, Anda dapat menulis synthesizer Anda sebagai kelas yang mengimplementasikan `IStackSynthesizer` (mungkin berasal dari). `DefaultStackSynthesizer`

Mengubah kualifikasi

Kualifikasi ditambahkan ke nama sumber daya bootstrap untuk membedakan sumber daya dalam tumpukan bootstrap terpisah. Untuk menerapkan dua versi tumpukan bootstrap yang berbeda di lingkungan yang sama (AWS akun dan Wilayah), tumpukan harus memiliki kualifikasi yang berbeda.

Fitur ini ditujukan untuk isolasi nama antara pengujian otomatis CDK itu sendiri. Kecuali Anda dapat dengan tepat mencakup izin IAM yang diberikan ke peran AWS CloudFormation eksekusi, tidak ada manfaat isolasi izin untuk memiliki dua tumpukan bootstrap yang berbeda dalam satu akun. Oleh karena itu, biasanya tidak perlu mengubah nilai ini.

Untuk mengubah qualifier, konfigurasikan `DefaultStackSynthesizer` keduanya dengan membuat instance synthesizer dengan properti:

TypeScript

```
new MyStack(this, 'MyStack', {
  synthesizer: new DefaultStackSynthesizer({
    qualifier: 'MYQUALIFIER',
  }),
});
```

JavaScript

```
new MyStack(this, 'MyStack', {
  synthesizer: new DefaultStackSynthesizer({
    qualifier: 'MYQUALIFIER',
  }),
})
```

Python

```
MyStack(self, "MyStack",
    synthesizer=DefaultStackSynthesizer(
        qualifier="MYQUALIFIER"
))
```

Java

```
new MyStack(app, "MyStack", StackProps.builder()
  .synthesizer(DefaultStackSynthesizer.Builder.create()
    .qualifier("MYQUALIFIER")
    .build())
  .build();
```

C#

```
new MyStack(app, "MyStack", new StackProps
{
    Synthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps
    {
        Qualifier = "MYQUALIFIER"
    })
});
```

Atau dengan mengonfigurasi qualifier sebagai kunci konteks di. `cdk.json`

```
{
  "app": "...",
  "context": {
    "@aws-cdk/core:bootstrapQualifier": "MYQUALIFIER"
  }
}
```

Mengubah nama sumber daya

Semua `DefaultStackSynthesizer` properti lainnya berhubungan dengan nama-nama sumber daya dalam template bootstrap. Anda hanya perlu menyediakan salah satu properti ini jika Anda memodifikasi template bootstrap dan mengubah nama sumber daya atau skema penamaan.

Semua properti menerima placeholder khusus`${Qualifier}`,, `${AWS::Partition}${AWS::AccountId}`, dan. `${AWS::Region}` Placeholder ini diganti dengan nilai `qualifier` parameter dan AWS partisi, ID akun, dan nilai Region untuk lingkungan tumpukan, masing-masing.

Contoh berikut menunjukkan properti yang paling umum digunakan `DefaultStackSynthesizer` bersama dengan nilai defaultnya, seolah-olah Anda membuat instance synthesizer. Untuk daftar lengkap, lihat DefaultStackSynthesizerProps.

TypeScript

```
new DefaultStackSynthesizer({
  // Name of the S3 bucket for file assets
  fileAssetsBucketName: 'cdk-${Qualifier}-assets-${AWS::AccountId}-${AWS::Region}',
  bucketPrefix: '',
```

```
  // Name of the ECR repository for Docker image assets
  imageAssetsRepositoryName: 'cdk-${Qualifier}-container-assets-${AWS::AccountId}-
${AWS::Region}',

  // ARN of the role assumed by the CLI and Pipeline to deploy here
  deployRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-deploy-role-${AWS::AccountId}-${AWS::Region}',
  deployRoleExternalId: '',

  // ARN of the role used for file asset publishing (assumed from the CLI role)
  fileAssetPublishingRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-file-publishing-role-${AWS::AccountId}-${AWS::Region}',
  fileAssetPublishingExternalId: '',

  // ARN of the role used for Docker asset publishing (assumed from the CLI role)
  imageAssetPublishingRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-image-publishing-role-${AWS::AccountId}-${AWS::Region}',
  imageAssetPublishingExternalId: '',

  // ARN of the role passed to CloudFormation to execute the deployments
  cloudFormationExecutionRole: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-cfn-exec-role-${AWS::AccountId}-${AWS::Region}',

  // ARN of the role used to look up context information in an environment
  lookupRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-lookup-role-${AWS::AccountId}-${AWS::Region}',
  lookupRoleExternalId: '',

  // Name of the SSM parameter which describes the bootstrap stack version number
  bootstrapStackVersionSsmParameter: '/cdk-bootstrap/${Qualifier}/version',

  // Add a rule to every template which verifies the required bootstrap stack
 version
  generateBootstrapVersionRule: true,

})
```

JavaScript

```
new DefaultStackSynthesizer({
  // Name of the S3 bucket for file assets
  fileAssetsBucketName: 'cdk-${Qualifier}-assets-${AWS::AccountId}-${AWS::Region}',
```

```
  bucketPrefix: '',

  // Name of the ECR repository for Docker image assets
  imageAssetsRepositoryName: 'cdk-${Qualifier}-container-assets-${AWS::AccountId}-
${AWS::Region}',

  // ARN of the role assumed by the CLI and Pipeline to deploy here
  deployRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-deploy-role-${AWS::AccountId}-${AWS::Region}',
  deployRoleExternalId: '',

  // ARN of the role used for file asset publishing (assumed from the CLI role)
  fileAssetPublishingRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-file-publishing-role-${AWS::AccountId}-${AWS::Region}',
  fileAssetPublishingExternalId: '',

  // ARN of the role used for Docker asset publishing (assumed from the CLI role)
  imageAssetPublishingRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-image-publishing-role-${AWS::AccountId}-${AWS::Region}',
  imageAssetPublishingExternalId: '',

  // ARN of the role passed to CloudFormation to execute the deployments
  cloudFormationExecutionRole: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-cfn-exec-role-${AWS::AccountId}-${AWS::Region}',

  // ARN of the role used to look up context information in an environment
  lookupRoleArn: 'arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-lookup-role-${AWS::AccountId}-${AWS::Region}',
  lookupRoleExternalId: '',

  // Name of the SSM parameter which describes the bootstrap stack version number
  bootstrapStackVersionSsmParameter: '/cdk-bootstrap/${Qualifier}/version',

  // Add a rule to every template which verifies the required bootstrap stack
 version
  generateBootstrapVersionRule: true,
})
```

Python

```
DefaultStackSynthesizer(
  # Name of the S3 bucket for file assets
```

```
  file_assets_bucket_name="cdk-${Qualifier}-assets-${AWS::AccountId}-
${AWS::Region}",
  bucket_prefix="",

  # Name of the ECR repository for Docker image assets
  image_assets_repository_name="cdk-${Qualifier}-container-assets-${AWS::AccountId}-
${AWS::Region}",

  # ARN of the role assumed by the CLI and Pipeline to deploy here
  deploy_role_arn="arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-deploy-role-${AWS::AccountId}-${AWS::Region}",
  deploy_role_external_id="",

  # ARN of the role used for file asset publishing (assumed from the CLI role)
  file_asset_publishing_role_arn="arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-file-publishing-role-${AWS::AccountId}-${AWS::Region}",
  file_asset_publishing_external_id="",

  # ARN of the role used for Docker asset publishing (assumed from the CLI role)
  image_asset_publishing_role_arn="arn:${AWS::Partition}:iam::
${AWS::AccountId}:role/cdk-${Qualifier}-image-publishing-role-${AWS::AccountId}-
${AWS::Region}",
  image_asset_publishing_external_id="",

  # ARN of the role passed to CloudFormation to execute the deployments
  cloud_formation_execution_role="arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-cfn-exec-role-${AWS::AccountId}-${AWS::Region}",

  # ARN of the role used to look up context information in an environment
  lookup_role_arn="arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-lookup-role-${AWS::AccountId}-${AWS::Region}",
  lookup_role_external_id="",

  # Name of the SSM parameter which describes the bootstrap stack version number
  bootstrap_stack_version_ssm_parameter="/cdk-bootstrap/${Qualifier}/version",

  # Add a rule to every template which verifies the required bootstrap stack version
  generate_bootstrap_version_rule=True,
)
```

Java

```
DefaultStackSynthesizer.Builder.create()
```

```
    // Name of the S3 bucket for file assets
    .fileAssetsBucketName("cdk-${Qualifier}-assets-${AWS::AccountId}-
${AWS::Region}")
    .bucketPrefix('')

    // Name of the ECR repository for Docker image assets
    .imageAssetsRepositoryName("cdk-${Qualifier}-container-assets-${AWS::AccountId}-
${AWS::Region}")

    // ARN of the role assumed by the CLI and Pipeline to deploy here
    .deployRoleArn("arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-deploy-role-${AWS::AccountId}-${AWS::Region}")
    .deployRoleExternalId("")

    // ARN of the role used for file asset publishing (assumed from the CLI role)
    .fileAssetPublishingRoleArn("arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-file-publishing-role-${AWS::AccountId}-${AWS::Region}")
    .fileAssetPublishingExternalId("")

    // ARN of the role used for Docker asset publishing (assumed from the CLI role)
    .imageAssetPublishingRoleArn("arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-image-publishing-role-${AWS::AccountId}-${AWS::Region}")
    .imageAssetPublishingExternalId("")

    // ARN of the role passed to CloudFormation to execute the deployments
    .cloudFormationExecutionRole("arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-cfn-exec-role-${AWS::AccountId}-${AWS::Region}")

    .lookupRoleArn("arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-lookup-role-${AWS::AccountId}-${AWS::Region}")
    .lookupRoleExternalId("")

    // Name of the SSM parameter which describes the bootstrap stack version number
    .bootstrapStackVersionSsmParameter("/cdk-bootstrap/${Qualifier}/version")

    // Add a rule to every template which verifies the required bootstrap stack
 version
    .generateBootstrapVersionRule(true)
.build()
```

C#

```
new DefaultStackSynthesizer(new DefaultStackSynthesizerProps
```

```
{
    // Name of the S3 bucket for file assets
    FileAssetsBucketName = "cdk-${Qualifier}-assets-${AWS::AccountId}-
${AWS::Region}",
    BucketPrefix = "",

    // Name of the ECR repository for Docker image assets
    ImageAssetsRepositoryName = "cdk-${Qualifier}-container-assets-
${AWS::AccountId}-${AWS::Region}",

    // ARN of the role assumed by the CLI and Pipeline to deploy here
    DeployRoleArn = "arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-deploy-role-${AWS::AccountId}-${AWS::Region}",
    DeployRoleExternalId = "",

    // ARN of the role used for file asset publishing (assumed from the CLI role)
    FileAssetPublishingRoleArn = "arn:${AWS::Partition}:iam::${AWS::AccountId}:role/
cdk-${Qualifier}-file-publishing-role-${AWS::AccountId}-${AWS::Region}",
    FileAssetPublishingExternalId = "",

    // ARN of the role used for Docker asset publishing (assumed from the CLI role)
    ImageAssetPublishingRoleArn = "arn:${AWS::Partition}:iam::
${AWS::AccountId}:role/cdk-${Qualifier}-image-publishing-role-${AWS::AccountId}-
${AWS::Region}",
    ImageAssetPublishingExternalId = "",

    // ARN of the role passed to CloudFormation to execute the deployments
    CloudFormationExecutionRole = "arn:${AWS::Partition}:iam::
${AWS::AccountId}:role/cdk-${Qualifier}-cfn-exec-role-${AWS::AccountId}-
${AWS::Region}",

    LookupRoleArn = "arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-
${Qualifier}-lookup-role-${AWS::AccountId}-${AWS::Region}",
    LookupRoleExternalId = "",

    // Name of the SSM parameter which describes the bootstrap stack version number
    BootstrapStackVersionSsmParameter = "/cdk-bootstrap/${Qualifier}/version",

    // Add a rule to every template which verifies the required bootstrap stack
 version
    GenerateBootstrapVersionRule = true,
})
```

Kontrak template bootstrap

Persyaratan tumpukan bootstrapping bergantung pada stack synthesizer yang digunakan. Jika Anda menulis stack synthesizer Anda sendiri, Anda memiliki kontrol penuh atas sumber daya bootstrap yang dibutuhkan synthesizer Anda dan bagaimana synthesizer menemukannya.

Bagian ini menjelaskan ekspektasi yang `DefaultStackSynthesizer` dimiliki template bootstrap.

Versioning

Template harus berisi sumber daya untuk membuat parameter SSM dengan nama terkenal dan output untuk mencerminkan versi template.

```
Resources:
  CdkBootstrapVersion:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Name:
        Fn::Sub: '/cdk-bootstrap/${Qualifier}/version'
      Value: 4
Outputs:
  BootstrapVersion:
    Value:
      Fn::GetAtt: [CdkBootstrapVersion, Value]
```

Peran

`DefaultStackSynthesizer`Dibutuhkan lima peran IAM untuk lima tujuan berbeda. Jika Anda tidak menggunakan peran default, Anda harus memberi tahu synthesizer ARN untuk peran yang ingin Anda gunakan.

Perannya adalah sebagai berikut:

- Peran penyebaran diasumsikan oleh AWS CDK Toolkit dan oleh AWS CodePipeline untuk menyebarkan ke lingkungan. `AssumeRolePolicy`Kontrolnya yang dapat menyebarkan ke lingkungan. Di template, Anda dapat melihat izin yang dibutuhkan peran ini.
- Peran pencarian diasumsikan oleh AWS CDK Toolkit untuk melakukan pencarian konteks di lingkungan. `AssumeRolePolicy`Kontrolnya yang dapat menyebarkan ke lingkungan. Izin yang dibutuhkan peran ini dapat dilihat di template.

- Peran penerbitan file dan peran penerbitan gambar diasumsikan oleh AWS CDK Toolkit dan oleh AWS CodeBuild proyek untuk mempublikasikan aset ke dalam lingkungan. Mereka digunakan untuk menulis ke bucket S3 dan repositori ECR, masing-masing. Peran ini memerlukan akses tulis ke sumber daya ini.
- Peran AWS CloudFormation eksekusi diteruskan ke AWS CloudFormation untuk melakukan penyebaran yang sebenarnya. Izinnya adalah izin yang akan dijalankan oleh penerapan. Izin diteruskan ke tumpukan sebagai parameter yang mencantumkan ARN kebijakan terkelola.

Output

AWS CDK Toolkit mensyaratkan bahwa CloudFormation output berikut ada di tumpukan bootstrap.

- `BucketName`: nama bucket aset file
- `BucketDomainName`: ember aset file dalam format nama domain
- `BootstrapVersion`: versi saat ini dari tumpukan bootstrap

Riwayat template

Template bootstrap berversi dan berkembang seiring waktu dengan dirinya sendiri. AWS CDK Jika Anda menyediakan template bootstrap Anda sendiri, tetap up to date dengan template default kanonik. Anda ingin memastikan bahwa template Anda terus bekerja dengan semua fitur CDK.

> **Note**
>
> Versi sebelumnya dari template bootstrap membuat AWS KMS key di setiap lingkungan bootstrap secara default. Untuk menghindari biaya untuk kunci KMS, bootstrap ulang lingkungan ini menggunakan. `--no-bootstrap-customer-key` Default saat ini tidak ada kunci KMS, yang membantu menghindari biaya ini.

Bagian ini berisi daftar perubahan yang dibuat di setiap versi.

Versi template	AWS CDK versi	Perubahan
1	1.40.0	Versi awal template dengan Bucket, Key, Repository, dan Roles.

Versi template	AWS CDK versi	Perubahan
2	1.45.0	Pisahkan peran penerbitan aset menjadi peran penerbitan file dan gambar yang terpisah.
3	1.46.0	Tambahkan `FileAsset KeyArn` ekspor untuk dapat menambahkan izin dekripsi ke konsumen aset.
4	1.61.0	AWS KMS izin sekarang tersirat melalui Amazon S3 dan tidak lagi diperluka n. `FileAsetKeyArn` Tambahkan parameter `CdkBootstrapVersion` SSM sehingga versi stack bootstrap dapat diverifikasi tanpa mengetahui nama tumpukan.
5	1.87.0	Peran penyebaran dapat membaca parameter SSM.
6	1.108.0	Tambahkan peran pencarian terpisah dari peran penerapan.
6	1.109.0	Lampirkan `aws-cdk:b ootstrap-role` tag ke peran penerapan, penerbitan file, dan penerbitan gambar.

Versi template	AWS CDK versi	Perubahan
7	1.110.0	Peran penyebaran tidak dapat lagi membaca Bucket di akun target secara langsung. (Namun, peran ini secara efektif adalah administrator, dan selalu dapat menggunakan AWS CloudFormation izinnya untuk membuat bucket dapat dibaca).
8	1.114.0	Peran pencarian memiliki izin hanya-baca penuh ke lingkungan target, dan memiliki `aws-cdk:bootstrap-role` tag juga.
9	2.1.0	Memperbaiki unggahan aset Amazon S3 agar tidak ditolak oleh SCP enkripsi yang biasa direferensikan.
10	2.4.0	Amazon ECR sekarang ScanOnPush diaktifkan secara default.
11	2.18.0	Menambahkan kebijakan yang memungkinkan Lambda menarik dari repo ECR Amazon sehingga bertahan dari bootstrap ulang.
12	2.20.0	Menambahkan dukungan untuk eksperimentalcdk import.

Versi template	AWS CDK versi	Perubahan
13	2.25.0	Membuat gambar kontainer di repositori Amazon ECR yang dibuat bootstrap tidak dapat diubah.
14	2.34.0	Mematikan pemindaian gambar Amazon ECR di tingkat repositori secara default untuk memungkinkan bootstrapping Wilayah yang tidak mendukung pemindaian gambar.
15	2.60.0	Kunci KMS tidak dapat ditandai.
16	2.69.0	Alamat Security Hub menemukan KMS.2.
17	2.72.0	Alamat Security Hub menemukan ECR.3.
18	2.80.0	Perubahan yang dikembalikan dibuat untuk versi 16 karena tidak berfungsi di semua partisi dan tidak disarankan.
19	2.106.1	Perubahan yang dikembali kan dibuat ke versi 18 di mana AccessControl properti telah dihapus dari template. (#27964)

Versi template	AWS CDK versi	Perubahan
20	2.119.0	Tambahkan `ssm:GetParameters` tindakan ke peran IAM AWS CloudFormation penerapan. Untuk informasi selengkapnya, lihat #28336.

Temuan Security Hub

Jika Anda menggunakan AWS Security Hub, Anda mungkin melihat temuan yang dilaporkan pada beberapa sumber daya yang dibuat oleh proses AWS CDK Bootstrapping. Temuan Security Hub membantu Anda menemukan konfigurasi sumber daya yang harus Anda periksa ulang untuk akurasi dan keamanannya. Kami telah meninjau konfigurasi sumber daya khusus ini dengan AWS Keamanan dan yakin itu bukan merupakan masalah keamanan.

[KMS.2] Prinsipal IAM tidak boleh memiliki kebijakan inline IAM yang memungkinkan tindakan dekripsi pada semua kunci KMS

Peran Deploy (nama default`cdk-hnb659fds-deploy-role-ACCOUNT-REGION`) memiliki izin untuk membaca data terenkripsi yang disimpan di Amazon S3. Kebijakan ini tidak memberikan izin ke data apa pun dengan sendirinya: hanya data yang dibaca dari Amazon S3 yang dapat didekripsi, dan hanya dari bucket yang secara eksplisit mengizinkan Peran Penerapan membaca dari mereka melalui Kebijakan Bucket mereka, dan kunci yang secara eksplisit memungkinkan Peran Penerapan untuk mendekripsi menggunakan Kebijakan Kunci mereka. Pernyataan ini digunakan untuk memungkinkan AWS CDK Pipelines melakukan penyebaran lintas akun.

Mengapa Security Hub menandai ini? Kebijakan berisi `Resource: *` gabungan dengan `Condition` klausa; Security Hub menandai. * *Hal ini diperlukan karena pada saat akun di-bootstrap, AWS KMS kunci yang dibuat oleh AWS CDK Pipelines untuk CodePipeline Artifact Bucket belum ada sehingga kami tidak dapat mereferensikan ARN-nya. Selain itu, Security Hub tidak menyertakan `Condition` klausul dalam pernyataan kebijakan dalam alasannya.

Bagaimana jika saya ingin memperbaiki temuan ini? Selama kebijakan sumber daya pada AWS KMS kunci Anda tidak perlu permisif, kebijakan Peran saat ini tidak mengizinkan Peran Penerapan mengakses data lebih dari yang seharusnya. Jika Anda masih ingin menyingkirkan temuan ini,

Anda dapat melakukannya dengan menyesuaikan tumpukan bootstrap (menggunakan proses yang diuraikan di atas) dengan salah satu dari 2 cara berikut:

- Jika Anda tidak menggunakan AWS CDK Pipelines untuk penyebaran lintas akun: hapus pernyataan dengan `Sid: PipelineCrossAccountArtifactsBucket` dari peran penerapan; atau
- Jika Anda menggunakan AWS CDK Pipelines untuk penerapan lintas akun: setelah menerapkan AWS CDK Pipeline Anda, cari AWS KMS ARN Kunci Bucket Artifact dan ganti `Resource: *` pernyataan dengan ARN Kunci yang sebenarnya. `Sid: PipelineCrossAccountArtifactsBucket`

Sumber daya

Sumber daya adalah apa yang Anda konfigurasikan untuk digunakan Layanan AWS dalam aplikasi Anda. Sumber daya adalah fitur dari AWS CloudFormation. Dengan mengonfigurasi sumber daya dan propertinya dalam AWS CloudFormation templat, Anda dapat menerapkan AWS CloudFormation untuk menyediakan sumber daya Anda. Dengan AWS Cloud Development Kit (AWS CDK), Anda dapat mengonfigurasi sumber daya melalui konstruksi. Anda kemudian menerapkan aplikasi CDK Anda, yang melibatkan sintesis AWS CloudFormation template dan penerapan untuk AWS CloudFormation menyediakan sumber daya Anda.

Topik

Mengkonfigurasi sumber daya menggunakan konstruksi

Seperti dijelaskan dalamthe section called “Membangun”, AWS CDK menyediakan perpustakaan kelas kaya konstruksi, yang disebut AWS konstruksi, yang mewakili semua AWS sumber daya.

Untuk membuat instance sumber daya menggunakan konstruksi yang sesuai, teruskan lingkup sebagai argumen pertama, ID logis dari konstruksi, dan satu set properti konfigurasi (props). Misalnya, berikut cara membuat antrean Amazon SQS dengan AWS KMS enkripsi menggunakan konstruksi SQS.Queue dari Construct Library. AWS

TypeScript

```
import * as sqs from '@aws-cdk/aws-sqs';

new sqs.Queue(this, 'MyQueue', {
    encryption: sqs.QueueEncryption.KMS_MANAGED
});
```

JavaScript

```
const sqs = require('@aws-cdk/aws-sqs');

new sqs.Queue(this, 'MyQueue', {
    encryption: sqs.QueueEncryption.KMS_MANAGED
});
```

Python

```
import aws_cdk.aws_sqs as sqs

sqs.Queue(self, "MyQueue", encryption=sqs.QueueEncryption.KMS_MANAGED)
```

Java

```
import software.amazon.awscdk.services.sqs.*;

Queue.Builder.create(this, "MyQueue").encryption(
        QueueEncryption.KMS_MANAGED).build();
```

C#

```
using Amazon.CDK.AWS.SQS;

new Queue(this, "MyQueue", new QueueProps
{
    Encryption = QueueEncryption.KMS_MANAGED
});
```

Beberapa alat peraga konfigurasi bersifat opsional, dan dalam banyak kasus memiliki nilai default. Dalam beberapa kasus, semua alat peraga bersifat opsional, dan argumen terakhir dapat dihilangkan seluruhnya.

Atribut sumber daya

Sebagian besar sumber daya dalam AWS Construct Library mengekspos atribut, yang diselesaikan pada waktu penerapan oleh. AWS CloudFormation Atribut diekspos dalam bentuk properti pada kelas sumber daya dengan nama tipe sebagai awalan. Contoh berikut menunjukkan cara mendapatkan URL antrian Amazon SQS menggunakan properti (`queueUrl`Python:). `queue_url`

TypeScript

```
import * as sqs from '@aws-cdk/aws-sqs';

const queue = new sqs.Queue(this, 'MyQueue');
const url = queue.queueUrl; // => A string representing a deploy-time value
```

JavaScript

```
const sqs = require('@aws-cdk/aws-sqs');

const queue = new sqs.Queue(this, 'MyQueue');
const url = queue.queueUrl; // => A string representing a deploy-time value
```

Python

```
import aws_cdk.aws_sqs as sqs

queue = sqs.Queue(self, "MyQueue")
```

```
url = queue.queue_url # => A string representing a deploy-time value
```

Java

```
Queue queue = new Queue(this, "MyQueue");
String url = queue.getQueueUrl();    // => A string representing a deploy-time value
```

C#

```
var queue = new Queue(this, "MyQueue");
var url = queue.QueueUrl; // => A string representing a deploy-time value
```

Lihat the section called “Token” untuk informasi tentang cara AWS CDK mengkodekan atribut deploy-time sebagai string.

Referensi sumber daya

Saat mengonfigurasi sumber daya, Anda sering harus mereferensikan properti dari sumber daya lain. Berikut ini adalah beberapa contohnya:

- Sumber daya Amazon Elastic Container Service (Amazon ECS) memerlukan referensi ke cluster tempat ia berjalan.
- CloudFront Distribusi Amazon memerlukan referensi ke bucket Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) yang berisi kode sumber.

Anda dapat mereferensikan sumber daya dengan salah satu cara berikut:

- Dengan meneruskan sumber daya yang ditentukan dalam aplikasi CDK Anda, baik di tumpukan yang sama atau di tumpukan yang berbeda
- Dengan meneruskan objek proxy yang mereferensikan sumber daya yang ditentukan di AWS akun Anda, dibuat dari pengenal unik sumber daya (seperti ARN)

Jika properti konstruksi mewakili konstruksi untuk sumber daya lain, tipenya adalah tipe antarmuka konstruksi. Misalnya, konstruksi Amazon ECS mengambil properti `cluster` tipe. `ecs.ICluster` Contoh lain, adalah konstruksi CloudFront distribusi yang mengambil properti `sourceBucket` (Python`source_bucket`:) tipe. `s3.IBucket`

Anda dapat langsung meneruskan objek sumber daya apa pun dari jenis yang tepat yang ditentukan dalam AWS CDK aplikasi yang sama. Contoh berikut mendefinisikan cluster Amazon ECS dan kemudian menggunakannya untuk menentukan layanan Amazon ECS.

TypeScript

```
const cluster = new ecs.Cluster(this, 'Cluster', { /*...*/ });

const service = new ecs.Ec2Service(this, 'Service', { cluster: cluster });
```

JavaScript

```
const cluster = new ecs.Cluster(this, 'Cluster', { /*...*/ });

const service = new ecs.Ec2Service(this, 'Service', { cluster: cluster });
```

Python

```
cluster = ecs.Cluster(self, "Cluster")

service = ecs.Ec2Service(self, "Service", cluster=cluster)
```

Java

```
Cluster cluster = new Cluster(this, "Cluster");
Ec2Service service = new Ec2Service(this, "Service",
        new Ec2ServiceProps.Builder().cluster(cluster).build());
```

C#

```
var cluster = new Cluster(this, "Cluster");
var service = new Ec2Service(this, "Service", new Ec2ServiceProps { Cluster =
 cluster });
```

Mereferensikan sumber daya dalam tumpukan yang berbeda

Anda dapat merujuk ke sumber daya di tumpukan yang berbeda selama mereka didefinisikan dalam aplikasi yang sama dan berada di AWS lingkungan yang sama. Pola berikut ini umumnya digunakan:

- Menyimpan referensi ke konstruksi sebagai atribut tumpukan yang menghasilkan sumber daya. (Untuk mendapatkan referensi ke tumpukan konstruksi saat ini, gunakan`Stack.of(this)`.)
- Teruskan referensi ini ke konstruktor tumpukan yang mengkonsumsi sumber daya sebagai parameter atau properti. Tumpukan konsumsi kemudian meneruskannya sebagai properti ke konstruksi apa pun yang membutuhkannya.

Contoh berikut mendefinisikan tumpukan`stack1`. Tumpukan ini mendefinisikan bucket Amazon S3 dan menyimpan referensi ke konstruksi bucket sebagai atribut tumpukan. Kemudian aplikasi mendefinisikan tumpukan kedua`stack2`, yang menerima bucket saat instantiation. `stack2`mungkin, misalnya, mendefinisikan AWS Glue Tabel yang menggunakan bucket untuk penyimpanan data.

TypeScript

```
const prod = { account: '123456789012', region: 'us-east-1' };

const stack1 = new StackThatProvidesABucket(app, 'Stack1', { env: prod });

// stack2 will take a property { bucket: IBucket }
const stack2 = new StackThatExpectsABucket(app, 'Stack2', {
  bucket: stack1.bucket,
  env: prod
});
```

JavaScript

```
const prod = { account: '123456789012', region: 'us-east-1' };

const stack1 = new StackThatProvidesABucket(app, 'Stack1', { env: prod });

// stack2 will take a property { bucket: IBucket }
const stack2 = new StackThatExpectsABucket(app, 'Stack2', {
  bucket: stack1.bucket,
  env: prod
});
```

Python

```
prod = core.Environment(account="123456789012", region="us-east-1")

stack1 = StackThatProvidesABucket(app, "Stack1", env=prod)
```

```
# stack2 will take a property "bucket"
stack2 = StackThatExpectsABucket(app, "Stack2", bucket=stack1.bucket, env=prod)
```

Java

```
// Helper method to build an environment
static Environment makeEnv(String account, String region) {
    return Environment.builder().account(account).region(region)
            .build();
}

App app = new App();

Environment prod = makeEnv("123456789012", "us-east-1");

StackThatProvidesABucket stack1 = new StackThatProvidesABucket(app, "Stack1",
        StackProps.builder().env(prod).build());

// stack2 will take an argument "bucket"
StackThatExpectsABucket stack2 = new StackThatExpectsABucket(app, "Stack,",
        StackProps.builder().env(prod).build(), stack1.bucket);
```

C#

```
Amazon.CDK.Environment makeEnv(string account, string region)
{
    return new Amazon.CDK.Environment { Account = account, Region = region };
}

var prod = makeEnv(account: "123456789012", region: "us-east-1");

var stack1 = new StackThatProvidesABucket(app, "Stack1", new StackProps { Env =
 prod });

// stack2 will take a property "bucket"
var stack2 = new StackThatExpectsABucket(app, "Stack2", new StackProps { Env = prod,
    bucket = stack1.Bucket});
```

Jika AWS CDK menentukan bahwa sumber daya berada di lingkungan yang sama, tetapi dalam tumpukan yang berbeda, secara otomatis mensintesis AWS CloudFormation ekspor di tumpukan

penghasil dan Fn:: ImportValue di tumpukan konsumsi untuk mentransfer informasi itu dari satu tumpukan ke tumpukan lainnya.

Menyelesaikan kebuntuan ketergantungan

Mereferensikan sumber daya dari satu tumpukan di tumpukan yang berbeda menciptakan ketergantungan antara dua tumpukan. Ini memastikan bahwa mereka dikerahkan dalam urutan yang benar. Setelah tumpukan digunakan, ketergantungan ini konkret. Setelah itu, menghapus penggunaan sumber daya bersama dari tumpukan konsumsi dapat menyebabkan kegagalan penerapan yang tidak terduga. Ini terjadi jika ada ketergantungan lain antara dua tumpukan yang memaksa mereka untuk digunakan dalam urutan yang sama. Ini juga dapat terjadi tanpa ketergantungan jika tumpukan penghasil hanya dipilih oleh CDK Toolkit untuk digunakan terlebih dahulu. AWS CloudFormation Ekspor dihapus dari tumpukan penghasil karena tidak lagi diperlukan, tetapi sumber daya yang diekspor masih digunakan di tumpukan konsumsi karena pembaruannya belum diterapkan. Oleh karena itu, penerapan tumpukan produsen gagal.

Untuk memecahkan kebuntuan ini, hapus penggunaan sumber daya bersama dari tumpukan konsumsi. (Ini menghapus ekspor otomatis dari tumpukan produksi.) Selanjutnya, tambahkan ekspor yang sama secara manual ke tumpukan penghasil menggunakan ID logis yang sama persis dengan ekspor yang dihasilkan secara otomatis. Hapus penggunaan sumber daya bersama di tumpukan konsumsi dan terapkan kedua tumpukan. Kemudian, hapus ekspor manual (dan sumber daya bersama jika tidak lagi diperlukan) dan gunakan kedua tumpukan lagi. `exportValue()`Metode tumpukan adalah cara mudah untuk membuat ekspor manual untuk tujuan ini. (Lihat contoh dalam referensi metode tertaut.)

Mereferensikan sumber daya di akun Anda AWS

Misalkan Anda ingin menggunakan sumber daya yang sudah tersedia di AWS akun Anda di AWS CDK aplikasi Anda. Ini mungkin sumber daya yang didefinisikan melalui konsol, AWS SDK, langsung dengan AWS CloudFormation, atau dalam AWS CDK aplikasi yang berbeda. Anda dapat mengubah ARN sumber daya (atau atribut pengenal lainnya, atau grup atribut) menjadi objek proxy. Objek proxy berfungsi sebagai referensi ke sumber daya dengan memanggil metode pabrik statis pada kelas sumber daya.

Saat Anda membuat proxy seperti itu, sumber daya eksternal tidak menjadi bagian dari AWS CDK aplikasi Anda. Oleh karena itu, perubahan yang Anda buat pada proxy di AWS CDK aplikasi Anda tidak memengaruhi sumber daya yang diterapkan. Proxy dapat, bagaimanapun, diteruskan ke AWS CDK metode apa pun yang membutuhkan sumber daya jenis itu.

Contoh berikut menunjukkan cara mereferensikan bucket berdasarkan bucket yang ada dengan ARN arn:aws:s3:::, dan Amazon my-bucket-name Virtual Private Cloud berdasarkan VPC yang sudah ada yang memiliki ID tertentu.

TypeScript

```
// Construct a proxy for a bucket by its name (must be same account)
s3.Bucket.fromBucketName(this, 'MyBucket', 'my-bucket-name');

// Construct a proxy for a bucket by its full ARN (can be another account)
s3.Bucket.fromBucketArn(this, 'MyBucket', 'arn:aws:s3:::my-bucket-name');

// Construct a proxy for an existing VPC from its attribute(s)
ec2.Vpc.fromVpcAttributes(this, 'MyVpc', {
  vpcId: 'vpc-1234567890abcde',
});
```

JavaScript

```
// Construct a proxy for a bucket by its name (must be same account)
s3.Bucket.fromBucketName(this, 'MyBucket', 'my-bucket-name');

// Construct a proxy for a bucket by its full ARN (can be another account)
s3.Bucket.fromBucketArn(this, 'MyBucket', 'arn:aws:s3:::my-bucket-name');

// Construct a proxy for an existing VPC from its attribute(s)
ec2.Vpc.fromVpcAttributes(this, 'MyVpc', {
  vpcId: 'vpc-1234567890abcde'
});
```

Python

```
# Construct a proxy for a bucket by its name (must be same account)
s3.Bucket.from_bucket_name(self, "MyBucket", "my-bucket-name")

# Construct a proxy for a bucket by its full ARN (can be another account)
s3.Bucket.from_bucket_arn(self, "MyBucket", "arn:aws:s3:::my-bucket-name")

# Construct a proxy for an existing VPC from its attribute(s)
ec2.Vpc.from_vpc_attributes(self, "MyVpc", vpc_id="vpc-1234567890abcdef")
```

Java

```
// Construct a proxy for a bucket by its name (must be same account)
Bucket.fromBucketName(this, "MyBucket", "my-bucket-name");

// Construct a proxy for a bucket by its full ARN (can be another account)
Bucket.fromBucketArn(this, "MyBucket",
        "arn:aws:s3:::my-bucket-name");

// Construct a proxy for an existing VPC from its attribute(s)
Vpc.fromVpcAttributes(this, "MyVpc", VpcAttributes.builder()
        .vpcId("vpc-1234567890abcdef").build());
```

C#

```
// Construct a proxy for a bucket by its name (must be same account)
Bucket.FromBucketName(this, "MyBucket", "my-bucket-name");

// Construct a proxy for a bucket by its full ARN (can be another account)
Bucket.FromBucketArn(this, "MyBucket", "arn:aws:s3:::my-bucket-name");

// Construct a proxy for an existing VPC from its attribute(s)
Vpc.FromVpcAttributes(this, "MyVpc", new VpcAttributes
{
    VpcId = "vpc-1234567890abcdef"
});
```

Mari kita lihat lebih dekat `Vpc.fromLookup()`metodenya. Karena `ec2.Vpc` konstruksinya rumit, ada banyak cara Anda mungkin ingin memilih VPC yang akan digunakan dengan aplikasi CDK Anda. Untuk mengatasi hal ini, konstruksi VPC memiliki metode `fromLookup` statis (Python:`from_lookup`) yang memungkinkan Anda mencari VPC Amazon yang diinginkan dengan menanyakan akun Anda pada waktu sintesis. AWS

Untuk menggunakannya`Vpc.fromLookup()`, sistem yang mensintesis tumpukan harus memiliki akses ke akun yang memiliki Amazon VPC. Ini karena CDK Toolkit menanyakan akun untuk menemukan VPC Amazon yang tepat pada waktu sintesis.

Selanjutnya, hanya `Vpc.fromLookup()` berfungsi di tumpukan yang didefinisikan dengan akun dan wilayah eksplisit (lihatthe section called “Lingkungan”). Jika AWS CDK mencoba mencari VPC

Amazon dari tumpukan agnostik lingkungan, CDK Toolkit tidak tahu lingkungan mana yang harus ditanyakan untuk menemukan VPC.

Anda harus memberikan `Vpc.fromLookup()` atribut yang cukup untuk mengidentifikasi VPC secara unik di akun Anda. AWS Misalnya, hanya ada satu VPC default, jadi cukup untuk menentukan VPC sebagai default.

TypeScript

```
ec2.Vpc.fromLookup(this, 'DefaultVpc', {
  isDefault: true
});
```

JavaScript

```
ec2.Vpc.fromLookup(this, 'DefaultVpc', {
  isDefault: true
});
```

Python

```
ec2.Vpc.from_lookup(self, "DefaultVpc", is_default=True)
```

Java

```
Vpc.fromLookup(this, "DefaultVpc", VpcLookupOptions.builder()
        .isDefault(true).build());
```

C#

```
Vpc.FromLookup(this, id = "DefaultVpc", new VpcLookupOptions { IsDefault = true });
```

Anda juga dapat menggunakan `tags` properti untuk query untuk VPC dengan tag. Anda dapat menambahkan tag ke VPC Amazon pada saat pembuatannya dengan menggunakan AWS CloudFormation atau. AWS CDK Anda dapat mengedit tag kapan saja setelah pembuatan dengan menggunakan AWS Management Console, SDK AWS CLI, atau AWS SDK. Selain tag apa pun yang Anda tambahkan sendiri, secara AWS CDK otomatis menambahkan tag berikut ke semua VPC yang dibuatnya.

- Nama — Nama VPC.
- aws-cdk:subnet-name — Nama subnet.
- aws-cdk:subnet-type — Jenis subnet: Publik, Pribadi, atau Terisolasi.

TypeScript

```
ec2.Vpc.fromLookup(this, 'PublicVpc',
    {tags: {'aws-cdk:subnet-type': "Public"}});
```

JavaScript

```
ec2.Vpc.fromLookup(this, 'PublicVpc',
    {tags: {'aws-cdk:subnet-type': "Public"}});
```

Python

```
ec2.Vpc.from_lookup(self, "PublicVpc",
    tags={"aws-cdk:subnet-type": "Public"})
```

Java

```
Vpc.fromLookup(this, "PublicVpc", VpcLookupOptions.builder()
        .tags(java.util.Map.of("aws-cdk:subnet-type", "Public"))  // Java 9 or later
        .build());
```

C#

```
Vpc.FromLookup(this, id = "PublicVpc", new VpcLookupOptions
     { Tags = new Dictionary<string, string> { ["aws-cdk:subnet-type"] =
 "Public" });
```

Hasil `Vpc.fromLookup()` cache dalam `cdk.context.json` file proyek. (Lihat the section called "Konteks"). Komit file ini ke kontrol versi sehingga aplikasi Anda akan terus merujuk ke VPC Amazon yang sama. Ini berfungsi bahkan jika nanti Anda mengubah atribut VPC Anda dengan cara yang akan menghasilkan VPC yang berbeda dipilih. Ini sangat penting jika Anda menerapkan tumpukan di lingkungan yang tidak memiliki akses ke AWS akun yang mendefinisikan VPC, seperti CDK Pipelines.

Meskipun Anda dapat menggunakan sumber daya eksternal di mana pun Anda menggunakan sumber daya serupa yang ditentukan dalam AWS CDK aplikasi, Anda tidak dapat memodifikasinya. Misalnya, memanggil `addToResourcePolicy` (Python:`add_to_resource_policy`) pada eksternal `s3.Bucket` tidak melakukan apa-apa.

Nama fisik sumber daya

Nama logis sumber daya AWS CloudFormation berbeda dari nama sumber daya yang ditampilkan AWS Management Console setelah digunakan oleh AWS CloudFormation. AWS CDK Disebut nama-nama akhir ini nama-nama fisik.

Misalnya, AWS CloudFormation mungkin membuat bucket Amazon S3 dengan ID logis `Stack2MyBucket4DD88B4F` dari contoh sebelumnya dengan nama fisik. `stack2mybucket4dd88b4f-iuv1rbv9z3to`

Anda dapat menentukan nama fisik saat membuat konstruksi yang mewakili sumber daya dengan menggunakan <resourceType>Nama properti. Contoh berikut membuat bucket Amazon S3 dengan nama fisik. `my-bucket-name`

TypeScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket-name',
});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket-name'
});
```

Python

```
bucket = s3.Bucket(self, "MyBucket", bucket_name="my-bucket-name")
```

Java

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
        .bucketName("my-bucket-name").build();
```

C#

```
var bucket = new Bucket(this, "MyBucket", new BucketProps { BucketName = "my-bucket-
name" });
```

Menetapkan nama fisik ke sumber daya memiliki beberapa kelemahan. AWS CloudFormation Yang paling penting, setiap perubahan pada sumber daya yang digunakan yang memerlukan penggantian sumber daya, seperti perubahan pada properti sumber daya yang tidak dapat diubah setelah pembuatan, akan gagal jika sumber daya memiliki nama fisik yang ditetapkan. Jika Anda berakhir dalam keadaan itu, satu-satunya solusi adalah menghapus AWS CloudFormation tumpukan, lalu menerapkan AWS CDK aplikasi lagi. Lihat AWS CloudFormation dokumentasi untuk detailnya.

Dalam beberapa kasus, seperti saat membuat AWS CDK aplikasi dengan referensi lintas lingkungan, nama fisik diperlukan AWS CDK agar berfungsi dengan benar. Dalam kasus tersebut, jika Anda tidak ingin repot-repot membuat nama fisik sendiri, Anda dapat membiarkan AWS CDK nama itu untuk Anda. Untuk melakukannya, gunakan nilai khususPhysicalName.GENERATE_IF_NEEDED, sebagai berikut.

TypeScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: core.PhysicalName.GENERATE_IF_NEEDED,
});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: core.PhysicalName.GENERATE_IF_NEEDED
});
```

Python

```
bucket = s3.Bucket(self, "MyBucket",
                           bucket_name=core.PhysicalName.GENERATE_IF_NEEDED)
```

Java

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
        .bucketName(PhysicalName.GENERATE_IF_NEEDED).build();
```

C#

```
var bucket = new Bucket(this, "MyBucket", new BucketProps
    { BucketName = PhysicalName.GENERATE_IF_NEEDED });
```

Melewati pengidentifikasi sumber daya unik

Bila memungkinkan, Anda harus melewatkan sumber daya dengan referensi, seperti yang dijelaskan di bagian sebelumnya. Namun, ada kasus di mana Anda tidak punya pilihan lain selain merujuk ke sumber daya dengan salah satu atributnya. Contoh kasus penggunaan meliputi yang berikut:

- Saat Anda menggunakan AWS CloudFormation sumber daya tingkat rendah.
- Saat Anda perlu mengekspos sumber daya ke komponen runtime AWS CDK aplikasi, seperti saat merujuk ke fungsi Lambda melalui variabel lingkungan.

Pengidentifikasi ini tersedia sebagai atribut pada sumber daya, seperti berikut ini.

TypeScript

```
bucket.bucketName
lambdaFunc.functionArn
securityGroup.groupArn
```

JavaScript

```
bucket.bucketName
lambdaFunc.functionArn
securityGroup.groupArn
```

Python

```
bucket.bucket_name
lambda_func.function_arn
security_group_arn
```

Java

AWS CDK Pengikatan Java menggunakan metode pengambil untuk atribut.

```
bucket.getBucketName()
lambdaFunc.getFunctionArn()
securityGroup.getGroupArn()
```

C#

```
bucket.BucketName
lambdaFunc.FunctionArn
securityGroup.GroupArn
```

Contoh berikut menunjukkan cara meneruskan nama bucket yang dihasilkan ke suatu AWS Lambda fungsi.

TypeScript

```
const bucket = new s3.Bucket(this, 'Bucket');

new lambda.Function(this, 'MyLambda', {
  // ...
  environment: {
    BUCKET_NAME: bucket.bucketName,
  },
});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'Bucket');

new lambda.Function(this, 'MyLambda', {
  // ...
  environment: {
    BUCKET_NAME: bucket.bucketName
  }
});
```

Python

```
bucket = s3.Bucket(self, "Bucket")

lambda.Function(self, "MyLambda", environment=dict(BUCKET_NAME=bucket.bucket_name))
```

Java

```
final Bucket bucket = new Bucket(this, "Bucket");

Function.Builder.create(this, "MyLambda")
        .environment(java.util.Map.of(    // Java 9 or later
                "BUCKET_NAME", bucket.getBucketName()))
        .build();
```

C#

```
var bucket = new Bucket(this, "Bucket");

new Function(this, "MyLambda", new FunctionProps
{
    Environment = new Dictionary<string, string>
    {
        ["BUCKET_NAME"] = bucket.BucketName
    }
});
```

Memberikan izin antar sumber daya

Konstruksi tingkat yang lebih tinggi membuat izin hak istimewa paling sedikit dapat dicapai dengan menawarkan API sederhana berbasis niat untuk menyatakan persyaratan izin. Misalnya, banyak konstruksi L2 menawarkan metode hibah yang dapat Anda gunakan untuk memberikan izin entitas (seperti peran IAM atau pengguna) untuk bekerja dengan sumber daya, tanpa harus membuat pernyataan izin IAM secara manual.

Contoh berikut membuat izin untuk mengizinkan peran eksekusi fungsi Lambda membaca dan menulis objek ke bucket Amazon S3 tertentu. Jika bucket Amazon S3 dienkripsi dengan AWS KMS kunci, metode ini juga memberikan izin ke peran eksekusi fungsi Lambda untuk mendekripsi dengan kunci.

TypeScript

```
if (bucket.grantReadWrite(func).success) {
  // ...
}
```

JavaScript

```
if ( bucket.grantReadWrite(func).success) {
  // ...
}
```

Python

```
if bucket.grant_read_write(func).success:
    # ...
```

Java

```
if (bucket.grantReadWrite(func).getSuccess()) {
    // ...
}
```

C#

```
if (bucket.GrantReadWrite(func).Success)
{
    // ...
}
```

Metode hibah mengembalikan `iam.Grant` objek. Gunakan `success` atribut `Grant` objek untuk menentukan apakah hibah diterapkan secara efektif (misalnya, mungkin tidak diterapkan pada sumber daya eksternal). Anda juga dapat menggunakan metode `assertSuccess` (Python:`assert_success`) `Grant` objek untuk menegakkan bahwa hibah berhasil diterapkan.

Jika metode hibah tertentu tidak tersedia untuk kasus penggunaan tertentu, Anda dapat menggunakan metode hibah generik untuk menentukan hibah baru dengan daftar tindakan tertentu.

Contoh berikut menunjukkan cara memberikan akses fungsi Lambda ke tindakan Amazon DynamoDB. `CreateBackup`

TypeScript

```
table.grant(func, 'dynamodb:CreateBackup');
```

JavaScript

```
table.grant(func, 'dynamodb:CreateBackup');
```

Python

```
table.grant(func, "dynamodb:CreateBackup")
```

Java

```
table.grant(func, "dynamodb:CreateBackup");
```

C#

```
table.Grant(func, "dynamodb:CreateBackup");
```

Banyak sumber daya, seperti fungsi Lambda, memerlukan peran yang harus diasumsikan saat mengeksekusi kode. Properti konfigurasi memungkinkan Anda untuk menentukan`iam.IRole`. Jika tidak ada peran yang ditentukan, fungsi secara otomatis membuat peran khusus untuk penggunaan ini. Anda kemudian dapat menggunakan metode hibah pada sumber daya untuk menambahkan pernyataan ke peran.

Metode hibah dibuat menggunakan API tingkat rendah untuk menangani kebijakan IAM. Kebijakan dimodelkan sebagai PolicyDocumentobjek. Tambahkan pernyataan langsung ke peran (atau peran terlampir konstruksi) menggunakan `addToRolePolicy` metode (Python`add_to_role_policy`:), atau ke kebijakan sumber daya (seperti `Bucket` kebijakan) menggunakan metode `addToResourcePolicy` (`add_to_resource_policy`Python:).

Metrik sumber daya dan alarm

Banyak sumber daya memancarkan CloudWatch metrik yang dapat digunakan untuk mengatur dasbor pemantauan dan alarm. Konstruksi tingkat yang lebih tinggi memiliki metode metrik yang memungkinkan Anda mengakses metrik tanpa mencari nama yang benar untuk digunakan.

Contoh berikut menunjukkan cara menentukan alarm ketika antrian Amazon SQS melebihi 100. `ApproximateNumberOfMessagesNotVisible`

TypeScript

```
import * as cw from '@aws-cdk/aws-cloudwatch';
import * as sqs from '@aws-cdk/aws-sqs';
import { Duration } from '@aws-cdk/core';

const queue = new sqs.Queue(this, 'MyQueue');

const metric = queue.metricApproximateNumberOfMessagesNotVisible({
  label: 'Messages Visible (Approx)',
  period: Duration.minutes(5),
  // ...
});
metric.createAlarm(this, 'TooManyMessagesAlarm', {
  comparisonOperator: cw.ComparisonOperator.GREATER_THAN_THRESHOLD,
  threshold: 100,
  // ...
});
```

JavaScript

```
const cw = require('@aws-cdk/aws-cloudwatch');
const sqs = require('@aws-cdk/aws-sqs');
const { Duration } = require('@aws-cdk/core');

const queue = new sqs.Queue(this, 'MyQueue');

const metric = queue.metricApproximateNumberOfMessagesNotVisible({
  label: 'Messages Visible (Approx)',
  period: Duration.minutes(5)
  // ...
});
metric.createAlarm(this, 'TooManyMessagesAlarm', {
  comparisonOperator: cw.ComparisonOperator.GREATER_THAN_THRESHOLD,
  threshold: 100
  // ...
});
```

Python

```
import aws_cdk.aws_cloudwatch as cw
import aws_cdk.aws_sqs as sqs
from aws_cdk.core import Duration
```

```
queue = sqs.Queue(self, "MyQueue")
metric = queue.metric_approximate_number_of_messages_not_visible(
    label="Messages Visible (Approx)",
    period=Duration.minutes(5),
    # ...
)
metric.create_alarm(self, "TooManyMessagesAlarm",
    comparison_operator=cw.ComparisonOperator.GREATER_THAN_THRESHOLD,
    threshold=100,
    # ...
)
```

Java

```
import software.amazon.awscdk.core.Duration;
import software.amazon.awscdk.services.sqs.Queue;
import software.amazon.awscdk.services.cloudwatch.Metric;
import software.amazon.awscdk.services.cloudwatch.MetricOptions;
import software.amazon.awscdk.services.cloudwatch.CreateAlarmOptions;
import software.amazon.awscdk.services.cloudwatch.ComparisonOperator;

Queue queue = new Queue(this, "MyQueue");

Metric metric = queue
        .metricApproximateNumberOfMessagesNotVisible(MetricOptions.builder()
                .label("Messages Visible (Approx)")
                .period(Duration.minutes(5)).build());

metric.createAlarm(this, "TooManyMessagesAlarm", CreateAlarmOptions.builder()
                .comparisonOperator(ComparisonOperator.GREATER_THAN_THRESHOLD)
                .threshold(100)
                // ...
                .build());
```

C#

```
using cdk = Amazon.CDK;
using cw = Amazon.CDK.AWS.CloudWatch;
using sqs = Amazon.CDK.AWS.SQS;

var queue = new sqs.Queue(this, "MyQueue");
var metric = queue.MetricApproximateNumberOfMessagesNotVisible(new cw.MetricOptions
```

```
{
    Label = "Messages Visible (Approx)",
    Period = cdk.Duration.Minutes(5),
    // ...
});
metric.CreateAlarm(this, "TooManyMessagesAlarm", new cw.CreateAlarmOptions
{
    ComparisonOperator = cw.ComparisonOperator.GREATER_THAN_THRESHOLD,
    Threshold = 100,
    // ..
});
```

Jika tidak ada metode untuk metrik tertentu, Anda dapat menggunakan metode metrik umum untuk menentukan nama metrik secara manual.

Metrik juga dapat ditambahkan ke CloudWatch dasbor. Lihat CloudWatch.

Lalu lintas jaringan

Dalam banyak kasus, Anda harus mengaktifkan izin pada jaringan agar aplikasi berfungsi, seperti ketika infrastruktur komputasi perlu mengakses lapisan persistensi. Sumber daya yang membangun atau mendengarkan koneksi mengekspos metode yang memungkinkan arus lalu lintas, termasuk menetapkan aturan grup keamanan atau ACL jaringan.

Sumber daya IconNectable memiliki `connections` properti yang merupakan gateway ke konfigurasi aturan lalu lintas jaringan.

Anda mengaktifkan data mengalir pada jalur jaringan tertentu dengan menggunakan `allow` metode. Contoh berikut memungkinkan koneksi HTTPS ke web dan koneksi masuk dari grup Auto Scaling Amazon EC2. `fleet2`

TypeScript

```
import * as asg from '@aws-cdk/aws-autoscaling';
import * as ec2 from '@aws-cdk/aws-ec2';

const fleet1: asg.AutoScalingGroup = asg.AutoScalingGroup(/*...*/);

// Allow surfing the (secure) web
fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443,
 toPort: 443 }));
```

```
const fleet2: asg.AutoScalingGroup = asg.AutoScalingGroup(/*...*/);
fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic());
```

JavaScript

```
const asg = require('@aws-cdk/aws-autoscaling');
const ec2 = require('@aws-cdk/aws-ec2');

const fleet1 = asg.AutoScalingGroup();

// Allow surfing the (secure) web
fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443,
 toPort: 443 }));

const fleet2 = asg.AutoScalingGroup();
fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic());
```

Python

```
import aws_cdk.aws_autoscaling as asg
import aws_cdk.aws_ec2 as ec2

fleet1 = asg.AutoScalingGroup( ... )

# Allow surfing the (secure) web
fleet1.connections.allow_to(ec2.Peer.any_ipv4(),
  ec2.Port(PortProps(from_port=443, to_port=443)))

fleet2 = asg.AutoScalingGroup( ... )
fleet1.connections.allow_from(fleet2, ec2.Port.all_traffic())
```

Java

```
import software.amazon.awscdk.services.autoscaling.AutoScalingGroup;
import software.amazon.awscdk.services.ec2.Peer;
import software.amazon.awscdk.services.ec2.Port;

AutoScalingGroup fleet1 = AutoScalingGroup.Builder.create(this, "MyFleet")
        /* ... */.build();

// Allow surfing the (secure) Web
fleet1.getConnections().allowTo(Peer.anyIpv4(),
```

```
        Port.Builder.create().fromPort(443).toPort(443).build());

AutoScalingGroup fleet2 = AutoScalingGroup.Builder.create(this, "MyFleet2")
        /* ... */.build();
fleet1.getConnections().allowFrom(fleet2, Port.allTraffic());
```

C#

```
using cdk = Amazon.CDK;
using asg = Amazon.CDK.AWS.AutoScaling;
using ec2 = Amazon.CDK.AWS.EC2;

// Allow surfing the (secure) Web
var fleet1 = new asg.AutoScalingGroup(this, "MyFleet", new asg.AutoScalingGroupProps
 { /* ... */ });
fleet1.Connections.AllowTo(ec2.Peer.AnyIpv4(), new ec2.Port(new ec2.PortProps
  { FromPort = 443, ToPort = 443 });

var fleet2 = new asg.AutoScalingGroup(this, "MyFleet2", new
 asg.AutoScalingGroupProps { /* ... */ });
fleet1.Connections.AllowFrom(fleet2, ec2.Port.AllTraffic());
```

Sumber daya tertentu memiliki port default yang terkait dengannya. Contohnya termasuk pendengar penyeimbang beban di port publik, dan port tempat mesin database menerima koneksi untuk instance database Amazon RDS. Dalam kasus seperti itu, Anda dapat menerapkan kontrol jaringan yang ketat tanpa harus menentukan port secara manual. Untuk melakukannya, gunakan `allowToDefaultPort` metode `allowDefaultPortFrom` dan (Python:`allow_default_port_from`,`allow_to_default_port`).

Contoh berikut menunjukkan cara mengaktifkan koneksi dari alamat IPV4 apa pun, dan koneksi dari grup Auto Scaling untuk mengakses database.

TypeScript

```
listener.connections.allowDefaultPortFromAnyIpv4('Allow public access');

fleet.connections.allowToDefaultPort(rdsDatabase, 'Fleet can access database');
```

JavaScript

```
listener.connections.allowDefaultPortFromAnyIpv4('Allow public access');
```

```
fleet.connections.allowToDefaultPort(rdsDatabase, 'Fleet can access database');
```

Python

```
listener.connections.allow_default_port_from_any_ipv4("Allow public access")

fleet.connections.allow_to_default_port(rds_database, "Fleet can access database")
```

Java

```
listener.getConnections().allowDefaultPortFromAnyIpv4("Allow public access");

fleet.getConnections().AllowToDefaultPort(rdsDatabase, "Fleet can access database");
```

C#

```
listener.Connections.AllowDefaultPortFromAnyIpv4("Allow public access");

fleet.Connections.AllowToDefaultPort(rdsDatabase, "Fleet can access database");
```

Penanganan acara

Beberapa sumber daya dapat bertindak sebagai sumber acara. Gunakan `addEventNotification` metode (Python:`add_event_notification`) untuk mendaftarkan target acara ke jenis peristiwa tertentu yang dipancarkan oleh sumber daya. Selain itu, `addXxxNotification` metode menawarkan cara sederhana untuk mendaftarkan handler untuk jenis acara umum.

Contoh berikut menunjukkan cara memicu fungsi Lambda saat objek ditambahkan ke bucket Amazon S3.

TypeScript

```
import * as s3nots from '@aws-cdk/aws-s3-notifications';

const handler = new lambda.Function(this, 'Handler', { /*…*/ });
const bucket = new s3.Bucket(this, 'Bucket');
bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler));
```

JavaScript

```
const s3nots = require('@aws-cdk/aws-s3-notifications');

const handler = new lambda.Function(this, 'Handler', { /*…*/ });
const bucket = new s3.Bucket(this, 'Bucket');
bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler));
```

Python

```
import aws_cdk.aws_s3_notifications as s3_nots

handler = lambda_.Function(self, "Handler", ...)
bucket = s3.Bucket(self, "Bucket")
bucket.add_object_created_notification(s3_nots.LambdaDestination(handler))
```

Java

```
import software.amazon.awscdk.services.s3.Bucket;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.s3.notifications.LambdaDestination;

Function handler = Function.Builder.create(this, "Handler")/* ... */.build();
Bucket bucket = new Bucket(this, "Bucket");
bucket.addObjectCreatedNotification(new LambdaDestination(handler));
```

C#

```
using lambda = Amazon.CDK.AWS.Lambda;
using s3 = Amazon.CDK.AWS.S3;
using s3Nots = Amazon.CDK.AWS.S3.Notifications;

var handler = new lambda.Function(this, "Handler", new lambda.FunctionProps { .. });
var bucket = new s3.Bucket(this, "Bucket");
bucket.AddObjectCreatedNotification(new s3Nots.LambdaDestination(handler));
```

Kebijakan penghapusan

Sumber daya yang menyimpan data persisten, seperti database, bucket Amazon S3, dan pendaftar Amazon ECR, memiliki kebijakan penghapusan. Kebijakan penghapusan menunjukkan apakah

akan menghapus objek persisten saat AWS CDK tumpukan yang berisi objek tersebut dihancurkan. Nilai yang menentukan kebijakan penghapusan tersedia melalui `RemovalPolicy` enumerasi dalam modul. AWS CDK `core`

> **Note**
>
> Sumber daya selain yang menyimpan data secara terus-menerus mungkin juga memiliki `removalPolicy` yang digunakan untuk tujuan yang berbeda. Misalnya, versi fungsi Lambda menggunakan `removalPolicy` atribut untuk menentukan apakah versi tertentu dipertahankan saat versi baru diterapkan. Ini memiliki arti dan default yang berbeda dibandingkan dengan kebijakan penghapusan pada bucket Amazon S3 atau tabel DynamoDB.

Nilai	arti
`RemovalPolicy.MEMPERTAHANKAN`	Keep the contents of the resource when destroying the stack (default). The resource is orphaned from the stack and must be deleted manually. If you attempt to re-deploy the stack while the resource still exists, you will receive an error message due to a name conflict.
`RemovalPolicy.HANCURKAN`	The resource will be destroyed along with the stack.

AWS CloudFormation tidak menghapus bucket Amazon S3 yang berisi file meskipun kebijakan penghapusannya disetel ke. `DESTROY` Mencoba melakukannya adalah AWS CloudFormation kesalahan. Untuk AWS CDK menghapus semua file dari bucket sebelum menghancurkannya, atur `autoDeleteObjects` properti bucket ke`true`.

Berikut ini adalah contoh membuat bucket Amazon S3 dengan `RemovalPolicy` of `DESTROY` dan `autoDeleteOjbects` disetel ke. `true`

TypeScript

```
import * as cdk from '@aws-cdk/core';
```

```
import * as s3 from '@aws-cdk/aws-s3';

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

    const bucket = new s3.Bucket(this, 'Bucket', {
      removalPolicy: cdk.RemovalPolicy.DESTROY,
      autoDeleteObjects: true
    });
  }
}
```

JavaScript

```
const cdk = require('@aws-cdk/core');
const s3 = require('@aws-cdk/aws-s3');

class CdkTestStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const bucket = new s3.Bucket(this, 'Bucket', {
      removalPolicy: cdk.RemovalPolicy.DESTROY,
      autoDeleteObjects: true
    });
  }
}

module.exports = { CdkTestStack }
```

Python

```
import aws_cdk.core as cdk
import aws_cdk.aws_s3 as s3

class CdkTestStack(cdk.stack):
    def __init__(self, scope: cdk.Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        bucket = s3.Bucket(self, "Bucket",
            removal_policy=cdk.RemovalPolicy.DESTROY,
            auto_delete_objects=True)
```

Java

```
software.amazon.awscdk.core.*;
import software.amazon.awscdk.services.s3.*;

public class CdkTestStack extends Stack {
    public CdkTestStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public CdkTestStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        Bucket.Builder.create(this, "Bucket")
                .removalPolicy(RemovalPolicy.DESTROY)
                .autoDeleteObjects(true).build();
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;

public CdkTestStack(Construct scope, string id, IStackProps props) : base(scope, id,
 props)
{
    new Bucket(this, "Bucket", new BucketProps {
        RemovalPolicy = RemovalPolicy.DESTROY,
        AutoDeleteObjects = true
    });
}
```

Anda juga dapat menerapkan kebijakan penghapusan langsung ke AWS CloudFormation sumber daya yang mendasarinya melalui `applyRemovalPolicy()` metode ini. Metode ini tersedia pada beberapa sumber daya stateful yang tidak memiliki `removalPolicy` properti di alat peraga sumber daya L2 mereka. Contohnya meliputi hal berikut:

- AWS CloudFormation tumpukan
- Kolam pengguna Amazon Cognito

- Contoh basis data Amazon DocumentDB
- Volume Amazon EC2
- Domain OpenSearch Layanan Amazon
- Sistem file Amazon FSx
- Antrean Amazon SQS

TypeScript

```
const resource = bucket.node.findChild('Resource') as cdk.CfnResource;
resource.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY);
```

JavaScript

```
const resource = bucket.node.findChild('Resource');
resource.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY);
```

Python

```
resource = bucket.node.find_child('Resource')
resource.apply_removal_policy(cdk.RemovalPolicy.DESTROY);
```

Java

```
CfnResource resource = (CfnResource)bucket.node.findChild("Resource");
resource.applyRemovalPolicy(cdk.RemovalPolicy.DESTROY);
```

C#

```

var resource = (CfnResource)bucket.node.findChild('Resource');
resource.ApplyRemovalPolicy(cdk.RemovalPolicy.DESTROY);
```

> **Note**
>
> AWS CDK Ini `RemovalPolicy` diterjemahkan menjadi AWS CloudFormation's. `DeletionPolicy` Namun, AWS CDK defaultnya adalah mempertahankan data, yang merupakan kebalikan dari AWS CloudFormation default.

Pengidentifikasi

Saat membuat AWS Cloud Development Kit (AWS CDK) aplikasi, Anda akan menggunakan banyak jenis pengenal dan nama. Untuk menggunakan secara AWS CDK efektif dan menghindari kesalahan, penting untuk memahami jenis pengidentifikasi.

Pengidentifikasi harus unik dalam ruang lingkup di mana mereka dibuat; mereka tidak perlu unik secara global dalam AWS CDK aplikasi Anda.

Jika Anda mencoba membuat pengenal dengan nilai yang sama dalam lingkup yang sama, maka akan AWS CDK melempar pengecualian.

Topik

Membangun ID

Pengidentifikasi yang paling umum,`id`, adalah pengidentifikasi yang diteruskan sebagai argumen kedua saat membuat instance objek konstruksi. Pengidentifikasi ini, seperti semua pengidentifikasi, hanya perlu unik dalam lingkup di mana ia dibuat, yang merupakan argumen pertama ketika membuat instance objek konstruksi.

> **Note**
>
> Tumpukan juga merupakan pengidentifikasi yang Anda gunakan untuk merujuknya di. `id` the section called “AWS CDK Toolkit”

Mari kita lihat contoh di mana kita memiliki dua konstruksi dengan identifier `MyBucket` di aplikasi kita. Yang pertama didefinisikan dalam lingkup tumpukan dengan pengenal`Stack1`. Yang kedua didefinisikan dalam lingkup tumpukan dengan pengenal`Stack2`. Karena mereka didefinisikan dalam cakupan yang berbeda, ini tidak menyebabkan konflik apa pun, dan mereka dapat hidup berdampingan di aplikasi yang sama tanpa masalah.

TypeScript

```
import { App, Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as s3 from 'aws-cdk-lib/aws-s3';

class MyStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps = {}) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyBucket');
  }
}

const app = new App();
new MyStack(app, 'Stack1');
new MyStack(app, 'Stack2');
```

JavaScript

```
const { App , Stack } = require('aws-cdk-lib');
const s3 = require('aws-cdk-lib/aws-s3');

class MyStack extends Stack {
  constructor(scope, id, props = {}) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyBucket');
  }
}

const app = new App();
new MyStack(app, 'Stack1');
new MyStack(app, 'Stack2');
```

Python

```
from aws_cdk import App, Construct, Stack, StackProps
from constructs import Construct
from aws_cdk import aws_s3 as s3

class MyStack(Stack):
```

```
    def __init__(self, scope: Construct, id: str, **kwargs):

        super().__init__(scope, id, **kwargs)
        s3.Bucket(self, "MyBucket")

app = App()
MyStack(app, 'Stack1')
MyStack(app, 'Stack2')
```

Java

```
// MyStack.java
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.constructs.Construct;
import software.amazon.awscdk.services.s3.Bucket;

public class MyStack extends Stack {
    public MyStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyStack(final Construct scope, final String id, final StackProps props) {
        super(scope, id, props);
        new Bucket(this, "MyBucket");
    }
}

// Main.java
package com.myorg;

import software.amazon.awscdk.App;

public class Main {
    public static void main(String[] args) {
        App app = new App();
        new MyStack(app, "Stack1");
        new MyStack(app, "Stack2");
    }
}
```

C#

```
using Amazon.CDK;
using constructs;
using Amazon.CDK.AWS.S3;

public class MyStack : Stack
{
    public MyStack(Construct scope, string id, IStackProps props) : base(scope, id,
 props)
    {
        new Bucket(this, "MyBucket");
    }
}

class Program
{
    static void Main(string[] args)
    {
        var app = new App();
        new MyStack(app, "Stack1");
        new MyStack(app, "Stack2");
    }
}
```

Jalan

Konstruksi dalam AWS CDK aplikasi membentuk hierarki yang berakar di kelas. App Kami mengacu pada kumpulan ID dari konstruksi tertentu, konstruksi induknya, kakek-neneknya, dan seterusnya ke akar pohon konstruksi, sebagai jalur.

AWS CDK Biasanya menampilkan jalur di template Anda sebagai string. ID dari level dipisahkan oleh garis miring, dimulai dari node segera di bawah App instance root, yang biasanya merupakan tumpukan. Misalnya, jalur dari dua sumber daya bucket Amazon S3 dalam contoh kode sebelumnya adalah `Stack1/MyBucket` dan. `Stack2/MyBucket`

Anda dapat mengakses jalur konstruksi apa pun secara terprogram, seperti yang ditunjukkan pada contoh berikut. Ini mendapatkan jalur `myConstruct` (ataumy_construct, seperti yang akan ditulis oleh pengembang Python). Karena ID harus unik dalam ruang lingkup mereka dibuat, jalur mereka selalu unik dalam AWS CDK aplikasi.

TypeScript

```
const path: string = myConstruct.node.path;
```

JavaScript

```
const path = myConstruct.node.path;
```

Python

```
path = my_construct.node.path
```

Java

```
String path = myConstruct.getNode().getPath();
```

C#

```
string path = myConstruct.Node.Path;
```

ID Unik

AWS CloudFormation mengharuskan semua ID logis dalam template menjadi unik. Karena itu, AWS CDK harus dapat menghasilkan pengidentifikasi unik untuk setiap konstruksi dalam aplikasi. Sumber daya memiliki jalur yang unik secara global (nama semua cakupan dari tumpukan ke sumber daya tertentu). Oleh karena itu, AWS CDK menghasilkan pengidentifikasi unik yang diperlukan dengan menggabungkan elemen jalur dan menambahkan hash 8 digit. (Hash diperlukan untuk membedakan jalur yang berbeda, seperti `A/B/C` dan`A/BC`, yang akan menghasilkan AWS CloudFormation pengidentifikasi yang sama. AWS CloudFormation pengidentifikasi alfanumerik dan tidak dapat berisi garis miring atau karakter pemisah lainnya.) AWS CDK Memanggil string ini ID unik dari konstruksi.

Secara umum, AWS CDK aplikasi Anda tidak perlu tahu tentang ID unik. Namun, Anda dapat mengakses ID unik dari konstruksi apa pun secara terprogram, seperti yang ditunjukkan pada contoh berikut.

TypeScript

```
const uid: string = Names.uniqueId(myConstruct);
```

JavaScript

```
const uid = Names.uniqueId(myConstruct);
```

Python

```
uid = Names.unique_id(my_construct)
```

Java

```
String uid = Names.uniqueId(myConstruct);
```

C#

```
string uid = Names.Uniqueid(myConstruct);
```

Alamat adalah jenis pengidentifikasi unik lain yang secara unik membedakan sumber daya CDK. Berasal dari hash SHA-1 jalan, itu tidak dapat dibaca manusia. Namun, panjangnya yang konstan dan relatif pendek (selalu 42 karakter heksadesimal) membuatnya berguna dalam situasi di mana ID unik “tradisional” mungkin terlalu panjang. Beberapa konstruksi mungkin menggunakan alamat dalam AWS CloudFormation template yang disintesis, bukan ID unik. Sekali lagi, aplikasi Anda umumnya tidak perlu tahu tentang alamat konstruksinya, tetapi Anda dapat mengambil alamat konstruksi sebagai berikut.

TypeScript

```
const addr: string = myConstruct.node.addr;
```

JavaScript

```
const addr = myConstruct.node.addr;
```

Python

```
addr = my_construct.node.addr
```

Java

```
String addr = myConstruct.getNode().getAddr();
```

C#

```
string addr = myConstruct.Node.Addr;
```

ID Logis

ID unik berfungsi sebagai pengidentifikasi logis (atau nama logis) sumber daya dalam AWS CloudFormation templat yang dihasilkan untuk konstruksi yang mewakili AWS sumber daya.

Misalnya, bucket Amazon S3 pada contoh sebelumnya yang dibuat dalam `Stack2` menghasilkan sumber daya. `AWS::S3::Bucket` ID logis sumber daya ada `Stack2MyBucket4DD88B4F` di AWS CloudFormation template yang dihasilkan. (Untuk detail tentang cara pengenal ini dihasilkan, lihatthe section called “ID Unik”.)

Stabilitas ID logis

Hindari mengubah ID logis sumber daya setelah dibuat. AWS CloudFormation mengidentifikasi sumber daya dengan ID logisnya. Oleh karena itu, jika Anda mengubah ID logis sumber daya, AWS CloudFormation membuat sumber daya baru dengan ID logis baru, lalu hapus yang sudah ada. Tergantung pada jenis sumber daya, ini dapat menyebabkan gangguan layanan, kehilangan data, atau keduanya.

Token

Token mewakili nilai yang hanya dapat diselesaikan di lain waktu dalam siklus hidup aplikasi. Misalnya, nama bucket Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) yang Anda tentukan di aplikasi CDK hanya dialokasikan saat template disintesis. AWS CloudFormation Jika Anda mencetak `bucket.bucketName` atribut, yang merupakan string, Anda akan melihat bahwa itu berisi sesuatu seperti berikut:

```
${TOKEN[Bucket.Name.1234]}
```

Ini adalah bagaimana AWS CDK mengkodekan token yang nilainya belum diketahui pada waktu konstruksi, tetapi akan tersedia nanti. AWS CDK Panggilan token placeholder ini. Dalam hal ini, ini adalah token yang dikodekan sebagai string.

Anda dapat melewatkan string ini seolah-olah itu adalah nama ember. Dalam contoh berikut, nama bucket ditentukan sebagai variabel lingkungan untuk AWS Lambda fungsi.

TypeScript

```
const bucket = new s3.Bucket(this, 'MyBucket');

const fn = new lambda.Function(stack, 'MyLambda', {
  // ...
  environment: {
    BUCKET_NAME: bucket.bucketName,
  }
});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'MyBucket');

const fn = new lambda.Function(stack, 'MyLambda', {
  // ...
  environment: {
    BUCKET_NAME: bucket.bucketName
  }
});
```

Python

```
bucket = s3.Bucket(self, "MyBucket")

fn = lambda_.Function(stack, "MyLambda",
        environment=dict(BUCKET_NAME=bucket.bucket_name))
```

Java

```
final Bucket bucket = new Bucket(this, "MyBucket");

Function fn = Function.Builder.create(this, "MyLambda")
```

```
        .environment(java.util.Map.of(    // Map.of requires Java 9+
            "BUCKET_NAME", bucket.getBucketName()))
        .build();
```

C#

```
var bucket = new s3.Bucket(this, "MyBucket");

var fn = new Function(this, "MyLambda", new FunctionProps {
    Environment = new Dictionary<string, string>
    {
        ["BUCKET_NAME"] = bucket.BucketName
    }
});
```

Ketika AWS CloudFormation template akhirnya disintesis, token dirender sebagai intrinsik. AWS CloudFormation `{ "Ref": "MyBucket" }` Pada waktu penerapan, AWS CloudFormation ganti intrinsik ini dengan nama sebenarnya dari bucket yang dibuat.

Topik

- Token dan pengkodean token
- Token yang dikodekan string
- Token yang dikodekan daftar
- Token yang dikodekan angka
- Nilai malas
- Konversi ke JSON

Token dan pengkodean token

Token adalah objek yang mengimplementasikan antarmuka IResolvable, yang berisi satu metode. `resolve` AWS CDK Memanggil metode ini selama sintesis untuk menghasilkan nilai akhir untuk AWS CloudFormation template. Token berpartisipasi dalam proses sintesis untuk menghasilkan nilai arbitrer dari jenis apa pun.

> **Note**
>
> Anda jarang akan bekerja secara langsung dengan `IResolvable` antarmuka. Anda kemungkinan besar hanya akan melihat versi token yang disandikan string.

Fungsi lain biasanya hanya menerima argumen tipe dasar, seperti `string` ataunumber. Untuk menggunakan token dalam kasus ini, Anda dapat menyandikannya menjadi salah satu dari tiga jenis dengan menggunakan metode statis pada kelas CDK.token.

- `Token.asString`untuk menghasilkan pengkodean string (atau memanggil `.toString()` objek token)
- `Token.asList`untuk menghasilkan pengkodean daftar
- `Token.asNumber`untuk menghasilkan pengkodean numerik

Ini mengambil nilai arbitrer, yang bisa berupa`IResolvable`, dan menyandikannya menjadi nilai primitif dari tipe yang ditunjukkan.

> **Important**
>
> Karena salah satu tipe sebelumnya berpotensi menjadi token yang dikodekan, berhati-hatilah saat Anda mengurai atau mencoba membaca isinya. Misalnya, jika Anda mencoba mengurai string untuk mengekstrak nilai darinya, dan string adalah token yang dikodekan, penguraian Anda gagal. Demikian pula, jika Anda mencoba menanyakan panjang array atau melakukan operasi matematika dengan angka, Anda harus terlebih dahulu memverifikasi bahwa itu bukan token yang dikodekan.

Untuk memeriksa apakah suatu nilai memiliki token yang belum terselesaikan di dalamnya, panggil metode `Token.isUnresolved` (Python`is_unresolved`:).

Contoh berikut memvalidasi bahwa nilai string, yang bisa berupa token, panjangnya tidak lebih dari 10 karakter.

TypeScript

```
if (!Token.isUnresolved(name) && name.length > 10) {
  throw new Error(`Maximum length for name is 10 characters`);
```

```
}
```

JavaScript

```
if ( !Token.isUnresolved(name) && name.length > 10) {
  throw ( new Error(`Maximum length for name is 10 characters`));
}
```

Python

```
if not Token.is_unresolved(name) and len(name) > 10:
    raise ValueError("Maximum length for name is 10 characters")
```

Java

```
if (!Token.isUnresolved(name) && name.length() > 10)
    throw new IllegalArgumentException("Maximum length for name is 10 characters");
```

C#

```
if (!Token.IsUnresolved(name) && name.Length > 10)
    throw new ArgumentException("Maximum length for name is 10 characters");
```

Jika nama adalah token, validasi tidak dilakukan, dan kesalahan masih dapat terjadi di tahap selanjutnya dalam siklus hidup, seperti selama penerapan.

> **Note**
>
> Anda dapat menggunakan pengkodean token untuk keluar dari sistem tipe. Misalnya, Anda dapat mengkodekan string token yang menghasilkan nilai angka pada waktu sintesis. Jika Anda menggunakan fungsi-fungsi ini, Anda bertanggung jawab untuk memastikan bahwa template Anda menyelesaikan ke status yang dapat digunakan setelah sintesis.

Token yang dikodekan string

Token yang disandikan string terlihat seperti berikut ini.

```
${TOKEN[Bucket.Name.1234]}
```

Mereka dapat diteruskan seperti string biasa, dan dapat digabungkan, seperti yang ditunjukkan pada contoh berikut.

TypeScript

```
const functionName = bucket.bucketName + 'Function';
```

JavaScript

```
const functionName = bucket.bucketName + 'Function';
```

Python

```
function_name = bucket.bucket_name + "Function"
```

Java

```
String functionName = bucket.getBucketName().concat("Function");
```

C#

```
string functionName = bucket.BucketName + "Function";
```

Anda juga dapat menggunakan interpolasi string, jika bahasa Anda mendukungnya, seperti yang ditunjukkan pada contoh berikut.

TypeScript

```
const functionName = `${bucket.bucketName}Function`;
```

JavaScript

```
const functionName = `${bucket.bucketName}Function`;
```

Python

```
function_name = f"{bucket.bucket_name}Function"
```

Java

```
String functionName = String.format("%sFunction". bucket.getBucketName());
```

C#

```
string functionName = $"${bucket.bucketName}Function";
```

Hindari memanipulasi string dengan cara lain. Misalnya, mengambil substring dari string kemungkinan akan merusak token string.

Token yang dikodekan daftar

Token yang dikodekan daftar terlihat seperti berikut:

```
["#{TOKEN[Stack.NotificationArns.1234]}"]
```

Satu-satunya hal yang aman untuk dilakukan dengan daftar ini adalah meneruskannya langsung ke konstruksi lain. Token dalam bentuk daftar string tidak dapat digabungkan, juga elemen tidak dapat diambil dari token. Satu-satunya cara aman untuk memanipulasinya adalah dengan menggunakan fungsi AWS CloudFormation intrinsik seperti fn.Select.

Token yang dikodekan angka

Token yang dikodekan angka adalah sekumpulan angka floating-point negatif kecil yang terlihat seperti berikut ini.

```
-1.8881545897087626e+289
```

Seperti halnya token daftar, Anda tidak dapat mengubah nilai angka, karena hal itu kemungkinan akan merusak token angka. Satu-satunya operasi yang diizinkan adalah meneruskan nilai ke konstruksi lain.

Nilai malas

Selain mewakili nilai deploy-time, seperti AWS CloudFormation parameter, token juga biasa digunakan untuk mewakili nilai malas waktu sintesis. Ini adalah nilai yang nilai akhirnya akan

ditentukan sebelum sintesis selesai, tetapi tidak pada titik di mana nilai dibangun. Gunakan token untuk meneruskan string literal atau nilai angka ke konstruksi lain, sedangkan nilai aktual pada waktu sintesis mungkin bergantung pada beberapa perhitungan yang belum terjadi.

Anda dapat membuat token yang mewakili nilai malas synth-time menggunakan metode statis pada `Lazy` kelas, seperti lazy.String dan Lazy.Number. Metode ini menerima objek yang `produce` propertinya adalah fungsi yang menerima argumen konteks dan mengembalikan nilai akhir ketika dipanggil.

Contoh berikut membuat grup Auto Scaling yang kapasitasnya ditentukan setelah pembuatannya.

TypeScript

```
let actualValue: number;

new AutoScalingGroup(this, 'Group', {
  desiredCapacity: Lazy.numberValue({
    produce(context) {
      return actualValue;
    }
  })
});

// At some later point
actualValue = 10;
```

JavaScript

```
let actualValue;

new AutoScalingGroup(this, 'Group', {
  desiredCapacity: Lazy.numberValue({
    produce(context) {
      return (actualValue);
    }
  })
});

// At some later point
actualValue = 10;
```

Python

```
class Producer:
    def __init__(self, func):
        self.produce = func

actual_value = None

AutoScalingGroup(self, "Group",
    desired_capacity=Lazy.number_value(Producer(lambda context: actual_value))
)

# At some later point
actual_value = 10
```

Java

```
double actualValue = 0;

class ProduceActualValue implements INumberProducer {

    @Override
    public Number produce(IResolveContext context) {
        return actualValue;
    }
}

AutoScalingGroup.Builder.create(this, "Group")
    .desiredCapacity(Lazy.numberValue(new ProduceActualValue())).build();

// At some later point
actualValue = 10;
```

C#

```
public class NumberProducer : INumberProducer
{
    Func<Double> function;

    public NumberProducer(Func<Double> function)
    {
        this.function = function;
    }
```

```
    public Double Produce(IResolveContext context)
    {
        return function();
    }
}

double actualValue = 0;

new AutoScalingGroup(this, "Group", new AutoScalingGroupProps
{
    DesiredCapacity = Lazy.NumberValue(new NumberProducer(() => actualValue))
});

// At some later point
actualValue = 10;
```

Konversi ke JSON

Terkadang Anda ingin menghasilkan string JSON dari data arbitrer, dan Anda mungkin tidak tahu apakah data tersebut berisi token. Untuk menyandikan JSON dengan benar struktur data apa pun, terlepas dari apakah itu berisi token, gunakan tumpukan metode. toJsonString, seperti yang ditunjukkan pada contoh berikut.

TypeScript

```
const stack = Stack.of(this);
const str = stack.toJsonString({
  value: bucket.bucketName
});
```

JavaScript

```
const stack = Stack.of(this);
const str = stack.toJsonString({
  value: bucket.bucketName
});
```

Python

```
stack = Stack.of(self)
```

```
string = stack.to_json_string(dict(value=bucket.bucket_name))
```

Java

```
Stack stack = Stack.of(this);
String stringVal = stack.toJsonString(java.util.Map.of(    // Map.of requires Java
 9+
        put("value", bucket.getBucketName())));
```

C#

```
var stack = Stack.Of(this);
var stringVal = stack.ToJsonString(new Dictionary<string, string>
{
    ["value"] = bucket.BucketName
});
```

Parameter-parameter

Parameter adalah nilai khusus yang disediakan pada waktu penerapan. Parameter adalah fitur dari AWS CloudFormation. Karena AWS CloudFormation template AWS Cloud Development Kit (AWS CDK) mensintesis, ia juga menawarkan dukungan untuk parameter waktu penerapan.

Topik

- Tentang parameter
- Mendefinisikan parameter
- Menggunakan parameter
- Menyebarkan dengan parameter

Tentang parameter

Dengan menggunakan AWS CDK, Anda dapat menentukan parameter, yang kemudian dapat digunakan dalam properti konstruksi yang Anda buat. Anda juga dapat menerapkan tumpukan yang berisi parameter.

Saat menerapkan AWS CloudFormation template menggunakan AWS CDK Toolkit, Anda memberikan nilai parameter pada baris perintah. Jika Anda menerapkan template melalui AWS CloudFormation konsol, Anda akan diminta untuk nilai parameter.

Secara umum, kami merekomendasikan untuk tidak menggunakan AWS CloudFormation parameter dengan AWS CDK. Cara biasa untuk meneruskan nilai ke dalam AWS CDK aplikasi adalah nilai konteks dan variabel lingkungan. Karena tidak tersedia pada waktu sintesis, nilai parameter tidak dapat dengan mudah digunakan untuk kontrol aliran dan tujuan lain di aplikasi CDK Anda.

> **Note**
>
> Untuk melakukan aliran kontrol dengan parameter, Anda dapat menggunakan `CfnCondition`konstruksi, meskipun ini canggung dibandingkan dengan pernyataan asli. `if`

Menggunakan parameter mengharuskan Anda untuk memperhatikan bagaimana kode yang Anda tulis berperilaku pada waktu penerapan, dan juga pada waktu sintesis. Hal ini membuat lebih sulit untuk memahami dan bernalar tentang AWS CDK aplikasi Anda, dalam banyak kasus untuk sedikit manfaat.

Secara umum, lebih baik meminta aplikasi CDK Anda menerima informasi yang diperlukan dengan cara yang terdefinisi dengan baik dan menggunakannya langsung untuk mendeklarasikan konstruksi di aplikasi CDK Anda. AWS CloudFormation Template AWS CDK yang dihasilkan ideal adalah konkret, tanpa nilai yang tersisa untuk ditentukan pada waktu penerapan.

Namun, ada kasus penggunaan yang AWS CloudFormation parameternya cocok secara unik. Jika Anda memiliki tim terpisah yang mendefinisikan dan menerapkan infrastruktur, misalnya, Anda dapat menggunakan parameter untuk membuat templat yang dihasilkan lebih berguna secara luas. Juga, karena AWS CDK mendukung AWS CloudFormation parameter, Anda dapat menggunakan AWS CDK dengan AWS layanan yang menggunakan AWS CloudFormation template (seperti Service Catalog). AWS Layanan ini menggunakan parameter untuk mengonfigurasi template yang sedang digunakan.

Mendefinisikan parameter

Gunakan `CfnParameter`kelas untuk menentukan parameter. Anda akan ingin menentukan setidaknya jenis dan deskripsi untuk sebagian besar parameter, meskipun keduanya secara teknis opsional. Deskripsi muncul ketika pengguna diminta untuk memasukkan nilai parameter di AWS CloudFormation konsol. Untuk informasi selengkapnya tentang jenis yang tersedia, lihat Jenis.

> **Note**
>
> Anda dapat menentukan parameter dalam lingkup apa pun. Namun, kami merekomendasikan untuk menentukan parameter pada tingkat tumpukan sehingga ID logisnya tidak berubah saat Anda memfaktorkan ulang kode Anda.

TypeScript

```
const uploadBucketName = new CfnParameter(this, "uploadBucketName", {
  type: "String",
  description: "The name of the Amazon S3 bucket where uploaded files will be
 stored."});
```

JavaScript

```
const uploadBucketName = new CfnParameter(this, "uploadBucketName", {
  type: "String",
  description: "The name of the Amazon S3 bucket where uploaded files will be
 stored."});
```

Python

```
upload_bucket_name = CfnParameter(self, "uploadBucketName", type="String",
    description="The name of the Amazon S3 bucket where uploaded files will be
 stored.")
```

Java

```
CfnParameter uploadBucketName = CfnParameter.Builder.create(this,
 "uploadBucketName")
        .type("String")
        .description("The name of the Amazon S3 bucket where uploaded files will be
 stored")
        .build();
```

C#

```
var uploadBucketName = new CfnParameter(this, "uploadBucketName", new
 CfnParameterProps
```

```
{
    Type = "String",
    Description = "The name of the Amazon S3 bucket where uploaded files will be
 stored"
});
```

Menggunakan parameter

Sebuah `CfnParameter` instance mengekspos nilainya ke AWS CDK aplikasi Anda melalui token. Seperti semua token, token parameter diselesaikan pada waktu sintesis. Tapi itu menyelesaikan referensi ke parameter yang ditentukan dalam AWS CloudFormation template (yang akan diselesaikan pada waktu penerapan), bukan ke nilai konkret.

Anda dapat mengambil token sebagai contoh dari `Token` kelas, atau dalam string, daftar string, atau pengkodean numerik. Pilihan Anda tergantung pada jenis nilai yang dibutuhkan oleh kelas atau metode yang ingin Anda gunakan parameternya.

TypeScript

Property	kind of value
`value`	`Token` class instance
`valueAsList`	The token represented as a string list
`valueAsNumber`	The token represented as a number
`valueAsString`	The token represented as a string

JavaScript

Property	kind of value
`value`	`Token` class instance
`valueAsList`	The token represented as a string list
`valueAsNumber`	The token represented as a number

Property	kind of value
`valueAsString`	The token represented as a string

Python

Property	kind of value
`value`	`Token` class instance
`value_as_list`	The token represented as a string list
`value_as_number`	The token represented as a number
`value_as_string`	The token represented as a string

Java

Property	kind of value
`getValue ()`	`Token` class instance
`getValueAsDaftar ()`	The token represented as a string list
`getValueAsNomor ()`	The token represented as a number
`getValueAsTali ()`	The token represented as a string

C#

Property	kind of value
`Nilai`	`Token` class instance
`ValueAsList`	The token represented as a string list
`ValueAsNumber`	The token represented as a number

Property	kind of value
`ValueAsString`	The token represented as a string

Misalnya, untuk menggunakan parameter dalam `Bucket` definisi:

TypeScript

```
const bucket = new Bucket(this, "myBucket",
  { bucketName: uploadBucketName.valueAsString});
```

JavaScript

```
const bucket = new Bucket(this, "myBucket",
  { bucketName: uploadBucketName.valueAsString});
```

Python

```
bucket = Bucket(self, "myBucket",
    bucket_name=upload_bucket_name.value_as_string)
```

Java

```
Bucket bucket = Bucket.Builder.create(this, "myBucket")
        .bucketName(uploadBucketName.getValueAsString())
        .build();
```

C#

```
var bucket = new Bucket(this, "myBucket")
{
    BucketName = uploadBucketName.ValueAsString
};
```

Menyebarkan dengan parameter

Template yang dihasilkan yang berisi parameter dapat digunakan dengan cara biasa melalui AWS CloudFormation konsol. Anda diminta untuk nilai setiap parameter.

AWS CDK Toolkit (alat baris `cdk` perintah) juga mendukung menentukan parameter pada penerapan. Anda memberikan ini pada baris perintah mengikuti `--parameters` bendera. Anda dapat menerapkan tumpukan yang menggunakan `uploadBucketName` parameter, seperti contoh berikut.

```
cdk deploy MyStack --parameters uploadBucketName=uploadbucket
```

Untuk menentukan beberapa parameter, gunakan beberapa `--parameters` bendera.

```
cdk deploy MyStack --parameters uploadBucketName=upbucket --parameters
 downloadBucketName=downbucket
```

Jika Anda menerapkan beberapa tumpukan, Anda dapat menentukan nilai yang berbeda dari setiap parameter untuk setiap tumpukan. Untuk melakukannya, awali nama parameter dengan nama tumpukan dan titik dua.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=uploadbucket --
parameters YourStack:uploadBucketName=upbucket
```

Secara default, AWS CDK mempertahankan nilai parameter dari penerapan sebelumnya dan menggunakannya dalam penerapan berikutnya jika tidak ditentukan secara eksplisit. Gunakan `--no-previous-parameters` bendera untuk meminta semua parameter ditentukan.

Pemberian tag

Tag adalah elemen nilai kunci informasi yang dapat Anda tambahkan ke konstruksi di aplikasi Anda. AWS CDK Tag yang diterapkan pada konstruksi tertentu juga berlaku untuk semua anak yang dapat diberi tag. Tag disertakan dalam AWS CloudFormation templat yang disintesis dari aplikasi Anda dan diterapkan ke AWS sumber daya yang diterapkan. Anda dapat menggunakan tag untuk mengidentifikasi dan mengkategorikan sumber daya untuk tujuan berikut:

- Menyederhanakan manajemen
- Alokasi biaya
- Kontrol akses
- Tujuan lain yang Anda rancang

> **Tip**
>
> Untuk informasi selengkapnya tentang cara menggunakan tag dengan AWS sumber daya, lihat Praktik Terbaik untuk Menandai AWS Sumber Daya di AWS Whitepaper.

Topik

Menggunakan tanda

`Tags`Kelas mencakup metode statis`of()`, di mana Anda dapat menambahkan tag ke, atau menghapus tag dari, konstruksi yang ditentukan.

- `Tags.of(SCOPE).add()`menerapkan tag baru ke konstruksi yang diberikan dan semua anak-anaknya.
- `Tags.of(SCOPE).remove()`menghapus tag dari konstruksi yang diberikan dan salah satu turunannya, termasuk tag yang mungkin diterapkan oleh konstruksi anak pada dirinya sendiri.

> **Note**
>
> Penandaan diimplementasikan menggunakanthe section called “Aspek”. Aspek adalah cara untuk menerapkan operasi (seperti penandaan) ke semua konstruksi dalam lingkup tertentu.

Contoh berikut menerapkan kunci tag dengan nilai nilai untuk konstruksi.

TypeScript

```
Tags.of(myConstruct).add('key', 'value');
```

JavaScript

```
Tags.of(myConstruct).add('key', 'value');
```

Python

```
Tags.of(my_construct).add("key", "value")
```

Java

```
Tags.of(myConstruct).add("key", "value");
```

C#

```
Tags.Of(myConstruct).Add("key", "value");
```

Contoh berikut menghapus kunci tag dari konstruksi.

TypeScript

```
Tags.of(myConstruct).remove('key');
```

JavaScript

```
Tags.of(myConstruct).remove('key');
```

Python

```
Tags.of(my_construct).remove("key")
```

Java

```
Tags.of(myConstruct).remove("key");
```

C#

```
Tags.Of(myConstruct).Remove("key");
```

Jika Anda menggunakan `Stage` konstruksi, terapkan tag di `Stage` tingkat atau di bawah. Tag tidak diterapkan melintasi `Stage` batas.

Menandai prioritas

AWS CDK Ini berlaku dan menghapus tag secara rekursif. Jika ada konflik, operasi penandaan dengan prioritas tertinggi menang. (Prioritas ditetapkan menggunakan `priority` properti opsional.) Jika prioritas dua operasi sama, operasi penandaan yang paling dekat dengan bagian bawah pohon konstruksi menang. Secara default, menerapkan tag memiliki prioritas 100 (kecuali untuk tag yang ditambahkan langsung ke AWS CloudFormation sumber daya, yang memiliki prioritas 50). Prioritas default untuk menghapus tag adalah 200.

Berikut ini menerapkan tag dengan prioritas 300 untuk konstruksi.

TypeScript

```
Tags.of(myConstruct).add('key', 'value', {
  priority: 300
});
```

JavaScript

```
Tags.of(myConstruct).add('key', 'value', {
  priority: 300
});
```

Python

```
Tags.of(my_construct).add("key", "value", priority=300)
```

Java

```
Tags.of(myConstruct).add("key", "value", TagProps.builder()
        .priority(300).build());
```

C#

```
Tags.Of(myConstruct).Add("key", "value", new TagProps { Priority = 300 });
```

Properti opsional

Tag mendukung properties yang menyempurnakan cara tag diterapkan, atau dihapus dari, sumber daya. Semua properti adalah opsional.

`applyToLaunchedInstances`(Python:) `apply_to_launched_instances`

Tersedia untuk add () saja. Secara default, tag diterapkan ke instance yang diluncurkan di grup Auto Scaling. Setel properti ini ke false untuk mengabaikan instance yang diluncurkan dalam grup Auto Scaling.

`includeResourceTypes`/`excludeResourceTypes`(Python:`include_resource_types`/) `exclude_resource_types`

Gunakan ini untuk memanipulasi tag hanya pada subset sumber daya, berdasarkan jenis AWS CloudFormation sumber daya. Secara default, operasi diterapkan ke semua sumber daya di subpohon konstruksi, tetapi ini dapat diubah dengan memasukkan atau mengecualikan jenis sumber daya tertentu. Kecualikan lebih diutamakan daripada include, jika keduanya ditentukan.

`priority`

Gunakan ini untuk menetapkan prioritas operasi ini sehubungan dengan operasi lain `Tags.add()` dan `Tags.remove()` operasi. Nilai yang lebih tinggi lebih diutamakan daripada nilai yang lebih rendah. Defaultnya adalah 100 untuk operasi tambah (50 untuk tag diterapkan langsung ke AWS CloudFormation sumber daya) dan 200 untuk menghapus operasi.

Contoh berikut menerapkan tag tag dengan nilai nilai dan prioritas 100 untuk sumber daya tipe AWS::Xxx::Yyydalam konstruksi. Itu tidak menerapkan tag ke instance yang diluncurkan di grup Auto Scaling Amazon EC2 atau ke sumber daya jenis. AWS::Xxx::Zzz (Ini adalah placeholder untuk dua jenis AWS CloudFormation sumber daya yang sewenang-wenang tetapi berbeda.)

TypeScript

```
Tags.of(myConstruct).add('tagname', 'value', {
  applyToLaunchedInstances: false,
  includeResourceTypes: ['AWS::Xxx::Yyy'],
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 100,
});
```

JavaScript

```
Tags.of(myConstruct).add('tagname', 'value', {
  applyToLaunchedInstances: false,
  includeResourceTypes: ['AWS::Xxx::Yyy'],
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 100
});
```

Python

```
Tags.of(my_construct).add("tagname", "value",
    apply_to_launched_instances=False,
    include_resource_types=["AWS::Xxx::Yyy"],
    exclude_resource_types=["AWS::Xxx::Zzz"],
    priority=100)
```

Java

```
Tags.of(myConstruct).add("key", "value", TagProps.builder()
                .applyToLaunchedInstances(false)
                .includeResourceTypes(Arrays.asList("AWS::Xxx::Yyy"))
                .excludeResourceTypes(Arrays.asList("AWS::Xxx::Zzz"))
                .priority(100).build());
```

C#

```
Tags.Of(myConstruct).Add("tagname", "value", new TagProps
{
    ApplyToLaunchedInstances = false,
    IncludeResourceTypes = ["AWS::Xxx::Yyy"],
    ExcludeResourceTypes = ["AWS::Xxx::Zzz"],
    Priority = 100
});
```

Contoh berikut menghapus tag tag dengan prioritas 200 dari sumber daya tipe AWS::Xxx::Yyydalam konstruksi, tetapi tidak dari sumber daya tipe. AWS::Xxx::Zzz

TypeScript

```
Tags.of(myConstruct).remove('tagname', {
```

```
  includeResourceTypes: ['AWS::Xxx::Yyy'],
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 200,
});
```

JavaScript

```
Tags.of(myConstruct).remove('tagname', {
  includeResourceTypes: ['AWS::Xxx::Yyy'],
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 200
});
```

Python

```
Tags.of(my_construct).remove("tagname",
    include_resource_types=["AWS::Xxx::Yyy"],
    exclude_resource_types=["AWS::Xxx::Zzz"],
    priority=200,)
```

Java

```
Tags.of((myConstruct).remove("tagname", TagProps.builder()
        .includeResourceTypes(Arrays.asList("AWS::Xxx::Yyy"))
        .excludeResourceTypes(Arrays.asList("AWS::Xxx::Zzz"))
        .priority(100).build());
```

C#

```
Tags.Of(myConstruct).Remove("tagname", new TagProps
{
    IncludeResourceTypes = ["AWS::Xxx::Yyy"],
    ExcludeResourceTypes = ["AWS::Xxx::Zzz"],
    Priority = 100
});
```

Contoh

Contoh berikut menambahkan kunci tag StackTypedengan nilai TheBestke sumber daya apa pun yang dibuat dalam `Stack` nama`MarketingSystem`. Kemudian menghapusnya lagi dari semua

sumber daya kecuali subnet VPC Amazon EC2. Hasilnya adalah hanya subnet yang memiliki tag yang diterapkan.

TypeScript

```
import { App, Stack, Tags } from 'aws-cdk-lib';

const app = new App();
const theBestStack = new Stack(app, 'MarketingSystem');

// Add a tag to all constructs in the stack
Tags.of(theBestStack).add('StackType', 'TheBest');

// Remove the tag from all resources except subnet resources
Tags.of(theBestStack).remove('StackType', {
  excludeResourceTypes: ['AWS::EC2::Subnet']
});
```

JavaScript

```
const { App, Stack, Tags } = require('aws-cdk-lib');

const app = new App();
const theBestStack = new Stack(app, 'MarketingSystem');

// Add a tag to all constructs in the stack
Tags.of(theBestStack).add('StackType', 'TheBest');

// Remove the tag from all resources except subnet resources
Tags.of(theBestStack).remove('StackType', {
  excludeResourceTypes: ['AWS::EC2::Subnet']
});
```

Python

```
from aws_cdk import App, Stack, Tags

app = App();
the_best_stack = Stack(app, 'MarketingSystem')

# Add a tag to all constructs in the stack
Tags.of(the_best_stack).add("StackType", "TheBest")
```

```
# Remove the tag from all resources except subnet resources
Tags.of(the_best_stack).remove("StackType",
    exclude_resource_types=["AWS::EC2::Subnet"])
```

Java

```
import software.amazon.awscdk.App;
import software.amazon.awscdk.Tags;

// Add a tag to all constructs in the stack
Tags.of(theBestStack).add("StackType", "TheBest");

// Remove the tag from all resources except subnet resources
Tags.of(theBestStack).remove("StackType", TagProps.builder()
        .excludeResourceTypes(Arrays.asList("AWS::EC2::Subnet"))
        .build());
```

C#

```
using Amazon.CDK;

var app = new App();
var theBestStack = new Stack(app, 'MarketingSystem');

// Add a tag to all constructs in the stack
Tags.Of(theBestStack).Add("StackType", "TheBest");

// Remove the tag from all resources except subnet resources
Tags.Of(theBestStack).Remove("StackType", new TagProps
{
    ExcludeResourceTypes = ["AWS::EC2::Subnet"]
});
```

Kode berikut mencapai hasil yang sama. Pertimbangkan pendekatan mana (inklusi atau pengecualian) yang membuat maksud Anda lebih jelas.

TypeScript

```
Tags.of(theBestStack).add('StackType', 'TheBest',
  { includeResourceTypes: ['AWS::EC2::Subnet']});
```

JavaScript

```
Tags.of(theBestStack).add('StackType', 'TheBest',
  { includeResourceTypes: ['AWS::EC2::Subnet']});
```

Python

```
Tags.of(the_best_stack).add("StackType", "TheBest",
    include_resource_types=["AWS::EC2::Subnet"])
```

Java

```
Tags.of(theBestStack).add("StackType", "TheBest", TagProps.builder()
        .includeResourceTypes(Arrays.asList("AWS::EC2::Subnet"))
        .build());
```

C#

```
Tags.Of(theBestStack).Add("StackType", "TheBest", new TagProps {
    IncludeResourceTypes = ["AWS::EC2::Subnet"]
});
```

Menandai konstruksi tunggal

`Tags.of(scope).add(key, value)`adalah cara standar untuk menambahkan tag ke konstruksi di. AWS CDK Perilaku berjalan di pohon, yang secara rekursif menandai semua sumber daya yang dapat diberi tag di bawah lingkup yang diberikan, hampir selalu seperti yang Anda inginkan. Namun, terkadang, Anda perlu menandai konstruksi (atau konstruksi) tertentu yang sewenang-wenang.

Salah satu kasus tersebut melibatkan penerapan tag yang nilainya berasal dari beberapa properti konstruksi yang diberi tag. Pendekatan penandaan standar secara rekursif menerapkan kunci dan nilai yang sama ke semua sumber daya yang cocok dalam ruang lingkup. Namun, di sini nilainya bisa berbeda untuk setiap konstruksi yang ditandai.

Tag diimplementasikan menggunakan aspek, dan CDK memanggil `visit()` metode tag untuk setiap konstruksi di bawah lingkup yang Anda tentukan menggunakan. `Tags.of(scope)` Kita dapat memanggil `Tag.visit()` langsung untuk menerapkan tag ke konstruksi tunggal.

TypeScript

```
new cdk.Tag(key, value).visit(scope);
```

JavaScript

```
new cdk.Tag(key, value).visit(scope);
```

Python

```
cdk.Tag(key, value).visit(scope)
```

Java

```
Tag.Builder.create(key, value).build().visit(scope);
```

C#

```
new Tag(key, value).Visit(scope);
```

Anda dapat menandai semua konstruksi di bawah lingkup tetapi membiarkan nilai tag berasal dari properti setiap konstruksi. Untuk melakukannya, tulis aspek dan terapkan tag dalam `visit()` metode aspek seperti yang ditunjukkan pada contoh sebelumnya. Kemudian, tambahkan aspek ke lingkup yang diinginkan menggunakan`Aspects.of(scope).add(aspect)`.

Contoh berikut menerapkan tag untuk setiap sumber daya dalam tumpukan yang berisi jalur sumber daya.

TypeScript

```
class PathTagger implements cdk.IAspect {
  visit(node: IConstruct) {
    new cdk.Tag("aws-cdk-path", node.node.path).visit(node);
  }
}

stack = new MyStack(app);
cdk.Aspects.of(stack).add(new PathTagger())
```

JavaScript

```
class PathTagger {
  visit(node) {
    new cdk.Tag("aws-cdk-path", node.node.path).visit(node);
  }
}

stack = new MyStack(app);
cdk.Aspects.of(stack).add(new PathTagger())
```

Python

```
@jsii.implements(cdk.IAspect)
class PathTagger:
    def visit(self, node: IConstruct):
        cdk.Tag("aws-cdk-path", node.node.path).visit(node)

stack = MyStack(app)
cdk.Aspects.of(stack).add(PathTagger())
```

Java

```
final class PathTagger implements IAspect {
 public void visit(IConstruct node) {
  Tag.Builder.create("aws-cdk-path", node.getNode().getPath()).build().visit(node);
 }
}

stack stack = new MyStack(app);
Aspects.of(stack).add(new PathTagger());
```

C#

```
public class PathTagger : IAspect
{
    public void Visit(IConstruct node)
    {
        new Tag("aws-cdk-path", node.Node.Path).Visit(node);
    }
}
```

```
var stack = new MyStack(app);
Aspects.Of(stack).Add(new PathTagger);
```

> **Tip**
>
> Logika penandaan bersyarat, termasuk prioritas, jenis sumber daya, dan sebagainya, dibangun ke dalam kelas. `Tag` Anda dapat menggunakan fitur ini saat menerapkan tag ke sumber daya arbitrer; tag tidak diterapkan jika kondisi tidak terpenuhi. Selain itu, `Tag` kelas hanya menandai sumber daya yang dapat diberi tag, jadi Anda tidak perlu menguji apakah konstruksi dapat diberi tag sebelum menerapkan tag.

Aset

Aset adalah file lokal, direktori, atau gambar Docker yang dapat dibundel ke dalam AWS CDK pustaka dan aplikasi. Misalnya, aset mungkin merupakan direktori yang berisi kode handler untuk suatu AWS Lambda fungsi. Aset dapat mewakili artefak apa pun yang dibutuhkan aplikasi untuk beroperasi.

Video tutorial berikut memberikan gambaran menyeluruh tentang aset CDK, dan menjelaskan bagaimana Anda dapat menggunakannya dalam insfrastructure as code (IAc) Anda.

Aset CDK Dijelaskan

Anda menambahkan aset melalui API yang diekspos oleh AWS konstruksi tertentu. Misalnya, saat Anda mendefinisikan konstruksi Lambda.function, properti code memungkinkan Anda meneruskan aset (direktori). `Function`menggunakan aset untuk menggabungkan isi direktori dan menggunakannya untuk kode fungsi. Demikian pula, ecs. ContainerImage.fromAsset menggunakan image Docker yang dibangun dari direktori lokal saat mendefinisikan definisi tugas Amazon ECS.

Aset secara detail

Saat Anda merujuk ke aset di aplikasi Anda, rakitan cloud yang disintesis dari aplikasi Anda menyertakan informasi metadata dengan instruksi untuk CLI. AWS CDK Instruksi termasuk di mana menemukan aset pada disk lokal dan jenis bundling apa yang harus dilakukan berdasarkan jenis aset, seperti direktori untuk dikompres (zip) atau gambar Docker yang akan dibuat.

Ini AWS CDK menghasilkan hash sumber untuk aset. Ini dapat digunakan pada waktu konstruksi untuk menentukan apakah isi aset telah berubah.

Secara default, AWS CDK membuat salinan aset di direktori perakitan cloud, yang defaultnya`cdk.out`, di bawah hash sumber. Dengan cara ini, perakitan cloud mandiri, jadi jika dipindahkan ke host yang berbeda untuk penerapan, itu masih dapat digunakan. Lihat the section called “Rakitan awan” untuk detail.

Saat AWS CDK menerapkan aplikasi yang mereferensikan aset (baik secara langsung dengan kode aplikasi atau melalui pustaka), AWS CDK CLI terlebih dahulu menyiapkan dan menerbitkan aset tersebut ke bucket Amazon S3 atau repositori Amazon ECR. (Bucket atau repositori S3 dibuat selama bootstrap.) Hanya dengan begitu sumber daya yang ditentukan dalam tumpukan digunakan.

Bagian ini menjelaskan API tingkat rendah yang tersedia dalam kerangka kerja.

Jenis aset

AWS CDK Mendukung jenis aset berikut:

Aset Amazon S3

Ini adalah file dan direktori lokal yang AWS CDK diunggah ke Amazon S3.

Gambar Docker

Ini adalah gambar Docker yang AWS CDK diunggah ke Amazon ECR.

Jenis aset ini dijelaskan di bagian berikut.

Aset Amazon S3

Anda dapat menentukan file dan direktori lokal sebagai aset, dan AWS CDK paket dan mengunggahnya ke Amazon S3 melalui modul aws-s3-assets.

Contoh berikut mendefinisikan aset direktori lokal dan aset file.

TypeScript

```
import { Asset } from 'aws-cdk-lib/aws-s3-assets';

// Archived and uploaded to Amazon S3 as a .zip file
const directoryAsset = new Asset(this, "SampleZippedDirAsset", {
```

```
  path: path.join(__dirname, "sample-asset-directory")
});

// Uploaded to Amazon S3 as-is
const fileAsset = new Asset(this, 'SampleSingleFileAsset', {
  path: path.join(__dirname, 'file-asset.txt')
});
```

JavaScript

```
const { Asset } = require('aws-cdk-lib/aws-s3-assets');

// Archived and uploaded to Amazon S3 as a .zip file
const directoryAsset = new Asset(this, "SampleZippedDirAsset", {
  path: path.join(__dirname, "sample-asset-directory")
});

// Uploaded to Amazon S3 as-is
const fileAsset = new Asset(this, 'SampleSingleFileAsset', {
  path: path.join(__dirname, 'file-asset.txt')
});
```

Python

```
import os.path
dirname = os.path.dirname(__file__)

from aws_cdk.aws_s3_assets import Asset

# Archived and uploaded to Amazon S3 as a .zip file
directory_asset = Asset(self, "SampleZippedDirAsset",
  path=os.path.join(dirname, "sample-asset-directory")
)

# Uploaded to Amazon S3 as-is
file_asset = Asset(self, 'SampleSingleFileAsset',
  path=os.path.join(dirname, 'file-asset.txt')
)
```

Java

```
import java.io.File;
```

```
import software.amazon.awscdk.services.s3.assets.Asset;

// Directory where app was started
File startDir = new File(System.getProperty("user.dir"));

// Archived and uploaded to Amazon S3 as a .zip file
Asset directoryAsset = Asset.Builder.create(this, "SampleZippedDirAsset")
                .path(new File(startDir, "sample-asset-
directory").toString()).build();

// Uploaded to Amazon S3 as-is
Asset fileAsset = Asset.Builder.create(this, "SampleSingleFileAsset")
                .path(new File(startDir, "file-asset.txt").toString()).build();
```

C#

```
using System.IO;
using Amazon.CDK.AWS.S3.Assets;

// Archived and uploaded to Amazon S3 as a .zip file
var directoryAsset = new Asset(this, "SampleZippedDirAsset", new AssetProps
{
    Path = Path.Combine(Directory.GetCurrentDirectory(), "sample-asset-directory")
});

// Uploaded to Amazon S3 as-is
var fileAsset = new Asset(this, "SampleSingleFileAsset", new AssetProps
{
    Path = Path.Combine(Directory.GetCurrentDirectory(), "file-asset.txt")
});
```

Go

```
dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

awss3assets.NewAsset(stack, jsii.String("SampleZippedDirAsset"),
 &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "sample-asset-directory")),
})
```

```
awss3assets.NewAsset(stack, jsii.String("SampleSingleFileAsset"),
 &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "file-asset.txt")),
})
```

Dalam kebanyakan kasus, Anda tidak perlu langsung menggunakan API dalam `aws-s3-assets` modul. Modul yang mendukung aset, sepertiaws-lambda, memiliki metode kemudahan sehingga Anda dapat menggunakan aset. Untuk fungsi Lambda, metode statis fromAsset () memungkinkan Anda menentukan direktori atau file.zip di sistem file lokal.

Contoh fungsi Lambda

Kasus penggunaan umum adalah membuat fungsi Lambda dengan kode handler sebagai aset Amazon S3.

Contoh berikut menggunakan aset Amazon S3 untuk menentukan handler Python di direktori lokal. `handler` Ini juga menciptakan fungsi Lambda dengan aset direktori lokal sebagai properti. `code` Berikut ini adalah kode Python untuk handler.

```
def lambda_handler(event, context):
  message = 'Hello World!'
  return {
    'message': message
  }
```

Kode untuk AWS CDK aplikasi utama akan terlihat seperti berikut.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { Constructs } from 'constructs';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as path from 'path';

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

    new lambda.Function(this, 'myLambdaFunction', {
      code: lambda.Code.fromAsset(path.join(__dirname, 'handler')),
```

```
      runtime: lambda.Runtime.PYTHON_3_6,
      handler: 'index.lambda_handler'
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const lambda = require('aws-cdk-lib/aws-lambda');
const path = require('path');

class HelloAssetStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    new lambda.Function(this, 'myLambdaFunction', {
      code: lambda.Code.fromAsset(path.join(__dirname, 'handler')),
      runtime: lambda.Runtime.PYTHON_3_6,
      handler: 'index.lambda_handler'
    });
  }
}

module.exports = { HelloAssetStack }
```

Python

```
from aws_cdk import Stack
from constructs import Construct
from aws_cdk import aws_lambda as lambda_

import os.path
dirname = os.path.dirname(__file__)

class HelloAssetStack(Stack):
    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        lambda_.Function(self, 'myLambdaFunction',
            code=lambda_.Code.from_asset(os.path.join(dirname, 'handler')),
            runtime=lambda_.Runtime.PYTHON_3_6,
            handler="index.lambda_handler")
```

Java

```
import java.io.File;

import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;

public class HelloAssetStack extends Stack {

    public HelloAssetStack(final App scope, final String id) {
        this(scope, id, null);
    }

    public HelloAssetStack(final App scope, final String id, final StackProps props)
 {
        super(scope, id, props);

        File startDir = new File(System.getProperty("user.dir"));

        Function.Builder.create(this, "myLambdaFunction")
                .code(Code.fromAsset(new File(startDir, "handler").toString()))
                .runtime(Runtime.PYTHON_3_6)
                .handler("index.lambda_handler").build();
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using System.IO;

public class HelloAssetStack : Stack
{
    public HelloAssetStack(Construct scope, string id, StackProps props) :
 base(scope, id, props)
    {
        new Function(this, "myLambdaFunction", new FunctionProps
        {
            Code = Code.FromAsset(Path.Combine(Directory.GetCurrentDirectory(),
 "handler")),
```

```
            Runtime = Runtime.PYTHON_3_6,
            Handler = "index.lambda_handler"
        });
    }
}
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awslambda"
  "github.com/aws/aws-cdk-go/awscdk/v2/awss3assets"
  "github.com/aws/constructs-go/constructs/v10"
  "github.com/aws/jsii-runtime-go"
)

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

  dirName, err := os.Getwd()
  if err != nil {
    panic(err)
  }

  awslambda.NewFunction(stack, jsii.String("myLambdaFunction"),
 &awslambda.FunctionProps{
    Code: awslambda.AssetCode_FromAsset(jsii.String(path.Join(dirName, "handler")),
 &awss3assets.AssetOptions{}),
    Runtime: awslambda.Runtime_PYTHON_3_6(),
    Handler: jsii.String("index.lambda_handler"),
  })

  return stack
}
```

`Function`Metode ini menggunakan aset untuk menggabungkan isi direktori dan menggunakannya untuk kode fungsi.

> **Tip**
>
> `.jar`File Java adalah file ZIP dengan ekstensi berbeda. Ini diunggah apa adanya ke Amazon S3, tetapi ketika digunakan sebagai fungsi Lambda, file yang dikandungnya diekstraksi, yang mungkin tidak Anda inginkan. Untuk menghindari hal ini, tempatkan `.jar` file dalam direktori dan tentukan direktori itu sebagai aset.

Contoh atribut deploy-time

Jenis aset Amazon S3 juga mengekspos atribut waktu penerapan yang dapat direferensikan di pustaka dan aplikasi. AWS CDK Perintah AWS CDK CLI cdk synth menampilkan properti aset sebagai AWS CloudFormation parameter.

Contoh berikut menggunakan atribut deploy-time untuk meneruskan lokasi aset gambar ke dalam fungsi Lambda sebagai variabel lingkungan. (Jenis file tidak masalah; gambar PNG yang digunakan di sini hanyalah sebuah contoh.)

TypeScript

```
import { Asset } from 'aws-cdk-lib/aws-s3-assets';
import * as path from 'path';

const imageAsset = new Asset(this, "SampleAsset", {
  path: path.join(__dirname, "images/my-image.png")
});

new lambda.Function(this, "myLambdaFunction", {
  code: lambda.Code.asset(path.join(__dirname, "handler")),
  runtime: lambda.Runtime.PYTHON_3_6,
  handler: "index.lambda_handler",
  environment: {
    'S3_BUCKET_NAME': imageAsset.s3BucketName,
    'S3_OBJECT_KEY': imageAsset.s3ObjectKey,
    'S3_OBJECT_URL': imageAsset.s3ObjectUrl
  }
});
```

JavaScript

```
const { Asset } = require('aws-cdk-lib/aws-s3-assets');
const path = require('path');

const imageAsset = new Asset(this, "SampleAsset", {
  path: path.join(__dirname, "images/my-image.png")
});

new lambda.Function(this, "myLambdaFunction", {
  code: lambda.Code.asset(path.join(__dirname, "handler")),
  runtime: lambda.Runtime.PYTHON_3_6,
  handler: "index.lambda_handler",
  environment: {
    'S3_BUCKET_NAME': imageAsset.s3BucketName,
    'S3_OBJECT_KEY': imageAsset.s3ObjectKey,
    'S3_OBJECT_URL': imageAsset.s3ObjectUrl
  }
});
```

Python

```
import os.path

import aws_cdk.aws_lambda as lambda_
from aws_cdk.aws_s3_assets import Asset

dirname = os.path.dirname(__file__)

image_asset = Asset(self, "SampleAsset",
    path=os.path.join(dirname, "images/my-image.png"))

lambda_.Function(self, "myLambdaFunction",
    code=lambda_.Code.asset(os.path.join(dirname, "handler")),
    runtime=lambda_.Runtime.PYTHON_3_6,
    handler="index.lambda_handler",
    environment=dict(
        S3_BUCKET_NAME=image_asset.s3_bucket_name,
        S3_OBJECT_KEY=image_asset.s3_object_key,
        S3_OBJECT_URL=image_asset.s3_object_url))
```

Java

```
import java.io.File;

import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.s3.assets.Asset;

public class FunctionStack extends Stack {
    public FunctionStack(final App scope, final String id, final StackProps props) {
        super(scope, id, props);

        File startDir = new File(System.getProperty("user.dir"));

        Asset imageAsset = Asset.Builder.create(this, "SampleAsset")
                .path(new File(startDir, "images/my-image.png").toString()).build())

        Function.Builder.create(this, "myLambdaFunction")
                .code(Code.fromAsset(new File(startDir, "handler").toString()))
                .runtime(Runtime.PYTHON_3_6)
                .handler("index.lambda_handler")
                .environment(java.util.Map.of(    // Java 9 or later
                    "S3_BUCKET_NAME", imageAsset.getS3BucketName(),
                    "S3_OBJECT_KEY", imageAsset.getS3ObjectKey(),
                    "S3_OBJECT_URL", imageAsset.getS3ObjectUrl()))
                .build();
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.S3.Assets;
using System.IO;
using System.Collections.Generic;

var imageAsset = new Asset(this, "SampleAsset", new AssetProps
{
    Path = Path.Combine(Directory.GetCurrentDirectory(), @"images\my-image.png")
});
```

```
new Function(this, "myLambdaFunction", new FunctionProps
{
    Code = Code.FromAsset(Path.Combine(Directory.GetCurrentDirectory(), "handler")),
    Runtime = Runtime.PYTHON_3_6,
    Handler = "index.lambda_handler",
    Environment = new Dictionary<string, string>
    {
        ["S3_BUCKET_NAME"] = imageAsset.S3BucketName,
        ["S3_OBJECT_KEY"] = imageAsset.S3ObjectKey,
        ["S3_OBJECT_URL"] = imageAsset.S3ObjectUrl
    }
});
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awslambda"
  "github.com/aws/aws-cdk-go/awscdk/v2/awss3assets"
)

dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

imageAsset := awss3assets.NewAsset(stack, jsii.String("SampleAsset"),
 &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "images/my-image.png")),
})

awslambda.NewFunction(stack, jsii.String("myLambdaFunction"),
 &awslambda.FunctionProps{
  Code: awslambda.AssetCode_FromAsset(jsii.String(path.Join(dirName, "handler"))),
  Runtime: awslambda.Runtime_PYTHON_3_6(),
  Handler: jsii.String("index.lambda_handler"),
  Environment: &map[string]*string{
    "S3_BUCKET_NAME": imageAsset.S3BucketName(),
    "S3_OBJECT_KEY": imageAsset.S3ObjectKey(),
```

```
    "S3_URL": imageAsset.S3ObjectUrl(),
  },
})
```

Izin

Jika Anda menggunakan aset Amazon S3 secara langsung melalui modul aws-s3-assets, peran IAM, pengguna, atau grup, dan Anda perlu membaca aset saat runtime, berikan izin IAM aset tersebut melalui metode Asset.grantRead.

Contoh berikut memberikan izin baca grup IAM pada aset file.

TypeScript

```
import { Asset } from 'aws-cdk-lib/aws-s3-assets';
import * as path from 'path';

const asset = new Asset(this, 'MyFile', {
  path: path.join(__dirname, 'my-image.png')
});

const group = new iam.Group(this, 'MyUserGroup');
asset.grantRead(group);
```

JavaScript

```
const { Asset } = require('aws-cdk-lib/aws-s3-assets');
const path = require('path');

const asset = new Asset(this, 'MyFile', {
  path: path.join(__dirname, 'my-image.png')
});

const group = new iam.Group(this, 'MyUserGroup');
asset.grantRead(group);
```

Python

```
from aws_cdk.aws_s3_assets import Asset
import aws_cdk.aws_iam as iam
```

```
import os.path
dirname = os.path.dirname(__file__)

        asset = Asset(self, "MyFile",
            path=os.path.join(dirname, "my-image.png"))

        group = iam.Group(self, "MyUserGroup")
        asset.grant_read(group)
```

Java

```
import java.io.File;

import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.iam.Group;
import software.amazon.awscdk.services.s3.assets.Asset;

public class GrantStack extends Stack {
    public GrantStack(final App scope, final String id, final StackProps props) {
        super(scope, id, props);

        File startDir = new File(System.getProperty("user.dir"));

        Asset asset = Asset.Builder.create(this, "SampleAsset")
                .path(new File(startDir, "images/my-image.png").toString()).build();

        Group group = new Group(this, "MyUserGroup");
        asset.grantRead(group);   }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.IAM;
using Amazon.CDK.AWS.S3.Assets;
using System.IO;

var asset = new Asset(this, "MyFile", new AssetProps {
    Path = Path.Combine(Path.Combine(Directory.GetCurrentDirectory(), @"images\my-
image.png"))
});
```

```
var group = new Group(this, "MyUserGroup");
asset.GrantRead(group);
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsiam"
  "github.com/aws/aws-cdk-go/awscdk/v2/awss3assets"
)

dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

asset := awss3assets.NewAsset(stack, jsii.String("MyFile"), &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "my-image.png")),
})

group := awsiam.NewGroup(stack, jsii.String("MyUserGroup"), &awsiam.GroupProps{})

asset.GrantRead(group)
```

Aset gambar Docker

AWS CDK Mendukung bundling image Docker lokal sebagai aset melalui modul. aws-ecr-assets

Contoh berikut mendefinisikan image Docker yang dibangun secara lokal dan didorong ke Amazon ECR. Gambar dibuat dari direktori konteks Docker lokal (dengan Dockerfile) dan diunggah ke Amazon ECR oleh CLI atau pipeline AWS CDK CI/CD aplikasi Anda. Gambar dapat direferensikan secara alami di AWS CDK aplikasi Anda.

TypeScript

```
import { DockerImageAsset } from 'aws-cdk-lib/aws-ecr-assets';

const asset = new DockerImageAsset(this, 'MyBuildImage', {
```

```
  directory: path.join(__dirname, 'my-image')
});
```

JavaScript

```
const { DockerImageAsset } = require('aws-cdk-lib/aws-ecr-assets');

const asset = new DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image')
});
```

Python

```
from aws_cdk.aws_ecr_assets import DockerImageAsset

import os.path
dirname = os.path.dirname(__file__)

asset = DockerImageAsset(self, 'MyBuildImage',
    directory=os.path.join(dirname, 'my-image'))
```

Java

```
import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

File startDir = new File(System.getProperty("user.dir"));

DockerImageAsset asset = DockerImageAsset.Builder.create(this, "MyBuildImage")
            .directory(new File(startDir, "my-image").toString()).build();
```

C#

```
using System.IO;
using Amazon.CDK.AWS.ECR.Assets;

var asset = new DockerImageAsset(this, "MyBuildImage", new DockerImageAssetProps
{
    Directory = Path.Combine(Directory.GetCurrentDirectory(), "my-image")
});
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets"
)

dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String("MyBuildImage"),
 &awsecrassets.DockerImageAssetProps{
  Directory: jsii.String(path.Join(dirName, "my-image")),
})
```

`my-image`Direktori harus menyertakan Dockerfile. AWS CDK CLI membangun image Docker dari`my-image`, mendorongnya ke repositori Amazon ECR, dan menentukan nama repositori sebagai parameter ke tumpukan Anda. AWS CloudFormation Jenis aset gambar Docker mengekspos atribut waktu penerapan yang dapat direferensikan di pustaka dan aplikasi. AWS CDK Perintah AWS CDK CLI cdk synth menampilkan properti aset sebagai AWS CloudFormation parameter.

Contoh definisi tugas Amazon ECS

Kasus penggunaan yang umum adalah membuat Amazon ECS TaskDefinitionuntuk menjalankan kontainer Docker. Contoh berikut menentukan lokasi aset image Docker yang AWS CDK dibangun secara lokal dan mendorong ke Amazon ECR.

TypeScript

```
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecr_assets from 'aws-cdk-lib/aws-ecr-assets';
import * as path from 'path';

const taskDefinition = new ecs.FargateTaskDefinition(this, "TaskDef", {
  memoryLimitMiB: 1024,
  cpu: 512
```

```
});

const asset = new ecr_assets.DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image')
});

taskDefinition.addContainer("my-other-container", {
  image: ecs.ContainerImage.fromDockerImageAsset(asset)
});
```

JavaScript

```
const ecs = require('aws-cdk-lib/aws-ecs');
const ecr_assets = require('aws-cdk-lib/aws-ecr-assets');
const path = require('path');

const taskDefinition = new ecs.FargateTaskDefinition(this, "TaskDef", {
  memoryLimitMiB: 1024,
  cpu: 512
});

const asset = new ecr_assets.DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image')
});

taskDefinition.addContainer("my-other-container", {
  image: ecs.ContainerImage.fromDockerImageAsset(asset)
});
```

Python

```
import aws_cdk.aws_ecs as ecs
import aws_cdk.aws_ecr_assets as ecr_assets

import os.path
dirname = os.path.dirname(__file__)

task_definition = ecs.FargateTaskDefinition(self, "TaskDef",
    memory_limit_mib=1024,
    cpu=512)

asset = ecr_assets.DockerImageAsset(self, 'MyBuildImage',
    directory=os.path.join(dirname, 'my-image'))
```

```
task_definition.add_container("my-other-container",
    image=ecs.ContainerImage.from_docker_image_asset(asset))
```

Java

```
import java.io.File;

import software.amazon.awscdk.services.ecs.FargateTaskDefinition;
import software.amazon.awscdk.services.ecs.ContainerDefinitionOptions;
import software.amazon.awscdk.services.ecs.ContainerImage;

import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

File startDir = new File(System.getProperty("user.dir"));

FargateTaskDefinition taskDefinition = FargateTaskDefinition.Builder.create(
        this, "TaskDef").memoryLimitMiB(1024).cpu(512).build();

DockerImageAsset asset = DockerImageAsset.Builder.create(this, "MyBuildImage")
            .directory(new File(startDir, "my-image").toString()).build();

taskDefinition.addContainer("my-other-container",
        ContainerDefinitionOptions.builder()
            .image(ContainerImage.fromDockerImageAsset(asset))
            .build();
```

C#

```
using System.IO;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.Ecr.Assets;

var taskDefinition = new FargateTaskDefinition(this, "TaskDef", new
 FargateTaskDefinitionProps
{
    MemoryLimitMiB = 1024,
    Cpu = 512
});

var asset = new DockerImageAsset(this, "MyBuildImage", new DockerImageAssetProps
{
    Directory = Path.Combine(Directory.GetCurrentDirectory(), "my-image")
```

```
});

taskDefinition.AddContainer("my-other-container", new ContainerDefinitionOptions
{
    Image = ContainerImage.FromDockerImageAsset(asset)
});
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsecs"
)

dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

taskDefinition := awsecs.NewTaskDefinition(stack, jsii.String("TaskDef"),
 &awsecs.TaskDefinitionProps{
  MemoryMiB: jsii.String("1024"),
  Cpu: jsii.String("512"),
})

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String("MyBuildImage"),
 &awsecrassets.DockerImageAssetProps{
  Directory: jsii.String(path.Join(dirName, "my-image")),
})

taskDefinition.AddContainer(jsii.String("MyOtherContainer"),
 &awsecs.ContainerDefinitionOptions{
  Image: awsecs.ContainerImage_FromDockerImageAsset(asset),
})
```

Contoh atribut deploy-time

Contoh berikut menunjukkan cara menggunakan atribut deploy-time `repository` dan `imageUri` membuat definisi tugas Amazon ECS dengan tipe peluncuran. AWS Fargate Perhatikan bahwa pencarian repo Amazon ECR memerlukan tag gambar, bukan URI-nya, jadi kami memotongnya dari akhir URI aset.

TypeScript

```
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as path from 'path';
import { DockerImageAsset } from 'aws-cdk-lib/aws-ecr-assets';

const asset = new DockerImageAsset(this, 'my-image', {
  directory: path.join(__dirname, "..", "demo-image")
});

const taskDefinition = new ecs.FargateTaskDefinition(this, "TaskDef", {
  memoryLimitMiB: 1024,
  cpu: 512
});

taskDefinition.addContainer("my-other-container", {
  image: ecs.ContainerImage.fromEcrRepository(asset.repository,
 asset.imageUri.split(":").pop())
});
```

JavaScript

```
const ecs = require('aws-cdk-lib/aws-ecs');
const path = require('path');
const { DockerImageAsset } = require('aws-cdk-lib/aws-ecr-assets');

const asset = new DockerImageAsset(this, 'my-image', {
  directory: path.join(__dirname, "..", "demo-image")
});

const taskDefinition = new ecs.FargateTaskDefinition(this, "TaskDef", {
  memoryLimitMiB: 1024,
  cpu: 512
});

taskDefinition.addContainer("my-other-container", {
```

```
  image: ecs.ContainerImage.fromEcrRepository(asset.repository,
 asset.imageUri.split(":").pop())
});
```

Python

```
import aws_cdk.aws_ecs as ecs
from aws_cdk.aws_ecr_assets import DockerImageAsset

import os.path
dirname = os.path.dirname(__file__)

asset = DockerImageAsset(self, 'my-image',
    directory=os.path.join(dirname, "..", "demo-image"))

task_definition = ecs.FargateTaskDefinition(self, "TaskDef",
    memory_limit_mib=1024, cpu=512)

task_definition.add_container("my-other-container",
    image=ecs.ContainerImage.from_ecr_repository(
        asset.repository, asset.image_uri.rpartition(":")[-1]))
```

Java

```
import java.io.File;

import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

import software.amazon.awscdk.services.ecs.FargateTaskDefinition;
import software.amazon.awscdk.services.ecs.ContainerDefinitionOptions;
import software.amazon.awscdk.services.ecs.ContainerImage;

File startDir = new File(System.getProperty("user.dir"));

DockerImageAsset asset = DockerImageAsset.Builder.create(this, "my-image")
            .directory(new File(startDir, "demo-image").toString()).build();

FargateTaskDefinition taskDefinition = FargateTaskDefinition.Builder.create(
        this, "TaskDef").memoryLimitMiB(1024).cpu(512).build();

// extract the tag from the asset's image URI for use in ECR repo lookup
String imageUri = asset.getImageUri();
String imageTag = imageUri.substring(imageUri.lastIndexOf(":") + 1);
```

```
taskDefinition.addContainer("my-other-container",
        ContainerDefinitionOptions.builder().image(ContainerImage.fromEcrRepository(
                asset.getRepository(), imageTag)).build());
```

C#

```
using System.IO;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECR.Assets;

var asset = new DockerImageAsset(this, "my-image", new DockerImageAssetProps {
    Directory = Path.Combine(Directory.GetCurrentDirectory(), "demo-image")
});

var taskDefinition = new FargateTaskDefinition(this, "TaskDef", new
 FargateTaskDefinitionProps
{
    MemoryLimitMiB = 1024,
    Cpu = 512
});

taskDefinition.AddContainer("my-other-container", new ContainerDefinitionOptions
{
    Image = ContainerImage.FromEcrRepository(asset.Repository,
 asset.ImageUri.Split(":").Last())
});
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsecs"
)

dirName, err := os.Getwd()
if err != nil {
  panic(err)
}
```

```
asset := awsecrassets.NewDockerImageAsset(stack, jsii.String("MyImage"),
 &awsecrassets.DockerImageAssetProps{
  Directory: jsii.String(path.Join(dirName, "demo-image")),
})

taskDefinition := awsecs.NewFargateTaskDefinition(stack, jsii.String("TaskDef"),
 &awsecs.FargateTaskDefinitionProps{
  MemoryLimitMiB: jsii.Number(1024),
  Cpu: jsii.Number(512),
})

taskDefinition.AddContainer(jsii.String("MyOtherContainer"),
 &awsecs.ContainerDefinitionOptions{
  Image: awsecs.ContainerImage_FromEcrRepository(asset.Repository(),
 asset.ImageTag()),
})
```

Membangun contoh argumen

Anda dapat memberikan argumen build yang disesuaikan untuk langkah build Docker melalui opsi properti `buildArgs` (Python`build_args`:) saat AWS CDK CLI membangun gambar selama penerapan.

TypeScript

```
const asset = new DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image'),
  buildArgs: {
    HTTP_PROXY: 'http://10.20.30.2:1234'
  }
});
```

JavaScript

```
const asset = new DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image'),
  buildArgs: {
    HTTP_PROXY: 'http://10.20.30.2:1234'
  }
});
```

Python

```
asset = DockerImageAsset(self, "MyBulidImage",
    directory=os.path.join(dirname, "my-image"),
    build_args=dict(HTTP_PROXY="http://10.20.30.2:1234"))
```

Java

```
DockerImageAsset asset = DockerImageAsset.Builder.create(this, "my-image"),
            .directory(new File(startDir, "my-image").toString())
            .buildArgs(java.util.Map.of(    // Java 9 or later
                "HTTP_PROXY", "http://10.20.30.2:1234"))
            .build();
```

C#

```
var asset = new DockerImageAsset(this, "MyBuildImage", new DockerImageAssetProps {
    Directory = Path.Combine(Directory.GetCurrentDirectory(), "my-image"),
    BuildArgs = new Dictionary<string, string>
    {
        ["HTTP_PROXY"] = "http://10.20.30.2:1234"
    }
});
```

Go

```
dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String("MyBuildImage"),
 &awsecrassets.DockerImageAssetProps{
  Directory: jsii.String(path.Join(dirName, "my-image")),
  BuildArgs: &map[string]*string{
    "HTTP_PROXY": jsii.String("http://10.20.30.2:1234"),
  },
})
```

Izin

Jika Anda menggunakan modul yang mendukung aset image Docker, seperti aws-ecs, akan AWS CDK mengelola izin untuk Anda saat Anda menggunakan aset secara langsung atau melalui. ContainerImage fromEcrRepository(Python:`from_ecr_repository`). Jika Anda menggunakan aset image Docker secara langsung, pastikan bahwa prinsipal konsumsi memiliki izin untuk menarik gambar.

Dalam kebanyakan kasus, Anda harus menggunakan metode asset.repository.GrantPull (Python:. `grant_pull` Ini memodifikasi kebijakan IAM dari prinsipal untuk memungkinkannya menarik gambar dari repositori ini. Jika prinsipal yang menarik gambar tidak berada di akun yang sama, atau jika itu adalah AWS layanan yang tidak berperan dalam akun Anda (seperti AWS CodeBuild), Anda harus memberikan izin tarik pada kebijakan sumber daya dan bukan pada kebijakan prinsipal. Gunakan asset.repository. addToResourceMetode kebijakan (Python:`add_to_resource_policy`) untuk memberikan izin utama yang sesuai.

AWS CloudFormation metadata sumber daya

> **Note**
>
> Bagian ini hanya relevan untuk penulis konstruksi. Dalam situasi tertentu, alat perlu mengetahui bahwa sumber daya CFN tertentu menggunakan aset lokal. Misalnya, Anda dapat menggunakan AWS SAM CLI untuk menjalankan fungsi Lambda secara lokal untuk tujuan debugging. Lihat the section called “AWS SAM integrasi” untuk detail.

Untuk mengaktifkan kasus penggunaan seperti itu, alat eksternal berkonsultasi dengan satu set entri metadata tentang sumber daya: AWS CloudFormation

- `aws:asset:path`— Menunjuk ke jalur lokal aset.
- `aws:asset:property`— Nama properti sumber daya tempat aset digunakan.

Dengan menggunakan dua entri metadata ini, alat dapat mengidentifikasi bahwa aset digunakan oleh sumber daya tertentu, dan memungkinkan pengalaman lokal tingkat lanjut.

Untuk menambahkan entri metadata ini ke sumber daya, gunakan metode (`asset.addResourceMetadata`Python:). `add_resource_metadata`

Izin

AWS Construct Library menggunakan beberapa idiom umum yang diimplementasikan secara luas untuk mengelola akses dan izin. Modul IAM memberi Anda alat yang Anda butuhkan untuk menggunakan idiom ini.

AWS CDK digunakan AWS CloudFormation untuk menyebarkan perubahan. Setiap penyebaran melibatkan aktor (baik pengembang, atau sistem otomatis) yang memulai AWS CloudFormation penerapan. Selama melakukan ini, aktor akan mengambil satu atau lebih Identitas IAM (pengguna atau peran) dan secara opsional memberikan peran ke. AWS CloudFormation

Jika Anda menggunakannya AWS IAM Identity Center untuk mengautentikasi sebagai pengguna, maka penyedia masuk tunggal menyediakan kredensil sesi berumur pendek yang mengizinkan Anda untuk bertindak sebagai peran IAM yang telah ditentukan sebelumnya. Untuk mempelajari cara AWS CDK memperoleh AWS kredensil dari autentikasi Pusat Identitas IAM, lihat Memahami autentikasi Pusat Identitas IAM di Panduan Referensi SDK dan Alat.AWS

Pengguna utama

Prinsipal IAM adalah AWS entitas yang diautentikasi yang mewakili pengguna, layanan, atau aplikasi yang dapat memanggil AWS API. AWS Construct Library mendukung penetapan prinsipal dalam beberapa cara fleksibel untuk memberi mereka akses sumber daya Anda. AWS

Dalam konteks keamanan, istilah “prinsipal” mengacu secara khusus pada entitas yang diautentikasi seperti pengguna. Objek seperti grup dan peran tidak mewakili pengguna (dan entitas lain yang diautentikasi) melainkan mengidentifikasi mereka secara tidak langsung untuk tujuan pemberian izin.

Misalnya, jika Anda membuat grup IAM, Anda dapat memberikan grup (dan dengan demikian anggotanya) akses tulis ke tabel Amazon RDS. Namun, grup itu sendiri bukan prinsipal karena tidak mewakili satu entitas (juga, Anda tidak dapat masuk ke grup).

Di perpustakaan IAM CDK, kelas yang secara langsung atau tidak langsung mengidentifikasi prinsipal mengimplementasikan `IPrincipal`antarmuka, memungkinkan objek ini digunakan secara bergantian dalam kebijakan akses. Namun, tidak semua dari mereka adalah kepala sekolah dalam arti keamanan. Benda-benda ini meliputi:

1. Sumber daya IAM seperti`Role`,`User`, dan `Group`
2. Prinsipal layanan () `new iam.ServicePrincipal('service.amazonaws.com')`

3. Kepala sekolah federasi () `new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`
4. Prinsipal akun (`new iam.AccountPrincipal('0123456789012')`)
5. Prinsipal pengguna kanonik () `new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`
6. AWS Organizations kepala sekolah () `new iam.OrganizationPrincipal('org-id')`
7. Prinsipal ARN sewenang-wenang () `new iam.ArnPrincipal(res.arn)`
8. An `iam.CompositePrincipal(principal1, principal2, ...)` untuk mempercayai banyak prinsip

Izin

Setiap konstruksi yang mewakili sumber daya yang dapat diakses, seperti bucket Amazon S3 atau tabel Amazon DynamoDB, memiliki metode yang memberikan akses ke entitas lain. Semua metode tersebut memiliki nama yang dimulai dengan hibah.

Misalnya, bucket Amazon S3 memiliki metode dan `grantRead` (`grantReadWrite`Python:`grant_read`,`grant_read_write`) untuk mengaktifkan akses baca dan baca/tulis, masing-masing, dari entitas ke bucket. Entitas tidak harus tahu persis izin IAM Amazon S3 mana yang diperlukan untuk melakukan operasi ini.

Argumen pertama dari metode hibah selalu bertipe iGrantable. Antarmuka ini mewakili entitas yang dapat diberikan izin. Artinya, ini mewakili sumber daya dengan peran, seperti objek IAM`Role`,`User`, dan`Group`.

Entitas lain juga dapat diberikan izin. Misalnya, nanti dalam topik ini, kami menunjukkan cara memberikan akses CodeBuild proyek ke bucket Amazon S3. Umumnya, peran terkait diperoleh melalui `role` properti pada entitas yang diberikan akses.

Sumber daya yang menggunakan peran eksekusi, seperti`lambda.Function`, juga diterapkan`IGrantable`, sehingga Anda dapat memberi mereka akses secara langsung alih-alih memberikan akses ke peran mereka. Misalnya, jika `bucket` adalah bucket Amazon S3, dan `function` merupakan fungsi Lambda, kode berikut memberikan akses baca fungsi ke bucket.

TypeScript

```
bucket.grantRead(function);
```

JavaScript

```
bucket.grantRead(function);
```

Python

```
bucket.grant_read(function)
```

Java

```
bucket.grantRead(function);
```

C#

```
bucket.GrantRead(function);
```

Terkadang izin harus diterapkan saat tumpukan Anda sedang digunakan. Salah satu kasus tersebut adalah ketika Anda memberikan akses sumber daya AWS CloudFormation khusus ke beberapa sumber daya lain. Sumber daya kustom akan dipanggil selama penerapan, sehingga harus memiliki izin yang ditentukan pada waktu penerapan.

Kasus lain adalah ketika layanan memverifikasi bahwa peran yang Anda berikan kepadanya memiliki kebijakan yang tepat diterapkan. (Sejumlah AWS layanan melakukan ini untuk memastikan bahwa Anda tidak lupa untuk menetapkan kebijakan.) Dalam kasus tersebut, penerapan mungkin gagal jika izin diterapkan terlambat.

Untuk memaksa izin hibah diterapkan sebelum sumber daya lain dibuat, Anda dapat menambahkan ketergantungan pada hibah itu sendiri, seperti yang ditunjukkan di sini. Meskipun nilai pengembalian metode hibah biasanya dibuang, setiap metode hibah sebenarnya mengembalikan objek`iam.Grant`.

TypeScript

```
const grant = bucket.grantRead(lambda);
const custom = new CustomResource(...);
custom.node.addDependency(grant);
```

JavaScript

```
const grant = bucket.grantRead(lambda);
```

```
const custom = new CustomResource(...);
custom.node.addDependency(grant);
```

Python

```
grant = bucket.grant_read(function)
custom = CustomResource(...)
custom.node.add_dependency(grant)
```

Java

```
Grant grant = bucket.grantRead(function);
CustomResource custom = new CustomResource(...);
custom.node.addDependency(grant);
```

C#

```
var grant = bucket.GrantRead(function);
var custom = new CustomResource(...);
custom.node.AddDependency(grant);
```

Peran

Paket IAM berisi `Role` konstruksi yang mewakili peran IAM. Kode berikut membuat peran baru, mempercayai layanan Amazon EC2.

TypeScript

```
import * as iam from 'aws-cdk-lib/aws-iam';

const role = new iam.Role(this, 'Role', {
  assumedBy: new iam.ServicePrincipal('ec2.amazonaws.com'),   // required
});
```

JavaScript

```
const iam = require('aws-cdk-lib/aws-iam');

const role = new iam.Role(this, 'Role', {
```

```
  assumedBy: new iam.ServicePrincipal('ec2.amazonaws.com')   // required
});
```

Python

```
import aws_cdk.aws_iam as iam

role = iam.Role(self, "Role",
          assumed_by=iam.ServicePrincipal("ec2.amazonaws.com")) # required
```

Java

```
import software.amazon.awscdk.services.iam.Role;
import software.amazon.awscdk.services.iam.ServicePrincipal;

Role role = Role.Builder.create(this, "Role")
        .assumedBy(new ServicePrincipal("ec2.amazonaws.com")).build();
```

C#

```
using Amazon.CDK.AWS.IAM;

var role = new Role(this, "Role", new RoleProps
{
    AssumedBy = new ServicePrincipal("ec2.amazonaws.com"),   // required
});
```

Anda dapat menambahkan izin ke peran dengan memanggil `addToPolicy` metode peran (Python`add_to_policy`:), meneruskan `PolicyStatement` sebuah yang mendefinisikan aturan yang akan ditambahkan. Pernyataan ditambahkan ke kebijakan default peran; jika tidak ada, satu akan dibuat.

Contoh berikut menambahkan pernyataan `Deny` kebijakan untuk peran untuk tindakan `ec2:SomeAction` dan `s3:AnotherAction` sumber daya `bucket` dan `otherRole` (Python:`other_role`), dengan syarat bahwa layanan resmi adalah. AWS CodeBuild

TypeScript

```
role.addToPolicy(new iam.PolicyStatement({
  effect: iam.Effect.DENY,
```

```
  resources: [bucket.bucketArn, otherRole.roleArn],
  actions: ['ec2:SomeAction', 's3:AnotherAction'],
  conditions: {StringEquals: {
    'ec2:AuthorizedService': 'codebuild.amazonaws.com',
}}}));
```

JavaScript

```
role.addToPolicy(new iam.PolicyStatement({
  effect: iam.Effect.DENY,
  resources: [bucket.bucketArn, otherRole.roleArn],
  actions: ['ec2:SomeAction', 's3:AnotherAction'],
  conditions: {StringEquals: {
    'ec2:AuthorizedService': 'codebuild.amazonaws.com'
}}}));
```

Python

```
role.add_to_policy(iam.PolicyStatement(
    effect=iam.Effect.DENY,
    resources=[bucket.bucket_arn, other_role.role_arn],
    actions=["ec2:SomeAction", "s3:AnotherAction"],
    conditions={"StringEquals": {
        "ec2:AuthorizedService": "codebuild.amazonaws.com"}}
))
```

Java

```
role.addToPolicy(PolicyStatement.Builder.create()
        .effect(Effect.DENY)
        .resources(Arrays.asList(bucket.getBucketArn(), otherRole.getRoleArn()))
        .actions(Arrays.asList("ec2:SomeAction", "s3:AnotherAction"))
        .conditions(java.util.Map.of(    // Map.of requires Java 9 or later
            "StringEquals", java.util.Map.of(
                "ec2:AuthorizedService", "codebuild.amazonaws.com")))
        .build());
```

C#

```
role.AddToPolicy(new PolicyStatement(new PolicyStatementProps
{
    Effect = Effect.DENY,
```

```
    Resources = new string[] { bucket.BucketArn, otherRole.RoleArn },
    Actions = new string[] { "ec2:SomeAction", "s3:AnotherAction" },
    Conditions = new Dictionary<string, object>
    {
        ["StringEquals"] = new Dictionary<string, string>
        {
            ["ec2:AuthorizedService"] = "codebuild.amazonaws.com"
        }
    }
}));
```

Pada contoh sebelumnya, kita telah membuat `PolicyStatement` inline baru dengan panggilan (`addToPolicy`Python:). `add_to_policy` Anda juga dapat meneruskan pernyataan kebijakan yang ada atau yang telah Anda modifikasi. PolicyStatementObjek memiliki banyak metode untuk menambahkan prinsip, sumber daya, kondisi, dan tindakan.

Jika Anda menggunakan konstruksi yang membutuhkan peran agar berfungsi dengan benar, Anda dapat melakukan salah satu hal berikut:

- Lewati peran yang ada saat membuat instance objek konstruksi.
- Biarkan konstruksi menciptakan peran baru untuk Anda, mempercayai prinsip layanan yang tepat. Contoh berikut menggunakan konstruksi seperti itu: CodeBuild proyek.

TypeScript

```
import * as codebuild from 'aws-cdk-lib/aws-codebuild';

// imagine roleOrUndefined is a function that might return a Role object
// under some conditions, and undefined under other conditions
const someRole: iam.IRole | undefined = roleOrUndefined();

const project = new codebuild.Project(this, 'Project', {
  // if someRole is undefined, the Project creates a new default role,
  // trusting the codebuild.amazonaws.com service principal
  role: someRole,
});
```

JavaScript

```
const codebuild = require('aws-cdk-lib/aws-codebuild');
```

```
// imagine roleOrUndefined is a function that might return a Role object
// under some conditions, and undefined under other conditions
const someRole = roleOrUndefined();

const project = new codebuild.Project(this, 'Project', {
  // if someRole is undefined, the Project creates a new default role,
  // trusting the codebuild.amazonaws.com service principal
  role: someRole
});
```

Python

```
import aws_cdk.aws_codebuild as codebuild

# imagine role_or_none is a function that might return a Role object
# under some conditions, and None under other conditions
some_role = role_or_none();

project = codebuild.Project(self, "Project",
# if role is None, the Project creates a new default role,
# trusting the codebuild.amazonaws.com service principal
role=some_role)
```

Java

```
import software.amazon.awscdk.services.iam.Role;
import software.amazon.awscdk.services.codebuild.Project;

// imagine roleOrNull is a function that might return a Role object
// under some conditions, and null under other conditions
Role someRole = roleOrNull();

// if someRole is null, the Project creates a new default role,
// trusting the codebuild.amazonaws.com service principal
Project project = Project.Builder.create(this, "Project")
        .role(someRole).build();
```

C#

```
using Amazon.CDK.AWS.CodeBuild;
```

```
// imagine roleOrNull is a function that might return a Role object
// under some conditions, and null under other conditions
var someRole = roleOrNull();

// if someRole is null, the Project creates a new default role,
// trusting the codebuild.amazonaws.com service principal
var project = new Project(this, "Project", new ProjectProps
{
    Role = someRole
});
```

Setelah objek dibuat, peran (apakah peran diteruskan atau peran default yang dibuat oleh konstruksi) tersedia sebagai properti`role`. Namun, properti ini tidak tersedia pada sumber daya eksternal. Oleh karena itu, konstruksi ini memiliki metode `addToRolePolicy` (Python`add_to_role_policy`:).

Metode ini tidak melakukan apa-apa jika konstruksinya adalah sumber daya eksternal, dan ia memanggil metode `addToPolicy` (Python`add_to_policy`:) dari properti sebaliknya`role`. Ini menghemat kesulitan menangani kasus yang tidak ditentukan secara eksplisit.

Contoh berikut menunjukkan:

TypeScript

```
// project is imported into the CDK application
const project = codebuild.Project.fromProjectName(this, 'Project', 'ProjectName');

// project is imported, so project.role is undefined, and this call has no effect
project.addToRolePolicy(new iam.PolicyStatement({
  effect: iam.Effect.ALLOW,   // ... and so on defining the policy
}));
```

JavaScript

```
// project is imported into the CDK application
const project = codebuild.Project.fromProjectName(this, 'Project', 'ProjectName');

// project is imported, so project.role is undefined, and this call has no effect
project.addToRolePolicy(new iam.PolicyStatement({
  effect: iam.Effect.ALLOW   // ... and so on defining the policy
}));
```

Python

```
# project is imported into the CDK application
project = codebuild.Project.from_project_name(self, 'Project', 'ProjectName')

# project is imported, so project.role is undefined, and this call has no effect
project.add_to_role_policy(iam.PolicyStatement(
  effect=iam.Effect.ALLOW,   # ... and so on defining the policy
)
```

Java

```
// project is imported into the CDK application
Project project = Project.fromProjectName(this, "Project", "ProjectName");

// project is imported, so project.getRole() is null, and this call has no effect
project.addToRolePolicy(PolicyStatement.Builder.create()
        .effect(Effect.ALLOW)   // .. and so on defining the policy
        .build());
```

C#

```
// project is imported into the CDK application
var project = Project.FromProjectName(this, "Project", "ProjectName");

// project is imported, so project.role is null, and this call has no effect
project.AddToRolePolicy(new PolicyStatement(new PolicyStatementProps
{
    Effect = Effect.ALLOW, // ... and so on defining the policy
}));
```

Kebijakan sumber daya

Beberapa sumber daya AWS, seperti bucket Amazon S3 dan peran IAM, juga memiliki kebijakan sumber daya. Konstruksi ini memiliki `addToResourcePolicy` metode (Python`add_to_resource_policy`:), yang mengambil `PolicyStatement` sebagai argumennya. Setiap pernyataan kebijakan yang ditambahkan ke kebijakan sumber daya harus menentukan setidaknya satu prinsipal.

Dalam contoh berikut, bucket Amazon S3 `bucket` memberikan peran dengan `s3:SomeAction` izin untuk dirinya sendiri.

TypeScript

```
bucket.addToResourcePolicy(new iam.PolicyStatement({
  effect: iam.Effect.ALLOW,
  actions: ['s3:SomeAction'],
  resources: [bucket.bucketArn],
  principals: [role]
}));
```

JavaScript

```
bucket.addToResourcePolicy(new iam.PolicyStatement({
  effect: iam.Effect.ALLOW,
  actions: ['s3:SomeAction'],
  resources: [bucket.bucketArn],
  principals: [role]
}));
```

Python

```
bucket.add_to_resource_policy(iam.PolicyStatement(
    effect=iam.Effect.ALLOW,
    actions=["s3:SomeAction"],
    resources=[bucket.bucket_arn],
    principals=role))
```

Java

```
bucket.addToResourcePolicy(PolicyStatement.Builder.create()
        .effect(Effect.ALLOW)
        .actions(Arrays.asList("s3:SomeAction"))
        .resources(Arrays.asList(bucket.getBucketArn()))
        .principals(Arrays.asList(role))
        .build());
```

C#

```
bucket.AddToResourcePolicy(new PolicyStatement(new PolicyStatementProps
```

```
{
    Effect = Effect.ALLOW,
    Actions = new string[] { "s3:SomeAction" },
    Resources = new string[] { bucket.BucketArn },
    Principals = new IPrincipal[] { role }
}));
```

Menggunakan objek IAM eksternal

Jika Anda telah menetapkan pengguna, prinsipal, grup, atau peran IAM di luar AWS CDK aplikasi, Anda dapat menggunakan objek IAM tersebut di aplikasi Anda AWS CDK . Untuk melakukannya, buat referensi untuk itu menggunakan ARN atau namanya. (Gunakan nama untuk pengguna, grup, dan peran.) Referensi yang dikembalikan kemudian dapat digunakan untuk memberikan izin atau untuk membuat pernyataan kebijakan seperti yang dijelaskan sebelumnya.

- Untuk pengguna, hubungi `User.fromUserArn()` atau`User.fromUserName()`. `User.fromUserAttributes()`juga tersedia, tetapi saat ini menyediakan fungsionalitas yang sama seperti`User.fromUserArn()`.
- Untuk prinsipal, buat instance objek. `ArnPrincipal`
- Untuk grup, hubungi `Group.fromGroupArn()` atau`Group.fromGroupName()`.
- Untuk peran, hubungi `Role.fromRoleArn()` atau`Role.fromRoleName()`.

Kebijakan (termasuk kebijakan terkelola) dapat digunakan dengan cara yang sama menggunakan metode berikut. Anda dapat menggunakan referensi ke objek ini di mana pun kebijakan IAM diperlukan.

- `Policy.fromPolicyName`
- `ManagedPolicy.fromManagedPolicyArn`
- `ManagedPolicy.fromManagedPolicyName`
- `ManagedPolicy.fromAwsManagedPolicyName`

> **Note**
>
> Seperti semua referensi ke AWS sumber daya eksternal, Anda tidak dapat memodifikasi objek IAM eksternal di aplikasi CDK Anda.

Konteks runtime

Nilai konteks adalah pasangan nilai kunci yang dapat dikaitkan dengan aplikasi, tumpukan, atau konstruksi. Mereka dapat diberikan ke aplikasi Anda dari file (biasanya salah satu `cdk.json` atau `cdk.context.json` di direktori proyek Anda) atau pada baris perintah.

CDK Toolkit menggunakan konteks untuk menyimpan nilai yang diambil dari AWS akun Anda selama sintesis. Nilai termasuk Availability Zone di akun Anda atau ID Amazon Machine Image (AMI) yang saat ini tersedia untuk instans Amazon EC2. Karena nilai-nilai ini disediakan oleh AWS akun Anda, mereka dapat berubah di antara proses aplikasi CDK Anda. Ini menjadikan mereka sumber potensial perubahan yang tidak diinginkan. Perilaku caching CDK Toolkit "membekukan" nilai-nilai ini untuk aplikasi CDK Anda hingga Anda memutuskan untuk menerima nilai baru.

Bayangkan skenario berikut tanpa cache konteks. Katakanlah Anda menentukan "Amazon Linux terbaru" sebagai AMI untuk instans Amazon EC2 Anda, dan versi baru AMI ini dirilis. Kemudian, lain kali Anda menerapkan tumpukan CDK Anda, instance Anda yang sudah diterapkan akan menggunakan AMI yang sudah ketinggalan zaman ("salah") dan perlu ditingkatkan. Memutakhirkan akan mengakibatkan penggantian semua instance Anda yang ada dengan yang baru, yang mungkin tidak terduga dan tidak diinginkan.

Sebagai gantinya, CDK mencatat AMI akun Anda yang tersedia di `cdk.context.json` file proyek Anda, dan menggunakan nilai tersimpan untuk operasi sintesis masa depan. Dengan cara ini, daftar AMI tidak lagi menjadi sumber perubahan potensial. Anda juga dapat yakin bahwa tumpukan Anda akan selalu disintesis ke templat yang sama AWS CloudFormation .

Tidak semua nilai konteks adalah nilai cache dari AWS lingkungan Anda. the section called "Bendera fitur"juga nilai konteks. Anda juga dapat membuat nilai konteks Anda sendiri untuk digunakan oleh aplikasi atau konstruksi Anda.

Kunci konteks adalah string. Nilai dapat berupa jenis apa pun yang didukung oleh JSON: angka, string, array, atau objek.

> **Tip**
>
> Jika konstruksi Anda membuat nilai konteksnya sendiri, sertakan nama paket perpustakaan Anda di kuncinya sehingga tidak akan bertentangan dengan nilai konteks paket lain.

Banyak nilai konteks dikaitkan dengan AWS lingkungan tertentu, dan aplikasi CDK tertentu dapat digunakan di lebih dari satu lingkungan. Kunci untuk nilai tersebut mencakup AWS akun dan Wilayah sehingga nilai dari lingkungan yang berbeda tidak bertentangan.

Kunci konteks berikut menggambarkan format yang digunakan oleh AWS CDK, termasuk akun dan Wilayah.

```
availability-zones:account=123456789012:region=eu-central-1
```

> **Important**
>
> Nilai konteks cache dikelola oleh AWS CDK dan konstruksinya, termasuk konstruksi yang dapat Anda tulis. Jangan menambahkan atau mengubah nilai konteks cache dengan mengedit file secara manual. Akan tetapi, berguna untuk meninjau `cdk.context.json` sesekali untuk melihat nilai apa yang sedang di-cache. Nilai konteks yang tidak mewakili nilai cache harus disimpan di bawah `context` kunci. `cdk.json` Dengan cara ini, mereka tidak akan dihapus ketika nilai cache dihapus.

Sumber nilai konteks

Nilai konteks dapat diberikan ke AWS CDK aplikasi Anda dengan enam cara berbeda:

- Secara otomatis dari AWS akun saat ini.
- Melalui --context opsi ke cdk perintah. (Nilai-nilai ini selalu string.)
- Dalam `cdk.context.json` file proyek.
- Di `context` kunci `cdk.json` file proyek.
- Di `context` kunci `~/.cdk.json` file Anda.
- Di AWS CDK aplikasi Anda menggunakan `construct.node.setContext()` metode ini.

File proyek `cdk.context.json` adalah tempat nilai konteks AWS CDK cache diambil dari akun Anda AWS . Praktik ini menghindari perubahan tak terduga pada penerapan Anda saat, misalnya, Availability Zone baru diperkenalkan. AWS CDK Tidak menulis data konteks ke salah satu file lain yang terdaftar.

> **Important**
>
> Karena mereka adalah bagian dari status aplikasi Anda, `cdk.json` dan `cdk.context.json` harus berkomitmen untuk kontrol sumber bersama dengan kode sumber aplikasi lainnya. Jika tidak, penerapan di lingkungan lain (misalnya, pipeline CI) mungkin menghasilkan hasil yang tidak konsisten.

Nilai-nilai konteks tercakup pada konstruksi yang menciptakannya; mereka terlihat oleh konstruksi anak, tetapi tidak untuk orang tua atau saudara kandung. Nilai konteks yang ditetapkan oleh AWS CDK Toolkit (cdkperintah) dapat diatur secara otomatis, dari file, atau dari --context opsi. Nilai konteks dari sumber-sumber ini secara implisit ditetapkan pada konstruksi. App Oleh karena itu, mereka terlihat oleh setiap konstruksi di setiap tumpukan di aplikasi.

Aplikasi Anda dapat membaca nilai konteks menggunakan `construct.node.tryGetContext` metode ini. Jika entri yang diminta tidak ditemukan pada konstruksi saat ini atau salah satu induknya, hasilnya adalah`undefined`. (Atau, hasilnya bisa setara dengan bahasa Anda, seperti None di Python.)

Metode konteks

AWS CDK Mendukung beberapa metode konteks yang memungkinkan AWS CDK aplikasi memperoleh informasi kontekstual dari lingkungan. AWS Misalnya, Anda bisa mendapatkan daftar Availability Zone yang tersedia di AWS akun dan Region tertentu, menggunakan metode Stack.availabilityZones.

Berikut ini adalah metode konteksnya:

HostedZone.FromLookup

: Mendapatkan zona yang dihosting di akun Anda.

Stack.availabilityZones

: Mendapatkan Availability Zone yang didukung.

StringParameter.valueFromLookup

: Mendapat nilai dari Amazon EC2 Systems Manager Parameter Store Wilayah saat ini.

VPC.FromLookup

: Mendapatkan Amazon Virtual Private Clouds yang ada di akun Anda.

LookupMachineImage

Mencari image mesin untuk digunakan dengan instance NAT di Amazon Virtual Private Cloud.

Jika nilai konteks yang diperlukan tidak tersedia, AWS CDK aplikasi akan memberi tahu CDK Toolkit bahwa informasi konteks tidak ada. Selanjutnya, CLI menanyakan AWS akun saat ini untuk informasi dan menyimpan informasi konteks yang dihasilkan dalam file. `cdk.context.json` Kemudian, ia mengeksekusi AWS CDK aplikasi lagi dengan nilai konteks.

Melihat dan mengelola konteks

Gunakan cdk context perintah untuk melihat dan mengelola informasi dalam `cdk.context.json` file Anda. Untuk melihat informasi ini, gunakan cdk context perintah tanpa opsi apa pun. Outputnya harus seperti berikut ini.

```
Context found in cdk.json:

##############################################################################################
# # # Key                                                                  # Value
                                     #
##############################################################################################
# 1 # availability-zones:account=123456789012:region=eu-central-1 # [ "eu-central-1a",
 "eu-central-1b", "eu-central-1c" ]   #
##############################################################################################
# 2 # availability-zones:account=123456789012:region=eu-west-1    # [ "eu-west-1a",
 "eu-west-1b", "eu-west-1c" ]            #
##############################################################################################

Run cdk context --reset KEY_OR_NUMBER to remove a context key. If it is a cached value,
 it will be refreshed on the next cdk synth.
```

Untuk menghapus nilai konteks, jalankancdk context --reset, tentukan kunci atau angka yang sesuai dengan nilai. Contoh berikut menghapus nilai yang sesuai dengan kunci kedua dalam contoh sebelumnya. Nilai ini mewakili daftar Availability Zone di Wilayah Eropa (Irlandia).

```
cdk context --reset 2
```

```
Context value
availability-zones:account=123456789012:region=eu-west-1
```

```
reset. It will be refreshed on the next SDK synthesis run.
```

Oleh karena itu, jika Anda ingin memperbarui ke versi terbaru AMI Amazon Linux, gunakan contoh sebelumnya untuk melakukan pembaruan terkontrol dari nilai konteks dan mengatur ulang. Kemudian, sintesis dan terapkan aplikasi Anda lagi.

```
cdk synth
```

Untuk menghapus semua nilai konteks yang disimpan untuk aplikasi Anda, jalankancdk context --clear, sebagai berikut.

```
cdk context --clear
```

Hanya nilai konteks yang disimpan di `cdk.context.json` dapat diatur ulang atau dihapus. AWS CDK Itu tidak menyentuh nilai konteks lainnya. Oleh karena itu, untuk melindungi nilai konteks agar tidak disetel ulang menggunakan perintah ini, Anda dapat menyalin nilainya`cdk.json`.

AWS CDK Bendera toolkit **`--context`**

Gunakan opsi `--context` (`-c`singkatnya) untuk meneruskan nilai konteks runtime ke aplikasi CDK Anda selama sintesis atau penerapan.

```
cdk synth --context key=value MyStack
```

Untuk menentukan beberapa nilai konteks, ulangi --context opsi beberapa kali, berikan satu pasangan kunci-nilai setiap kali.

```
cdk synth --context key1=value1 --context key2=value2 MyStack
```

Saat mensintesis beberapa tumpukan, nilai konteks yang ditentukan diteruskan ke semua tumpukan. Untuk memberikan nilai konteks yang berbeda ke tumpukan individu, gunakan kunci yang berbeda untuk nilai, atau gunakan beberapa cdk synth atau cdk deploy perintah.

Nilai konteks yang diteruskan dari baris perintah selalu string. Jika nilai biasanya dari beberapa jenis lain, kode Anda harus siap untuk mengonversi atau mengurai nilai. Anda mungkin memiliki nilai konteks non-string yang disediakan dengan cara lain (misalnya, dalam`cdk.context.json`). Untuk memastikan nilai semacam ini berfungsi seperti yang diharapkan, konfirmasikan bahwa nilainya adalah string sebelum mengonversinya.

Contoh

Berikut ini adalah contoh penggunaan VPC Amazon yang ada menggunakan AWS CDK konteks.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import { Construct } from 'constructs';

export class ExistsVpcStack extends cdk.Stack {

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {

    super(scope, id, props);

    const vpcid = this.node.tryGetContext('vpcid');
    const vpc = ec2.Vpc.fromLookup(this, 'VPC', {
      vpcId: vpcid,
    });

    const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

    new cdk.CfnOutput(this, 'publicsubnets', {
      value: pubsubnets.subnetIds.toString(),
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const ec2 = require('aws-cdk-lib/aws-ec2');

class ExistsVpcStack extends cdk.Stack {

  constructor(scope, id, props) {

    super(scope, id, props);

    const vpcid = this.node.tryGetContext('vpcid');
    const vpc = ec2.Vpc.fromLookup(this, 'VPC', {
      vpcId: vpcid
```

```
    });

    const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

    new cdk.CfnOutput(this, 'publicsubnets', {
      value: pubsubnets.subnetIds.toString()
    });
  }
}

module.exports = { ExistsVpcStack }
```

Python

```
import aws_cdk as cdk
import aws_cdk.aws_ec2 as ec2
from constructs import Construct

class ExistsVpcStack(cdk.Stack):

    def __init__(scope: Construct, id: str, **kwargs):

        super().__init__(scope, id, **kwargs)

        vpcid = self.node.try_get_context("vpcid")
        vpc = ec2.Vpc.from_lookup(self, "VPC", vpc_id=vpcid)

        pubsubnets = vpc.select_subnets(subnetType=ec2.SubnetType.PUBLIC)

        cdk.CfnOutput(self, "publicsubnets",
            value=pubsubnets.subnet_ids.to_string())
```

Java

```
import software.amazon.awscdk.CfnOutput;

import software.amazon.awscdk.services.ec2.Vpc;
import software.amazon.awscdk.services.ec2.VpcLookupOptions;
import software.amazon.awscdk.services.ec2.SelectedSubnets;
import software.amazon.awscdk.services.ec2.SubnetSelection;
import software.amazon.awscdk.services.ec2.SubnetType;
import software.constructs.Construct;
```

```
public class ExistsVpcStack extends Stack {
    public ExistsVpcStack(Construct context, String id) {
        this(context, id, null);
    }

    public ExistsVpcStack(Construct context, String id, StackProps props) {
        super(context, id, props);

        String vpcId = (String)this.getNode().tryGetContext("vpcid");
        Vpc vpc = (Vpc)Vpc.fromLookup(this, "VPC", VpcLookupOptions.builder()
                .vpcId(vpcId).build());

        SelectedSubnets pubSubNets = vpc.selectSubnets(SubnetSelection.builder()
                .subnetType(SubnetType.PUBLIC).build());

        CfnOutput.Builder.create(this, "publicsubnets")
                .value(pubSubNets.getSubnetIds().toString()).build();

    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.EC2;
using Constructs;

class ExistsVpcStack : Stack
{
    public ExistsVpcStack(Construct scope, string id, StackProps props) :
 base(scope, id, props)
    {
        var vpcId = (string)this.Node.TryGetContext("vpcid");
        var vpc = Vpc.FromLookup(this, "VPC", new VpcLookupOptions
        {
            VpcId = vpcId
        });

        SelectedSubnets pubSubNets = vpc.SelectSubnets([new SubnetSelection
        {
            SubnetType = SubnetType.PUBLIC
        }]);
```

```
        new CfnOutput(this, "publicsubnets", new CfnOutputProps {
            Value = pubSubNets.SubnetIds.ToString()
        });
    }
}
```

Anda dapat menggunakan cdk diff untuk melihat efek dari meneruskan nilai konteks pada baris perintah:

```
cdk diff -c vpcid=vpc-0cb9c31031d0d3e22
```

```
Stack ExistsvpcStack
Outputs
[+] Output publicsubnets publicsubnets:
 {"Value":"subnet-06e0ea7dd302d3e8f,subnet-01fc0acfb58f3128f"}
```

Nilai konteks yang dihasilkan dapat dilihat seperti yang ditunjukkan di sini.

```
cdk context -j
```

```
{
  "vpc-provider:account=123456789012:filter.vpc-id=vpc-0cb9c31031d0d3e22:region=us-
east-1": {
    "vpcId": "vpc-0cb9c31031d0d3e22",
    "availabilityZones": [
      "us-east-1a",
      "us-east-1b"
    ],
    "privateSubnetIds": [
      "subnet-03ecfc033225be285",
      "subnet-0cded5da53180ebfa"
    ],
    "privateSubnetNames": [
      "Private"
    ],
    "privateSubnetRouteTableIds": [
      "rtb-0e955393ced0ada04",
      "rtb-05602e7b9f310e5b0"
    ],
    "publicSubnetIds": [
```

```
      "subnet-06e0ea7dd302d3e8f",
      "subnet-01fc0acfb58f3128f"
    ],
    "publicSubnetNames": [
      "Public"
    ],
    "publicSubnetRouteTableIds": [
      "rtb-00d1fdfd823c82289",
      "rtb-04bb1969b42969bcb"
    ]
  }
}
```

Bendera fitur

AWS CDK Menggunakan flag fitur untuk mengaktifkan perilaku yang berpotensi melanggar dalam rilis. Bendera disimpan sebagai the section called “Konteks” nilai di `cdk.json` (atau`~/.cdk.json`). Mereka tidak dihapus oleh cdk context --clear perintah cdk context --reset atau.

Bendera fitur dinonaktifkan secara default. Proyek yang ada yang tidak menentukan bendera akan terus berfungsi seperti sebelumnya dengan AWS CDK rilis selanjutnya. Proyek baru yang dibuat menggunakan flag cdk init include yang memungkinkan semua fitur yang tersedia dalam rilis yang membuat proyek. Edit `cdk.json` untuk menonaktifkan bendera apa pun yang Anda sukai perilaku sebelumnya. Anda juga dapat menambahkan flag untuk mengaktifkan perilaku baru setelah memutakhirkan. AWS CDK

Daftar semua flag fitur saat ini dapat ditemukan di AWS CDK GitHub repositori di. `FEATURE_FLAGS.md` Lihat `CHANGELOG` dalam rilis tertentu untuk deskripsi bendera fitur baru yang ditambahkan dalam rilis itu.

Kembali ke perilaku v1

Di CDK v2, default untuk beberapa flag fitur telah diubah sehubungan dengan v1. Anda dapat menyetelnya kembali `false` untuk kembali ke perilaku AWS CDK v1 tertentu. Gunakan `cdk diff` perintah untuk memeriksa perubahan pada template yang disintesis untuk melihat apakah salah satu flag ini diperlukan.

`@aws-cdk/core:newStyleStackSynthesis`

Gunakan metode sintesis tumpukan baru, yang mengasumsikan sumber daya bootstrap dengan nama terkenal. Membutuhkan bootstrap modern, tetapi pada gilirannya memungkinkan CI/CD melalui CDK Pipelines dan penyebaran lintas akun di luar kotak.

`@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId`

Jika aplikasi Anda menggunakan beberapa kunci API Amazon API Gateway dan mengaitkannya dengan paket penggunaan.

`@aws-cdk/aws-rds:lowercaseDbIdentifier`

Jika aplikasi Anda menggunakan instance database Amazon RDS atau cluster database, dan secara eksplisit menentukan identifier untuk ini.

`@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021`

Jika aplikasi Anda menggunakan kebijakan keamanan TLS_V1_2_2019 dengan distribusi. Amazon CloudFront CDK v2 menggunakan kebijakan keamanan TLSV1.2_2021 secara default.

`@aws-cdk/core:stackRelativeExports`

Jika aplikasi Anda menggunakan beberapa tumpukan dan Anda merujuk ke sumber daya dari satu tumpukan di tumpukan lain, ini menentukan apakah jalur absolut atau relatif digunakan untuk membangun ekspor AWS CloudFormation .

`@aws-cdk/aws-lambda:recognizeVersionProps`

Jika disetel ke`false`, CDK menyertakan metadata saat mendeteksi apakah fungsi Lambda telah berubah. Ini dapat menyebabkan kegagalan penerapan ketika hanya metadata yang berubah, karena versi duplikat tidak diizinkan. Tidak perlu mengembalikan bendera ini jika Anda telah membuat setidaknya satu perubahan ke semua Fungsi Lambda di aplikasi Anda.

Sintaks untuk mengembalikan bendera ini ditampilkan di sini`cdk.json`.

```
{
  "context": {
    "@aws-cdk/core:newStyleStackSynthesis": false,
    "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": false,
    "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": false,
    "@aws-cdk/aws-rds:lowercaseDbIdentifier": false,
    "@aws-cdk/core:stackRelativeExports": false,
    "@aws-cdk/aws-lambda:recognizeVersionProps": false
```

```
  }
}
```

Aspek

Aspek adalah cara untuk menerapkan operasi ke semua konstruksi dalam lingkup tertentu. Aspek dapat memodifikasi konstruksi, seperti dengan menambahkan tag. Atau bisa memverifikasi sesuatu tentang status konstruksi, seperti memastikan bahwa semua ember dienkripsi.

Untuk menerapkan aspek ke konstruksi dan semua konstruksi dalam lingkup yang sama, panggil `Aspects.of(SCOPE).add()`dengan aspek baru, seperti yang ditunjukkan pada contoh berikut.

TypeScript

```
Aspects.of(myConstruct).add(new SomeAspect(...));
```

JavaScript

```
Aspects.of(myConstruct).add(new SomeAspect(...));
```

Python

```
Aspects.of(my_construct).add(SomeAspect(...))
```

Java

```
Aspects.of(myConstruct).add(new SomeAspect(...));
```

C#

```
Aspects.Of(myConstruct).add(new SomeAspect(...));
```

Go

```
awscdk.Aspects_Of(stack).Add(awscdk.NewTag(...))
```

AWS CDK Menggunakan aspek untuk menandai sumber daya, tetapi kerangka kerja juga dapat digunakan untuk tujuan lain. Misalnya, Anda dapat menggunakannya untuk memvalidasi atau

mengubah AWS CloudFormation sumber daya yang ditentukan untuk Anda oleh konstruksi tingkat yang lebih tinggi.

Aspek secara detail

Aspek menggunakan pola pengunjung. Aspek adalah kelas yang mengimplementasikan antarmuka berikut.

TypeScript

```
interface IAspect {
   visit(node: IConstruct): void;}
```

JavaScript

JavaScript tidak memiliki antarmuka sebagai fitur bahasa. Oleh karena itu, sebuah aspek hanyalah sebuah instance dari kelas yang memiliki `visit` metode yang menerima node yang akan dioperasikan.

Python

Python tidak memiliki antarmuka sebagai fitur bahasa. Oleh karena itu, sebuah aspek hanyalah sebuah instance dari kelas yang memiliki `visit` metode yang menerima node yang akan dioperasikan.

Java

```
public interface IAspect {
    public void visit(Construct node);
}
```

C#

```
public interface IAspect
{
    void Visit(IConstruct node);
}
```

Go

```
type IAspect interface {
```

```
  Visit(node constructs.IConstruct)
}
```

Saat Anda meneleponAspects.of(*SCOPE*).add(...), konstruksi menambahkan aspek ke daftar aspek internal. Anda dapat memperoleh daftar denganAspects.of(*SCOPE*).

Selama fase persiapan, AWS CDK panggilan `visit` metode objek untuk konstruksi dan masing-masing anaknya dalam urutan top-down.

`visit`Metode ini bebas untuk mengubah apa pun dalam konstruksi. Dalam bahasa yang diketik dengan kuat, lemparkan konstruksi yang diterima ke tipe yang lebih spesifik sebelum mengakses properti atau metode khusus konstruksi.

Aspek tidak menyebar melintasi batas-batas `Stage` konstruksi, karena mandiri dan `Stages` tidak dapat diubah setelah definisi. Terapkan aspek pada `Stage` konstruksi itu sendiri (atau lebih rendah) jika Anda ingin mereka mengunjungi konstruksi di dalam. `Stage`

Contoh

Contoh berikut memvalidasi bahwa semua bucket yang dibuat di tumpukan mengaktifkan versi. Aspek menambahkan anotasi kesalahan ke konstruksi yang gagal validasi. Hal ini mengakibatkan synth operasi gagal dan mencegah penerapan rakitan cloud yang dihasilkan.

TypeScript

```
class BucketVersioningChecker implements IAspect {
  public visit(node: IConstruct): void {
    // See that we're dealing with a CfnBucket
    if (node instanceof s3.CfnBucket) {

      // Check for versioning property, exclude the case where the property
      // can be a token (IResolvable).
      if (!node.versioningConfiguration
        || (!Tokenization.isResolvable(node.versioningConfiguration)
            && node.versioningConfiguration.status !== 'Enabled')) {
        Annotations.of(node).addError('Bucket versioning is not enabled');
      }
    }
  }
}
```

```
// Later, apply to the stack
Aspects.of(stack).add(new BucketVersioningChecker());
```

JavaScript

```
class BucketVersioningChecker {
   visit(node) {
    // See that we're dealing with a CfnBucket
    if ( node instanceof s3.CfnBucket) {

      // Check for versioning property, exclude the case where the property
      // can be a token (IResolvable).
      if (!node.versioningConfiguration
        || !Tokenization.isResolvable(node.versioningConfiguration)
            && node.versioningConfiguration.status !== 'Enabled')) {
        Annotations.of(node).addError('Bucket versioning is not enabled');
      }
    }
  }
}

// Later, apply to the stack
Aspects.of(stack).add(new BucketVersioningChecker());
```

Python

```
@jsii.implements(cdk.IAspect)
class BucketVersioningChecker:

  def visit(self, node):
    # See that we're dealing with a CfnBucket
    if isinstance(node, s3.CfnBucket):

      # Check for versioning property, exclude the case where the property
      # can be a token (IResolvable).
      if (not node.versioning_configuration or
              not Tokenization.is_resolvable(node.versioning_configuration)
                  and node.versioning_configuration.status != "Enabled"):
          Annotations.of(node).add_error('Bucket versioning is not enabled')

# Later, apply to the stack
Aspects.of(stack).add(BucketVersioningChecker())
```

Java

```
public class BucketVersioningChecker implements IAspect
{
    @Override
    public void visit(Construct node)
    {
        // See that we're dealing with a CfnBucket
        if (node instanceof CfnBucket)
        {
            CfnBucket bucket = (CfnBucket)node;
            Object versioningConfiguration = bucket.getVersioningConfiguration();
            if (versioningConfiguration == null ||
                    !Tokenization.isResolvable(versioningConfiguration.toString())
 &&
                    !versioningConfiguration.toString().contains("Enabled"))
                Annotations.of(bucket.getNode()).addError("Bucket versioning is not
 enabled");
        }
    }
}


// Later, apply to the stack
Aspects.of(stack).add(new BucketVersioningChecker());
```

C#

```
class BucketVersioningChecker : Amazon.Jsii.Runtime.Deputy.DeputyBase, IAspect
{
    public void Visit(IConstruct node)
    {
        // See that we're dealing with a CfnBucket
        if (node is CfnBucket)
        {
            var bucket = (CfnBucket)node;
            if (bucket.VersioningConfiguration is null ||
                    !Tokenization.IsResolvable(bucket.VersioningConfiguration) &&
                    !bucket.VersioningConfiguration.ToString().Contains("Enabled"))
                Annotations.Of(bucket.Node).AddError("Bucket versioning is not
 enabled");
        }
    }
```

```
}

// Later, apply to the stack
Aspects.Of(stack).add(new BucketVersioningChecker());
```

Memulai dengan AWS CDK

Mulailah AWS Cloud Development Kit (AWS CDK) dengan menginstal AWS CDK CLI dan membuat aplikasi CDK pertama Anda.

Topik

Prasyarat

Sumber daya yang direkomendasikan

Sebelum memulai dengan AWS CDK, kami merekomendasikan pemahaman dasar tentang hal-hal berikut:

- Pengantar untuk AWS CDK. Untuk mempelajari selengkapnya, lihat Apa itu AWS CDK?
- Konsep inti di balik AWS CDK. Untuk mempelajari selengkapnya, lihat AWS CDK konsep.
- Layanan AWS Yang ingin Anda kelola dengan AWS CDK.
- AWS Identity and Access Management. Untuk informasi lebih lanjut, lihat Apa itu IAM? dan Apa itu Pusat Identitas IAM?
- AWS CloudFormation karena AWS CDK memanfaatkan AWS CloudFormation layanan untuk menyediakan sumber daya yang dibuat di CDK. Untuk mempelajari selengkapnya, lihat Apa itu AWS CloudFormation?
- Bahasa pemrograman yang didukung yang Anda rencanakan untuk digunakan dengan AWS CDK.

Persiapkan lingkungan lokal Anda

Semua AWS CDK pengembang, terlepas dari bahasa pilihan Anda, memerlukan Node.js14.15.0 atau yang lebih baru. Semua bahasa pemrograman yang didukung menggunakan backend yang sama, yang berjalan pada. Node.js Kami merekomendasikan versi dalam dukungan jangka panjang aktif. Organisasi Anda mungkin memiliki rekomendasi yang berbeda.

> **Important**
>
> Node.js versi 13.0.0 hingga 13.6.0 tidak kompatibel dengan AWS CDK karena masalah kompatibilitas dengan dependensinya.

Prasyarat lain tergantung pada bahasa di mana Anda mengembangkan AWS CDK aplikasi dan adalah sebagai berikut.

TypeScript

- TypeScript 3.8 atau lebih baru () `npm -g install typescript`

JavaScript

Tidak ada persyaratan tambahan

Python

- Python 3.7 atau yang lebih baru termasuk dan `pip` `virtualenv`

Java

- Java Development Kit (JDK) 8 (alias 1.8) atau yang lebih baru
- Apache Maven 3.5 atau yang lebih baru

Java IDE direkomendasikan (kami menggunakan Eclipse dalam beberapa contoh dalam panduan ini). IDE harus dapat mengimpor proyek Maven. Periksa untuk memastikan bahwa proyek Anda diatur untuk menggunakan Java 1.8. Atur variabel lingkungan JAVA_HOME ke jalur tempat Anda menginstal JDK.

C#

.NET Core 3.1 atau yang lebih baru, atau .NET 6.0 atau yang lebih baru.

Visual Studio 2019 (edisi apa pun) atau Visual Studio Code direkomendasikan.

Go

Pergi 1.1.8 atau yang lebih baru.

Untuk informasi lebih rinci, lihat bagian Prasyarat untuk bahasa Anda:

- the section called “Di TypeScript”
- the section called “Di JavaScript”
- the section called “Dengan Python”
- the section called “Di Jawa”
- the section called “Dalam C #”
- the section called “Di Go”

> **Pengunduran bahasa pihak ketiga**
>
> Setiap versi bahasa hanya didukung sampai EOL (End Of Life) dan dapat berubah sewaktu-waktu dengan pemberitahuan sebelumnya.

Langkah 1: Buat Akun AWS

Jika Anda baru AWS, Anda harus mendaftar untuk Akun AWS dan membuat pengguna administratif. Untuk informasi selengkapnya, lihat Menyiapkan IAM di Panduan Pengguna IAM.

Ketika Anda berinteraksi AWS, Anda menentukan kredensi AWS keamanan Anda untuk memverifikasi siapa Anda dan apakah Anda memiliki izin untuk mengakses sumber daya yang Anda minta. AWS menggunakan kredensi keamanan untuk mengautentikasi dan mengotorisasi permintaan Anda. Untuk mempelajari selengkapnya, lihat kredensi AWS keamanan di Panduan Pengguna IAM.

Langkah 2: Konfigurasikan akses terprogram

Ketika mengembangkan dengan AWS CDK di lingkungan lokal Anda, Anda akan bergantung pada AWS CDK CLI untuk berinteraksi dengan Layanan AWS dan mengelola AWS sumber daya Anda. Untuk menggunakan AWS CDK CLI, Anda harus mengkonfigurasi akses terprogram. Untuk mempelajari selengkapnya tentang berbagai cara mengonfigurasi akses terprogram, lihat Otentikasi dan akses di Panduan Referensi AWS SDK dan Alat.

Untuk pengguna baru yang tidak diberi metode otentikasi oleh perusahaan mereka, sebaiknya gunakan AWS IAM Identity Center. Metode ini termasuk menginstal AWS Command Line Interface (AWS CLI) dan menggunakannya untuk konfigurasi dan masuk ke portal AWS akses. Untuk mengonfigurasi akses terprogram menggunakan IAM Identity Center, lihat Autentikasi IAM Identity

Center di AWS SDK and Tools Reference Guide. Setelah selesai, lingkungan Anda harus berisi elemen-elemen berikut:

- Itu AWS CLI, yang Anda gunakan untuk memulai sesi portal AWS akses sebelum Anda menjalankan aplikasi Anda.
- AWSconfigFile bersama yang memiliki `[default]` profil dengan serangkaian nilai konfigurasi yang dapat direferensikan dari file. AWS CDK Untuk menemukan lokasi file ini, lihat Lokasi file bersama di AWS SDK dan Panduan Referensi Alat.
- `config`File bersama menetapkan `region`pengaturan. Ini menetapkan default Wilayah AWS AWS CDK penggunaan untuk AWS permintaan.
- AWS CDK Menggunakan konfigurasi penyedia token SSO profil untuk memperoleh kredensional sebelum mengirim permintaan ke. AWS`sso_role_name`Nilai, yang merupakan peran IAM yang terhubung ke set izin Pusat Identitas IAM, harus memungkinkan akses ke yang Layanan AWS digunakan dalam aplikasi Anda.

  `config`File contoh berikut menunjukkan profil default yang diatur dengan konfigurasi penyedia token SSO. `sso_session`Pengaturan profil mengacu pada `sso-session`bagian bernama. `sso-session`Bagian ini berisi pengaturan untuk memulai sesi portal AWS akses.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Memulai sesi portal AWS akses

Sebelum mengakses Layanan AWS, Anda memerlukan sesi portal AWS akses aktif agar dapat menggunakan autentikasi IAM Identity Center AWS CDK untuk menyelesaikan kredensialnya. Bergantung pada panjang sesi yang dikonfigurasi, akses Anda pada akhirnya akan kedaluwarsa dan AWS CDK akan mengalami kesalahan otentikasi. Jalankan perintah berikut di AWS CLI untuk masuk ke portal AWS akses.

```
aws sso login
```

Jika konfigurasi penyedia token SSO Anda menggunakan profil bernama alih-alih profil default, perintahnya adalah`aws sso login --profile NAME`. Tentukan juga profil ini saat mengeluarkan cdk perintah menggunakan --profile opsi atau variabel `AWS_PROFILE` lingkungan.

Untuk menguji apakah Anda sudah memiliki sesi aktif, jalankan AWS CLI perintah berikut.

```
aws sts get-caller-identity
```

Respons terhadap perintah ini harus melaporkan akun IAM Identity Center dan set izin yang dikonfigurasi dalam `config` file bersama.

> **Note**
>
> Jika Anda sudah memiliki sesi portal AWS akses aktif dan menjalankannya`aws sso login`, Anda tidak akan diminta untuk memberikan kredensi.
> Proses masuk dapat meminta Anda untuk mengizinkan AWS CLI akses ke data Anda. Karena AWS CLI dibangun di atas SDK untuk Python, pesan izin mungkin berisi variasi nama. `botocore`

Langkah 3: Instal AWS CDKCLI

Instal secara AWS CDK CLI global menggunakan perintah Node Package Manager berikut.

```
npm install -g aws-cdk
```

> **Note**
>
> Jika Anda mendapatkan kesalahan izin, dan memiliki akses administrator di sistem Anda, cobasudo `npm install -g aws-cdk`.

Jalankan perintah berikut untuk memverifikasi instalasi yang berhasil. AWS CDK CLIHarus menampilkan nomor versi:

```
cdk --version
```

Jika Anda menerima pesan kesalahan, coba hapus instalan AWS CDK CLI dengan menjalankan yang berikut ini:

```
npm uninstall -g aws-cdk
```

Kemudian, ulangi langkah-langkah untuk menginstal ulang file. AWS CDK CLI

Jika Anda masih menerima kesalahan, hapus `node-modules` folder dari proyek saat ini dan juga dari `node-modules` folder global. Untuk menemukan folder ini, jalankan`npm config get prefix`.

Ini AWS CDK CLI akan memperoleh kredensi keamanan dari sumber yang Anda konfigurasikan pada langkah sebelumnya.

> **Note**
>
> CDK Toolkit v2 bekerja dengan proyek CDK v1 yang ada. Namun, itu tidak dapat menginisialisasi proyek CDK v1 baru. Lihat the section called “Prasyarat baru” apakah Anda harus bisa melakukan itu.

Langkah 4: Bootstrap lingkungan Anda

Setiap AWS lingkungan yang Anda rencanakan untuk menyebarkan sumber daya harus di-bootstrap.

Untuk bootstrap, jalankan yang berikut ini:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

> **Tip**
>
> Jika Anda tidak memiliki nomor AWS akun yang berguna, Anda bisa mendapatkannya dari AWS Management Console. Atau, jika Anda telah AWS CLI menginstal, perintah berikut menampilkan informasi akun default Anda, termasuk nomor akun.
>
> ```
> aws sts get-caller-identity
> ```

Jika Anda membuat profil bernama dalam AWS konfigurasi lokal Anda, Anda dapat menggunakan `--profile` opsi untuk menampilkan informasi akun untuk profil tertentu. Contoh berikut menunjukkan cara menampilkan informasi akun untuk profil prod.

```
aws sts get-caller-identity --profile prod
```

Untuk menampilkan Wilayah default, gunakan`aws configure get`.

```
aws configure get region
aws configure get region --profile prod
```

AWS CDK Alat opsional

AWS Toolkit for Visual Studio CodeIni adalah plug-in open source untuk Visual Studio Code yang membantu Anda membuat, men-debug, dan menyebarkan aplikasi. AWS Toolkit ini memberikan pengalaman terintegrasi untuk mengembangkan AWS CDK aplikasi. Ini termasuk fitur AWS CDK Explorer untuk daftar AWS CDK proyek Anda dan menelusuri berbagai komponen aplikasi CDK. Instal plug-in dan pelajari lebih lanjut tentang menggunakan AWS CDK Explorer.

Langkah selanjutnya

Sekarang setelah Anda menginstal AWS CDK CLI, gunakan untuk membangun AWS CDK aplikasi pertama Anda.

Untuk mempelajari lebih lanjut tentang menggunakan bahasa pemrograman pilihan Anda, lihatBekerja dengan AWS CDK dalam bahasa pemrograman yang didukung. AWS CDK

AWS CDK Ini adalah proyek sumber terbuka. Untuk berkontribusi, lihat Berkontribusi pada AWS Cloud Development Kit (AWS CDK).

Pelajari selengkapnya

Untuk mempelajari lebih lanjut tentang ini AWS CDK, lihat berikut ini:

- Lokakarya CDK — Lokakarya langsung yang mendalam.
- Referensi API — Jelajahi konstruksi yang tersedia untuk Layanan AWS yang akan Anda gunakan.

- Membangun Hub — Temukan konstruksi dari komunitas CDK.
- AWS CDK contoh — Jelajahi contoh kode AWS CDK proyek.

AWS CDK Aplikasi pertama Anda

Mulailah menggunakan AWS Cloud Development Kit (AWS CDK) dengan membuat aplikasi CDK pertama Anda.

Sebelum memulai tutorial ini, kami sarankan Anda menyelesaikan yang berikut ini:

- Lihat Apa itu AWS CDK? untuk pengantar AWS CDK.
- Lihat AWS CDK konsep untuk mempelajari konsep inti dari AWS CDK.
- Pergi melalui prasyarat dan AWS CDK langkah-langkah pengaturan di. Memulai dengan AWS CDK

Topik

Tentang tutorial ini

Dalam tutorial ini, Anda akan membuat dan menyebarkan AWS CDK aplikasi sederhana. Aplikasi ini berisi satu tumpukan dengan satu sumber daya bucket Amazon Simple Storage Service (Amazon S3). Melalui tutorial ini, Anda akan mempelajari hal-hal berikut:

- Struktur suatu AWS CDK proyek.

- Cara membuat AWS CDK aplikasi.
- Cara menggunakan AWS Construct Library untuk menentukan aplikasi, tumpukan, dan AWS sumber daya.
- Cara menggunakan CDK CLI untuk mensintesis, membedakan, menyebarkan, dan menghapus aplikasi CDK Anda.
- Cara memodifikasi dan menerapkan kembali aplikasi CDK Anda untuk memperbarui sumber daya yang Anda gunakan.

Alur kerja AWS CDK pengembangan standar terdiri dari langkah-langkah berikut:

1. Buat AWS CDK aplikasi Anda — Di sini, Anda akan menggunakan template yang disediakan oleh AWS CDK CLI.
2. Tentukan tumpukan dan sumber daya Anda — Gunakan konstruksi untuk menentukan tumpukan dan AWS sumber daya dalam aplikasi Anda.
3. Bangun aplikasi Anda — Langkah ini opsional. AWS CDK CLISecara otomatis melakukan langkah ini jika perlu. Melakukan langkah ini disarankan untuk mengidentifikasi sintaks dan kesalahan tipe.
4. Sintesis tumpukan Anda — Langkah ini membuat AWS CloudFormation template untuk setiap tumpukan di aplikasi Anda. Langkah ini berguna untuk mengidentifikasi kesalahan logis dalam AWS sumber daya yang Anda tentukan.
5. Menerapkan aplikasi Anda — Terapkan ke AWS lingkungan Anda menggunakan AWS CloudFormation untuk menyediakan sumber daya Anda. Selama penerapan, Anda akan mengidentifikasi masalah izin apa pun dengan aplikasi Anda.

Melalui alur kerja biasa, Anda akan kembali dan mengulangi langkah-langkah sebelumnya untuk memodifikasi atau men-debug aplikasi Anda.

Kami menyarankan Anda menggunakan kontrol versi untuk AWS CDK proyek Anda.

Langkah 1: Buat aplikasi

Aplikasi CDK harus berada di direktorinya sendiri, dengan dependensi modul lokalnya sendiri. Di mesin pengembangan Anda, buat direktori baru. Berikut ini adalah contoh yang membuat `hello-cdk` direktori baru:

```
$ mkdir hello-cdk
$ cd hello-cdk
```

> **Important**
>
> Pastikan untuk memberi nama direktori proyek Anda`hello-cdk`, persis seperti yang ditunjukkan di sini. Template AWS CDK proyek menggunakan nama direktori untuk menamai hal-hal dalam kode yang dihasilkan. Jika Anda menggunakan nama yang berbeda, kode dalam tutorial ini tidak akan berfungsi.

Selanjutnya, dari direktori baru Anda, inisialisasi aplikasi dengan menggunakan cdk init perintah. Tentukan `app` template dan bahasa pemrograman pilihan Anda dengan `--language` opsi. Berikut ini adalah contohnya:

TypeScript

```
cdk init app --language typescript
```

JavaScript

```
cdk init app --language javascript
```

Python

```
cdk init app --language python
```

Setelah aplikasi dibuat, masukkan juga dua perintah berikut. Ini mengaktifkan lingkungan virtual Python aplikasi dan menginstal dependensi AWS CDK inti.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
cdk init app --language java
```

Jika Anda menggunakan IDE, Anda sekarang dapat membuka atau mengimpor proyek. Di Eclipse, misalnya, pilih File > Import > Maven > Existing Maven Projects. Pastikan bahwa pengaturan proyek diatur untuk menggunakan Java 8 (1.8).

C#

```
cdk init app --language csharp
```

Jika Anda menggunakan Visual Studio, buka file solusi di `src` direktori.

Go

```
cdk init app --language go
```

Setelah aplikasi dibuat, masukkan juga perintah berikut untuk menginstal modul AWS Construct Library yang dibutuhkan aplikasi.

```
go get
```

cdk initPerintah membuat sejumlah file dan folder di dalam `hello-cdk` direktori untuk membantu Anda mengatur kode sumber untuk AWS CDK aplikasi Anda. Secara kolektif, ini disebut AWS CDK proyek Anda. Luangkan waktu sejenak untuk menjelajahi proyek CDK.

Jika Anda telah Git menginstal, setiap proyek yang Anda buat menggunakan juga cdk init diinisialisasi sebagai Git repositori.

Langkah 2: Bangun aplikasi

Di sebagian besar lingkungan pemrograman, Anda membangun atau mengkompilasi kode setelah melakukan perubahan. Ini tidak diperlukan AWS CDK karena CDK CLI akan secara otomatis melakukan langkah ini. Namun, Anda masih dapat membangun secara manual ketika Anda ingin menangkap kesalahan sintaks dan mengetik. Berikut ini adalah contohnya:

TypeScript

```
npm run build
```

JavaScript

Tidak ada langkah membangun yang diperlukan.

Python

Tidak ada langkah membangun yang diperlukan.

Java

```
mvn compile -q
```

Atau tekan Control-B di Eclipse (IDE Java lainnya mungkin berbeda)

C#

```
dotnet build src
```

Atau tekan F6 di Visual Studio

Go

```
go build
```

Langkah 3: Buat daftar tumpukan di aplikasi

Pastikan aplikasi Anda telah dibuat dengan benar dengan mencantumkan tumpukan di aplikasi Anda. Jalankan hal berikut:

```
cdk ls
```

Output harus ditampilkan`HelloCdkStack`. Jika Anda tidak melihat output ini, verifikasi bahwa Anda berada di direktori kerja yang benar dari proyek Anda dan coba lagi. Jika Anda masih tidak melihat tumpukan Anda, ulangi the section called “Langkah 1: Buat aplikasi” dan coba lagi.

Langkah 4: Tambahkan bucket Amazon S3

Pada titik ini, aplikasi CDK Anda berisi satu tumpukan. Selanjutnya, Anda akan menentukan sumber daya bucket Amazon Simple Storage Service (Amazon S3) dalam stack Anda. Untuk melakukan ini, Anda akan mengimpor dan menggunakan konstruksi `Bucket` L2 dari Construct Library AWS .

Ubah aplikasi CDK Anda dengan mengimpor `Bucket` konstruksi dan menentukan sumber daya bucket Amazon S3 Anda. Berikut ini adalah contohnya:

TypeScript

Dalam `lib/hello-cdk-stack.ts`:

```
import * as cdk from 'aws-cdk-lib';
```

```
import { aws_s3 as s3 } from 'aws-cdk-lib';

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

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}
```

JavaScript

Dalam `lib/hello-cdk-stack.js`:

```
const cdk = require('aws-cdk-lib');
const s3 = require('aws-cdk-lib/aws-s3');

class HelloCdkStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}

module.exports = { HelloCdkStack }
```

Python

Dalam `hello_cdk/hello_cdk_stack.py`:

```
import aws_cdk as cdk
import aws_cdk.aws_s3 as s3

class HelloCdkStack(cdk.Stack):

    def __init__(self, scope: cdk.App, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
```

```
        bucket = s3.Bucket(self, "MyFirstBucket", versioned=True)
```

Java

Dalam `src/main/java/com/myorg/HelloCdkStack.java`:

```
package com.myorg;

import software.amazon.awscdk.*;
import software.amazon.awscdk.services.s3.Bucket;

public class HelloCdkStack extends Stack {
    public HelloCdkStack(final App scope, final String id) {
        this(scope, id, null);
    }

    public HelloCdkStack(final App scope, final String id, final StackProps props) {
        super(scope, id, props);

        Bucket.Builder.create(this, "MyFirstBucket")
            .versioned(true).build();
    }
}
```

C#

Dalam `src/HelloCdk/HelloCdkStack.cs`:

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;

namespace HelloCdk
{
    public class HelloCdkStack : Stack
    {
        public HelloCdkStack(App scope, string id, IStackProps props=null) :
 base(scope, id, props)
        {
            new Bucket(this, "MyFirstBucket", new BucketProps
            {
                Versioned = true
            });
        }
```

```
    }
}
```

Go

Dalam `hello-cdk.go`:

```
package main

import (
 "github.com/aws/aws-cdk-go/awscdk/v2"
 "github.com/aws/aws-cdk-go/awscdk/v2/awss3"
 "github.com/aws/constructs-go/constructs/v10"
 "github.com/aws/jsii-runtime-go"
)

type HelloCdkStackProps struct {
 awscdk.StackProps
}

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

 awss3.NewBucket(stack, jsii.String("MyFirstBucket"), &awss3.BucketProps{
  Versioned: jsii.Bool(true),
 })

 return stack
}

func main() {
 defer jsii.Close()

 app := awscdk.NewApp(nil)

 NewHelloCdkStack(app, "HelloCdkStack", &HelloCdkStackProps{
  awscdk.StackProps{
   Env: env(),
  },
```

```
 })

 app.Synth(nil)
}

func env() *awscdk.Environment {
 return nil
}
```

Mari kita lihat lebih dekat `Bucket` konstruksinya. Seperti semua konstruksi, `Bucket` kelas mengambil tiga parameter:

- scope — Mendefinisikan `Stack` kelas sebagai induk dari `Bucket` konstruksi. Semua konstruksi yang mendefinisikan AWS sumber daya dibuat dalam lingkup tumpukan. Anda dapat menentukan konstruksi di dalam konstruksi, membuat hierarki (pohon). Di sini, dan dalam banyak kasus, ruang lingkupnya adalah `this` (`self`dalamPython).
- Id — ID logis dari `Bucket` dalam AWS CDK aplikasi Anda. ID ini, ditambah hash berdasarkan lokasi bucket di dalam tumpukan, secara unik mengidentifikasi bucket selama penerapan. Ini AWS CDK juga mereferensikan ID ini saat Anda memperbarui build di aplikasi dan menerapkan ulang untuk memperbarui sumber daya yang diterapkan. Di sini, ID logis Anda adalah`MyFirstBucket`. Bucket juga dapat memiliki nama, ditentukan dengan `bucketName` properti. Ini berbeda dengan ID logis.
- props — Sebuah bundel nilai yang menentukan properti bucket. Di sini Anda mendefinisikan `versioned` properti sebagai`true`, yang memungkinkan pembuatan versi untuk file di bucket.

  Alat peraga diwakili secara berbeda dalam bahasa yang didukung oleh. AWS CDK

  - Dalam TypeScript dan JavaScript, `props` adalah argumen tunggal dan Anda meneruskan objek yang berisi properti yang diinginkan.
  - Dalam Python, props dilewatkan sebagai argumen kata kunci.
  - Di Jawa, Builder disediakan untuk melewati alat peraga. Ada dua: satu untuk`BucketProps`, dan yang kedua `Bucket` untuk membiarkan Anda membangun konstruksi dan objek alat peraga dalam satu langkah. Kode ini menggunakan yang terakhir.
  - Di C #, Anda membuat instance `BucketProps` objek menggunakan penginisialisasi objek dan meneruskannya sebagai parameter ketiga.

  Jika alat peraga konstruksi bersifat opsional, Anda dapat menghilangkan parameter sepenuhnya. `props`

Semua konstruksi mengambil tiga argumen yang sama ini, jadi mudah untuk tetap berorientasi saat Anda belajar tentang yang baru. Dan seperti yang Anda duga, Anda dapat mensubkelas konstruksi apa pun untuk memperluasnya sesuai dengan kebutuhan Anda, atau jika Anda ingin mengubah defaultnya.

Langkah 5: Sintesis template AWS CloudFormation

Sintesis AWS CloudFormation template untuk aplikasi, sebagai berikut:

```
cdk synth
```

Jika aplikasi berisi lebih dari satu tumpukan, Anda harus menentukan tumpukan mana yang akan disintesis. Karena aplikasi Anda berisi satu tumpukan, CDK CLI secara otomatis mendeteksi tumpukan yang akan disintesis.

Jika Anda tidak menjalankan`cdk synth`, CDK CLI akan secara otomatis melakukan langkah ini saat Anda menerapkan. Namun, kami menyarankan Anda menjalankan langkah ini sebelum setiap penerapan.

> **Tip**
>
> Jika Anda menerima kesalahan seperti`--app is required ...`, periksa direktori tempat Anda menjalankan CLI perintah CDK. Anda harus berada di direktori aplikasi utama Anda.

`cdk synth`Perintah menjalankan aplikasi Anda. Ini membuat AWS CloudFormation template untuk setiap tumpukan di aplikasi Anda. CDK CLI akan menampilkan versi template Anda yang diformat YAMB di baris perintah dan menyimpan versi template Anda yang diformat JSON di direktori. `cdk.out` Berikut ini adalah cuplikan dari output baris perintah yang menunjukkan bucket yang didefinisikan dalam template: AWS CloudFormation

```
Resources:
  MyFirstBucketB8884501:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
```

```
    Metadata:...
```

> **Note**
>
> Setiap template yang dihasilkan berisi `AWS::CDK::Metadata` sumber daya secara default. AWS CDK Tim menggunakan metadata ini untuk mendapatkan wawasan tentang AWS CDK penggunaan dan menemukan cara untuk memperbaikinya. Untuk detail, termasuk cara memilih keluar dari pelaporan versi, lihatPelaporan versi.

Template yang dihasilkan dapat digunakan melalui AWS CloudFormation konsol atau AWS CloudFormation alat penyebaran apa pun. Anda juga dapat menggunakan CDK CLI untuk menyebarkan. Pada langkah berikutnya, Anda menggunakan CDK CLI untuk menyebarkan.

Langkah 6: Menyebarkan tumpukan Anda

Untuk menerapkan tumpukan CDK Anda untuk AWS CloudFormation menggunakan CDKCLI, jalankan yang berikut ini:

```
cdk deploy
```

> **Important**
>
> Anda harus melakukan bootstrap satu kali dari lingkungan Anda AWS sebelum penerapan. Untuk petunjuk, lihat Bootstrap lingkungan Anda.

Mirip dengan`cdk synth`, Anda tidak perlu menentukan AWS CDK tumpukan karena aplikasi berisi satu tumpukan.

Jika kode Anda memiliki implikasi keamanan, CDK CLI akan menampilkan ringkasan. Anda perlu mengonfirmasi mereka untuk melanjutkan penerapan. Aplikasi dalam tutorial ini tidak memiliki implikasi ini.

Setelah berjalan`cdk deploy`, CDK CLI menampilkan informasi kemajuan saat tumpukan Anda diterapkan. Setelah selesai, Anda dapat pergi ke AWS CloudFormation konsol untuk melihat `HelloCdkStack` tumpukan Anda. Anda juga dapat pergi ke konsol Amazon S3 untuk melihat sumber daya Anda`MyFirstBucket`.

Selamat! Anda telah menerapkan tumpukan pertama Anda menggunakan file. AWS CDK Selanjutnya, Anda akan memodifikasi aplikasi dan menerapkan ulang untuk memperbarui sumber daya Anda.

Langkah 7: Ubah aplikasi Anda

Pada langkah ini, Anda akan memodifikasi bucket Amazon S3 Anda dengan mengonfigurasinya agar dihapus secara otomatis saat tumpukan Anda dihapus. Modifikasi ini melibatkan perubahan `RemovalPolicy` properti bucket. Anda juga akan mengonfigurasi `autoDeleteObjects` properti untuk mengonfigurasi CDK CLI untuk menghapus objek dari ember sebelum menghancurkannya. Secara default, AWS CloudFormation tidak menghapus bucket Amazon S3 yang berisi objek.

Gunakan contoh berikut untuk memodifikasi sumber daya Anda:

TypeScript

Perbarui`lib/hello-cdk-stack.ts`.

```
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

JavaScript

Perbarui`lib/hello-cdk-stack.js`.

```
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

Python

Perbarui`hello_cdk/hello_cdk_stack.py`.

```
bucket = s3.Bucket(self, "MyFirstBucket",
    versioned=True,
    removal_policy=cdk.RemovalPolicy.DESTROY,
```

```
    auto_delete_objects=True)
```

Java

Perbaruisrc/main/java/com/myorg/HelloCdkStack.java.

```
Bucket.Builder.create(this, "MyFirstBucket")
        .versioned(true)
        .removalPolicy(RemovalPolicy.DESTROY)
        .autoDeleteObjects(true)
        .build();
```

C#

Perbaruisrc/HelloCdk/HelloCdkStack.cs.

```
new Bucket(this, "MyFirstBucket", new BucketProps
{
    Versioned = true,
    RemovalPolicy = RemovalPolicy.DESTROY,
    AutoDeleteObjects = true
});
```

Go

Perbaruihello-cdk.go.

```
   awss3.NewBucket(stack, jsii.String("MyFirstBucket"), &awss3.BucketProps{
  Versioned:         jsii.Bool(true),
  RemovalPolicy:     awscdk.RemovalPolicy_DESTROY,
  AutoDeleteObjects: jsii.Bool(true),
 })
```

Saat ini, perubahan kode Anda belum membuat pembaruan langsung ke sumber daya bucket Amazon S3 yang Anda gunakan. Kode Anda menentukan status sumber daya yang diinginkan. Untuk memodifikasi sumber daya yang Anda gunakan, Anda akan menggunakan CDK CLI untuk mensintesis status yang diinginkan ke dalam templat baru. AWS CloudFormation Kemudian, Anda akan menerapkan AWS CloudFormation template baru Anda sebagai set perubahan. Set perubahan hanya membuat perubahan yang diperlukan untuk mencapai status baru yang Anda inginkan.

Untuk melihat perubahan ini, gunakan `cdk diff` perintah. Jalankan hal berikut:

```
cdk diff
```

CDK CLI menanyakan Akun AWS akun Anda untuk AWS CloudFormation template terbaru untuk tumpukan. `HelloCdkStack` Kemudian, ia membandingkan template terbaru dengan template yang baru saja disintesis dari aplikasi Anda. Outputnya akan terlihat seperti berikut ini.

```
Stack HelloCdkStack
IAM Statement Changes
###############################################################################################
#   # Resource                    # Effect # Action                      # Principal
                  # Condition #
###############################################################################################
# + # ${Custom::S3AutoDeleteObject # Allow  # sts:AssumeRole              #
 Service:lambda.amazonaws.com #           #
#   # sCustomResourceProvider/Role #        #                           #
                  #           #
#   # .Arn}                       #        #                           #
                  #           #
###############################################################################################
# + # ${MyFirstBucket.Arn}         # Allow  # s3:DeleteObject*            # AWS:
${Custom::S3AutoDeleteOb #           #
#   # ${MyFirstBucket.Arn}/*       #        # s3:GetBucket*              #
 jectsCustomResourceProvider/ #           #
#   #                             #        # s3:GetObject*              # Role.Arn}
                  #           #
#   #                             #        # s3:List*                   #
                  #           #
###############################################################################################
IAM Policy Changes
###############################################################################################
#   # Resource                                                  # Managed Policy ARN
                            #
###############################################################################################
# + # ${Custom::S3AutoDeleteObjectsCustomResourceProvider/Ro # {"Fn::Sub":"arn:
${AWS::Partition}:iam::aws:policy/serv #
#   # le}                                                       # ice-role/
AWSLambdaBasicExecutionRole"}                #
###############################################################################################
(NOTE: There may be security-related changes not in this list. See https://github.com/
aws/aws-cdk/issues/1299)

Parameters
```

```
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3Bucket
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3BucketBF7A7F3
 {"Type":"String","Description":"S3 bucket for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3VersionKey
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3VersionKeyFAF
 {"Type":"String","Description":"S3 key for asset version
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
ArtifactHash
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392ArtifactHashE56
 {"Type":"String","Description":"Artifact hash for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}

Resources
[+] AWS::S3::BucketPolicy MyFirstBucket/Policy MyFirstBucketPolicy3243DEFD
[+] Custom::S3AutoDeleteObjects MyFirstBucket/AutoDeleteObjectsCustomResource
 MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E
[+] AWS::IAM::Role Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092
[+] AWS::Lambda::Function Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F
[~] AWS::S3::Bucket MyFirstBucket MyFirstBucketB8884501
 ## [~] DeletionPolicy
 #   ## [-] Retain
 #   ## [+] Delete
 ## [~] UpdateReplacePolicy
     ## [-] Retain
     ## [+] Delete
```

Perbedaan ini memiliki empat bagian:

- Perubahan Pernyataan IAM dan Perubahan Kebijakan IAM — Perubahan izin ini ada karena Anda menyetel `AutoDeleteObjects` properti di bucket Amazon S3. Fitur hapus otomatis menggunakan sumber daya khusus untuk menghapus objek di bucket sebelum bucket itu sendiri dihapus. Objek IAM memberikan akses kode sumber daya kustom ke bucket.

- Parameter — AWS CDK Menggunakan entri ini untuk menemukan aset AWS Lambda fungsi untuk sumber daya kustom.
- Sumber daya — Sumber daya baru dan berubah dalam tumpukan ini. Kita dapat melihat objek IAM yang disebutkan sebelumnya, sumber daya khusus, dan fungsi Lambda terkait yang ditambahkan. Kita juga dapat melihat bahwa bucket `DeletionPolicy` dan `UpdateReplacePolicy` atribut sedang diperbarui. Ini memungkinkan bucket dihapus bersama dengan tumpukan, dan diganti dengan yang baru.

Anda mungkin memperhatikan bahwa kami menentukan `RemovalPolicy` di AWS CDK aplikasi kami tetapi mendapat `DeletionPolicy` properti di AWS CloudFormation template yang dihasilkan. Hal ini karena AWS CDK menggunakan nama yang berbeda untuk properti. AWS CDK Defaultnya adalah mempertahankan bucket saat tumpukan dihapus, sedangkan AWS CloudFormation defaultnya adalah menghapusnya. Untuk informasi selengkapnya, lihat the section called “Kebijakan penghapusan”.

Untuk melihat AWS CloudFormation template baru Anda, Anda dapat menjalankancdk synth. Dengan membuat beberapa perubahan pada aplikasi CDK Anda, AWS CloudFormation template baru Anda sekarang menyertakan banyak baris kode tambahan dibandingkan dengan AWS CloudFormation template asli.

Selanjutnya, terapkan aplikasi Anda dengan menjalankan yang berikut:

```
cdk deploy
```

Kami AWS CDK akan memberi tahu Anda tentang perubahan kebijakan keamanan yang telah kami lihat di diff. Masuk y untuk menyetujui perubahan dan menyebarkan tumpukan yang diperbarui. CDK CLI akan menyebarkan tumpukan Anda untuk membuat perubahan yang Anda inginkan. Berikut ini adalah contoh output:

```
HelloCdkStack: deploying...
[0%] start: Publishing
 4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392:current
[100%] success: Published
 4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392:current
HelloCdkStack: creating CloudFormation changeset...
 0/5 | 4:32:31 PM | UPDATE_IN_PROGRESS   | AWS::CloudFormation::Stack  | HelloCdkStack
 User Initiated
```

```
 0/5 | 4:32:36 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092)
 1/5 | 4:32:36 PM | UPDATE_COMPLETE      | AWS::S3::Bucket             | MyFirstBucket
 (MyFirstBucketB8884501)
 1/5 | 4:32:36 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092) Resource creation
 Initiated
 3/5 | 4:32:54 PM | CREATE_COMPLETE      | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::Lambda::Function
     | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::Lambda::Function
     | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F) Resource creation
 Initiated
 3/5 | 4:32:57 PM | CREATE_COMPLETE      | AWS::Lambda::Function
     | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F)
 3/5 | 4:32:57 PM | CREATE_IN_PROGRESS   | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD) Resource creation Initiated
 4/5 | 4:32:57 PM | CREATE_COMPLETE      | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD)
 4/5 | 4:32:59 PM | CREATE_IN_PROGRESS   | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E)
 5/5 | 4:33:06 PM | CREATE_IN_PROGRESS   | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E) Resource creation Initiated
 5/5 | 4:33:06 PM | CREATE_COMPLETE      | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E)
 5/5 | 4:33:08 PM | UPDATE_COMPLETE_CLEA | AWS::CloudFormation::Stack  | HelloCdkStack
 6/5 | 4:33:09 PM | UPDATE_COMPLETE      | AWS::CloudFormation::Stack  | HelloCdkStack

 #  HelloCdkStack

Stack ARN:
```

```
arn:aws:cloudformation:REGION:ACCOUNT:stack/HelloCdkStack/UNIQUE-ID
```

Langkah 8: Menghancurkan sumber daya aplikasi

Sekarang setelah Anda menyelesaikan tutorial ini, Anda dapat menghapus AWS CloudFormation tumpukan yang digunakan dan semua sumber daya yang terkait dengannya. Ini adalah praktik yang baik untuk meminimalkan biaya yang tidak perlu dan menjaga lingkungan Anda tetap bersih. Jalankan hal berikut:

```
cdk destroy
```

Masukkan y untuk menyetujui perubahan dan menghapus tumpukan Anda.

> **Note**
>
> Jika Anda tidak mengubah bucket`RemovalPolicy`, penghapusan tumpukan akan berhasil diselesaikan, tetapi bucket akan menjadi yatim piatu (tidak lagi terkait dengan tumpukan).

Langkah selanjutnya

Selamat! Anda telah menyelesaikan tutorial ini dan telah menggunakan AWS CDK untuk berhasil membuat, memodifikasi, dan menghapus sumber daya di AWS Cloud. Anda sekarang siap untuk mulai menggunakan AWS CDK.

Untuk mempelajari lebih lanjut tentang menggunakan bahasa pemrograman pilihan Anda, lihatBekerja dengan AWS CDK dalam bahasa pemrograman yang didukung. AWS CDK

Untuk sumber daya tambahan, lihat berikut ini:

- Cobalah Lokakarya CDK untuk tur yang lebih mendalam yang melibatkan proyek yang lebih kompleks.
- Selami lebih dalam konsep sepertithe section called “Lingkungan”,the section called “Aset”,the section called “Izin”,the section called “Konteks”,the section called “Parameter-parameter”, danthe section called “Menyesuaikan konstruksi”.
- Lihat referensi API untuk mulai menjelajahi konstruksi CDK yang tersedia untuk layanan favorit AWS Anda.
- Kunjungi Construct Hub untuk menemukan konstruksi yang dibuat oleh AWS dan lainnya.

- Jelajahi Contoh menggunakan AWS CDK.

AWS CDK Ini adalah proyek sumber terbuka. Untuk berkontribusi, lihat Berkontribusi pada AWS Cloud Development Kit (AWS CDK).

Migrasi dari AWS CDK v1 ke v2 AWS CDK

Versi 2 AWS Cloud Development Kit (AWS CDK) dirancang untuk membuat infrastruktur penulisan sebagai kode dalam bahasa pemrograman pilihan Anda lebih mudah. Topik ini menjelaskan perubahan antara v1 dan v2 dari. AWS CDK

> **Tip**
>
> Untuk mengidentifikasi tumpukan yang digunakan dengan AWS CDK v1, gunakan utilitas awscdk-v1-stack-finder.

Perubahan utama dari AWS CDK v1 ke CDK v2 adalah sebagai berikut.

- AWS CDK v2 mengkonsolidasikan bagian stabil dari AWS Construct Library, termasuk pustaka inti, ke dalam satu paket,. `aws-cdk-lib` Pengembang tidak perlu lagi menginstal paket tambahan untuk AWS layanan individual yang mereka gunakan. Pendekatan paket tunggal ini juga berarti Anda tidak perlu menyinkronkan versi berbagai paket perpustakaan CDK.

  Konstruksi L1 (CFNXXxx), yang mewakili sumber daya tepat yang tersedia di AWS CloudFormation, selalu dianggap stabil dan termasuk dalam. `aws-cdk-lib`
- Modul eksperimental, di mana kami masih bekerja dengan komunitas untuk mengembangkan konstruksi L2 atau L3 baru, tidak termasuk dalam. `aws-cdk-lib` Sebaliknya, mereka didistribusikan sebagai paket individual. Paket eksperimental diberi nama dengan `alpha` akhiran dan nomor versi semantik. Nomor versi semantik cocok dengan versi pertama dari AWS Construct Library yang kompatibel dengannya, juga dengan akhiran. `alpha` Konstruksi pindah ke `aws-cdk-lib` setelah ditunjuk stabil, memungkinkan Perpustakaan Konstruksi utama untuk mematuhi versi semantik yang ketat.

  Stabilitas ditentukan pada tingkat layanan. Misalnya, jika kita mulai membuat satu atau lebih konstruksi L2 untuk Amazon AppFlow, yang pada tulisan ini hanya memiliki konstruksi L1, mereka pertama kali muncul dalam modul bernama. `@aws-cdk/aws-appflow-alpha` Kemudian, mereka pindah ke `aws-cdk-lib` ketika kita merasa bahwa konstruksi baru memenuhi kebutuhan mendasar pelanggan.

  Setelah modul ditetapkan stabil dan dimasukkan ke dalam`aws-cdk-lib`, API baru ditambahkan menggunakan konvensi “BETan” yang dijelaskan dalam bullet berikutnya.

Versi baru dari setiap modul eksperimental dirilis dengan setiap rilis AWS CDK. Namun, untuk sebagian besar, mereka tidak perlu disinkronkan. Anda dapat memutakhirkan `aws-cdk-lib` atau modul eksperimental kapan pun Anda mau. Pengecualiannya adalah ketika dua atau lebih modul eksperimental terkait saling bergantung, mereka harus versi yang sama.

- Untuk modul stabil yang fungsionalitas baru ditambahkan, API baru (baik konstruksi yang sama sekali baru atau metode atau properti baru pada konstruksi yang ada) menerima `Beta1` akhiran saat pekerjaan sedang berlangsung. (Diikuti oleh `Beta2``Beta3`,, dan seterusnya ketika melanggar perubahan diperlukan.) Versi API tanpa akhiran ditambahkan saat API ditetapkan stabil. Semua metode kecuali yang terbaru (baik beta atau final) kemudian tidak digunakan lagi.

  Misalnya, jika kita menambahkan metode baru `grantPower()` ke konstruksi, awalnya muncul sebagai`grantPowerBeta1()`. Jika perubahan yang melanggar diperlukan (misalnya, parameter atau properti baru yang diperlukan), versi metode berikutnya akan diberi nama`grantPowerBeta2()`, dan seterusnya. Ketika pekerjaan selesai dan API diselesaikan, metode `grantPower()` (tanpa akhiran) ditambahkan, dan metode BETan tidak digunakan lagi.

  Semua API beta tetap berada di Perpustakaan Konstruksi hingga rilis versi utama berikutnya (3.0), dan tanda tangannya tidak akan berubah. Anda akan melihat peringatan penghentian jika Anda menggunakannya, jadi Anda harus pindah ke versi final API secepatnya. Namun, tidak ada rilis AWS CDK 2.x future yang akan merusak aplikasi Anda.
- `Construct`Kelas telah diekstraksi dari AWS CDK ke dalam perpustakaan terpisah, bersama dengan jenis terkait. Hal ini dilakukan untuk mendukung upaya penerapan Construct Programming Model ke domain lain. Jika Anda menulis konstruksi Anda sendiri atau menggunakan API terkait, Anda harus mendeklarasikan `constructs` modul sebagai dependensi dan membuat perubahan kecil pada impor Anda. Jika Anda menggunakan fitur-fitur canggih, seperti menghubungkan ke siklus hidup aplikasi CDK, lebih banyak perubahan mungkin diperlukan. Untuk detail selengkapnya, lihat RFC.
- Properti, metode, dan tipe yang tidak digunakan lagi di AWS CDK v1.x dan Perpustakaan Konstruksinya telah dihapus sepenuhnya dari CDK v2 API. Dalam sebagian besar bahasa yang didukung, API ini menghasilkan peringatan di bawah v1.x, sehingga Anda mungkin telah bermigrasi ke API pengganti. Daftar lengkap API usang di CDK v1.x tersedia di. GitHub
- Perilaku yang dijaga oleh flag fitur di AWS CDK v1.x diaktifkan secara default di CDK v2. Bendera fitur sebelumnya tidak lagi diperlukan, dan dalam banyak kasus mereka tidak didukung. Beberapa masih tersedia untuk memungkinkan Anda kembali ke perilaku CDK v1 dalam keadaan yang sangat spesifik. Untuk informasi selengkapnya, lihat the section called “Memperbarui bendera fitur”.

- Dengan CDK v2, lingkungan yang Anda terapkan harus di-bootstrap menggunakan tumpukan bootstrap modern. Tumpukan bootstrap lama (default di bawah v1) tidak lagi didukung. CDK v2 selanjutnya membutuhkan versi baru dari tumpukan modern. Untuk meng-upgrade lingkungan yang ada, re-bootstrap mereka. Tidak perlu lagi mengatur flag fitur atau variabel lingkungan apa pun untuk menggunakan tumpukan bootstrap modern.

> **Important**
>
> Template bootstrap modern secara efektif memberikan izin yang tersirat oleh `--cloudformation-execution-policies` ke AWS akun mana pun dalam daftar. `--trust` Secara default, ini memperluas izin untuk membaca dan menulis ke sumber daya apa pun di akun bootstrap. Pastikan untuk mengonfigurasi tumpukan bootstrap dengan kebijakan dan akun tepercaya yang nyaman bagi Anda.

Prasyarat baru

Sebagian besar persyaratan untuk AWS CDK v2 sama dengan untuk AWS CDK v1.x. Persyaratan tambahan tercantum di sini.

- Untuk TypeScript pengembang, TypeScript 3.8 atau yang lebih baru diperlukan.
- Versi baru CDK Toolkit diperlukan untuk digunakan dengan CDK v2. Sekarang CDK v2 tersedia secara umum, v2 adalah versi default saat menginstal CDK Toolkit. Ini kompatibel dengan proyek CDK v1, jadi Anda tidak perlu menginstal versi sebelumnya kecuali Anda ingin membuat proyek CDK v1. Untuk meningkatkan, masalah`npm install -g aws-cdk`.

Upgrade dari AWS CDK v2 Developer Preview

Jika Anda menggunakan Pratinjau Pengembang CDK v2, Anda memiliki dependensi dalam proyek Anda pada versi Kandidat Rilis AWS CDK, seperti. `2.0.0-rc1` Perbarui ini ke`2.0.0`, lalu perbarui modul yang diinstal di proyek Anda.

TypeScript

`npm install` atau `yarn install`

JavaScript

`npm install` atau `yarn install`

Python

```
python -m pip install -r requirements.txt
```

Java

```
mvn package
```

C#

```
dotnet restore
```

Go

```
go get
```

Setelah memperbarui dependensi Anda, masalah `npm update -g aws-cdk` untuk memperbarui CDK Toolkit ke versi rilis.

Bermigrasi dari AWS CDK v1 ke CDK v2

Untuk memigrasikan aplikasi Anda ke AWS CDK v2, perbarui flag fitur terlebih dahulu. `cdk.json` Kemudian perbarui dependensi dan impor aplikasi Anda seperlunya untuk bahasa pemrograman yang ditulisnya.

Memperbarui ke v1 terbaru

Kami melihat sejumlah pelanggan meningkatkan dari versi lama AWS CDK v1 ke versi terbaru v2 dalam satu langkah. Meskipun tentu saja mungkin untuk melakukan itu, Anda berdua akan meningkatkan beberapa tahun perubahan (yang sayangnya mungkin tidak semua memiliki jumlah pengujian evolusi yang sama yang kita miliki saat ini), serta meningkatkan seluruh versi dengan default baru dan organisasi kode yang berbeda.

Untuk pengalaman upgrade yang paling aman dan untuk lebih mudah mendiagnosis sumber dari setiap perubahan yang tidak terduga, kami sarankan Anda memisahkan dua langkah tersebut: pertama upgrade ke versi v1 terbaru, kemudian beralih ke v2 sesudahnya.

Memperbarui bendera fitur

Hapus flag fitur v1 berikut dari `cdk.json` jika ada, karena ini semua aktif secara default di AWS CDK v2. Jika efek lama mereka penting untuk infrastruktur Anda, Anda perlu membuat perubahan kode sumber. Lihat daftar bendera GitHub untuk informasi selengkapnya.

- `@aws-cdk/core:enableStackNameDuplicates`
- `aws-cdk:enableDiffNoFail`
- `@aws-cdk/aws-ecr-assets:dockerIgnoreSupport`
- `@aws-cdk/aws-secretsmanager:parseOwnedSecretName`
- `@aws-cdk/aws-kms:defaultKeyPolicies`
- `@aws-cdk/aws-s3:grantWriteWithoutAcl`
- `@aws-cdk/aws-efs:defaultEncryptionAtRest`

Beberapa flag fitur v1 dapat diatur untuk kembali ke `false` perilaku AWS CDK v1 tertentu; lihat the section called “Kembali ke perilaku v1” atau daftar untuk referensi lengkap. GitHub

Untuk kedua jenis flag, gunakan `cdk diff` perintah untuk memeriksa perubahan pada template yang disintesis untuk melihat apakah perubahan pada salah satu flag ini memengaruhi infrastruktur Anda.

Kompatibilitas CDK Toolkit

CDK v2 membutuhkan v2 atau yang lebih baru dari CDK Toolkit. Versi ini kompatibel dengan aplikasi CDK v1. Oleh karena itu, Anda dapat menggunakan satu versi CDK Toolkit yang diinstal secara global dengan semua AWS CDK proyek Anda, apakah mereka menggunakan v1 atau v2. Pengecualian adalah bahwa CDK Toolkit v2 hanya membuat proyek CDK v2.

Jika Anda perlu membuat proyek CDK v1 dan v2, jangan instal CDK Toolkit v2 secara global. (Hapus jika Anda sudah menginstalnya:`npm remove -g aws-cdk`.) Untuk menjalankan CDK Toolkit, gunakan npx untuk menjalankan v1 atau v2 dari CDK Toolkit sesuai keinginan.

```
npx aws-cdk@1.x init app --language typescript
```

```
npx aws-cdk@2.x init app --language typescript
```

> **Tip**
>
> Siapkan alias baris perintah sehingga Anda dapat menggunakan cdk1 perintah cdk dan untuk memanggil versi CDK Toolkit yang diinginkan.
>
> macOS/Linux
>
> ```
> alias cdk1="npx aws-cdk@1.x"
> alias cdk="npx aws-cdk@2.x"
> ```
>
> Windows
>
> ```
> doskey cdk1=npx aws-cdk@1.x $*
> doskey cdk=npx aws-cdk@2.x $*
> ```

Memperbarui dependensi dan impor

Perbarui dependensi aplikasi Anda, lalu instal paket baru. Terakhir, perbarui impor dalam kode Anda.

TypeScript

Aplikasi

Untuk aplikasi CDK, perbarui `package.json` sebagai berikut. Hapus dependensi pada modul stabil individual bergaya v1 dan buat versi terendah yang `aws-cdk-lib` Anda perlukan untuk aplikasi Anda (2.0.0 di sini).

Konstruksi eksperimental disediakan dalam paket terpisah, berversi independen dengan nama yang diakhiri `alpha` dan nomor versi alfa. Nomor versi alfa sesuai `aws-cdk-lib` dengan rilis pertama yang kompatibel. Di sini, kami telah menyematkan `aws-codestar` ke v2.0.0-alpha.1.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
    "constructs": "^10.0.0"
```

```
  }
}
```

Membangun perpustakaan

Untuk membangun pustaka, buat versi terendah yang `aws-cdk-lib` Anda perlukan untuk aplikasi Anda (2.0.0 di sini) dan perbarui sebagai berikut. `package.json`

Perhatikan yang `aws-cdk-lib` muncul baik sebagai ketergantungan rekan dan ketergantungan dev.

```
{
  "peerDependencies": {
    "aws-cdk-lib": "^2.0.0",
    "constructs": "^10.0.0"
  },
  "devDependencies": {
    "aws-cdk-lib": "^2.0.0",
    "constructs": "^10.0.0",
    "typescript": "~3.9.0"
  }
}
```

> **Note**
>
> Anda harus melakukan bump versi utama pada nomor versi perpustakaan Anda saat merilis pustaka yang kompatibel dengan v2, karena ini adalah perubahan besar bagi konsumen perpustakaan. Tidak mungkin mendukung CDK v1 dan v2 dengan satu pustaka. Untuk terus mendukung pelanggan yang masih menggunakan v1, Anda dapat mempertahankan rilis sebelumnya secara paralel, atau membuat paket baru untuk v2. Terserah Anda berapa lama Anda ingin terus mendukung pelanggan AWS CDK v1. Anda mungkin mengambil isyarat dari siklus hidup CDK v1 itu sendiri, yang memasuki pemeliharaan pada 1 Juni 2022 dan akan mencapai end-of-life pada 1 Juni 2023. Untuk detail selengkapnya, lihat Kebijakan AWS CDK Pemeliharaan.

Pustaka dan aplikasi

Instal dependensi baru dengan menjalankan `npm install` atau. `yarn install`

Ubah impor Anda untuk mengimpor `Construct` dari `constructs` modul baru, tipe inti seperti `App` dan `Stack` dari tingkat atas`aws-cdk-lib`, dan modul Construct Library yang stabil untuk layanan yang Anda gunakan dari ruang nama di bawah. `aws-cdk-lib`

```
import { Construct } from 'constructs';
import { App, Stack } from 'aws-cdk-lib';                 // core constructs
import { aws_s3 as s3 } from 'aws-cdk-lib';               // stable module
import * as codestar from '@aws-cdk/aws-codestar-alpha';  // experimental module
```

JavaScript

Perbarui `package.json` sebagai berikut. Hapus dependensi pada modul stabil individual bergaya v1 dan buat versi terendah yang `aws-cdk-lib` Anda perlukan untuk aplikasi Anda (2.0.0 di sini).

Konstruksi eksperimental disediakan dalam paket terpisah, berversi independen dengan nama yang diakhiri `alpha` dan nomor versi alfa. Nomor versi alfa sesuai `aws-cdk-lib` dengan rilis pertama yang kompatibel. Di sini, kami telah menyematkan `aws-codestar` ke v2.0.0-alpha.1.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
    "constructs": "^10.0.0"
  }
}
```

Instal dependensi baru dengan menjalankan `npm install` atau. `yarn install`

Ubah impor aplikasi Anda untuk melakukan hal berikut:

- Impor `Construct` dari `constructs` modul baru
- Impor jenis inti`Stack`, seperti `App` dan, dari tingkat atas `aws-cdk-lib`
- Impor modul AWS Construct Library dari ruang nama di bawah `aws-cdk-lib`

```
const { Construct } = require('constructs');
const { App, Stack } = require('aws-cdk-lib');              // core constructs
const s3 = require('aws-cdk-lib').aws_s3;                   // stable module
const codestar = require('@aws-cdk/aws-codestar-alpha');    // experimental module
```

Python

Perbarui `requirements.txt` atau `install_requires` definisi `setup.py` sebagai berikut. Hapus dependensi pada modul stabil individual bergaya v1.

Konstruksi eksperimental disediakan dalam paket terpisah, berversi independen dengan nama yang diakhiri `alpha` dan nomor versi alfa. Nomor versi alfa sesuai `aws-cdk-lib` dengan rilis pertama yang kompatibel. Di sini, kami telah menyematkan `aws-codestar` ke v2.0.0alpha1.

```
install_requires=[
     "aws-cdk-lib>=2.0.0",
     "constructs>=10.0.0",
     "aws-cdk.aws-codestar-alpha>=2.0.0alpha1",
     # ...
],
```

> **Tip**
>
> Copot pemasangan versi AWS CDK modul lain yang sudah diinstal di lingkungan virtual aplikasi Anda menggunakan`pip uninstall`. Kemudian Instal dependensi baru dengan. `python -m pip install -r requirements.txt`

Ubah impor aplikasi Anda untuk melakukan hal berikut:

- Impor `Construct` dari `constructs` modul baru
- Impor jenis inti`Stack`, seperti App dan, dari tingkat atas `aws_cdk`
- Impor modul AWS Construct Library dari ruang nama di bawah `aws_cdk`

```
from constructs import Construct
from aws_cdk import App, Stack                    # core constructs
from aws_cdk import aws_s3 as s3                  # stable module
import aws_cdk.aws_codestar_alpha as codestar     # experimental module

# ...

class MyConstruct(Construct):
    # ...
```

```
class MyStack(Stack):
    # ...

s3.Bucket(...)
```

Java

Dipom.xml, hapus semua software.amazon.awscdk dependensi untuk modul stabil dan ganti dengan dependensi pada software.constructs (for) dan. Construct software.amazon.awscdk

Konstruksi eksperimental disediakan dalam paket terpisah, berversi independen dengan nama yang diakhiri alpha dan nomor versi alfa. Nomor versi alfa sesuai aws-cdk-lib dengan rilis pertama yang kompatibel. Di sini, kami telah menyematkan aws-codestar ke v2.0.0-alpha.1.

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>aws-cdk-lib</artifactId>
    <version>2.0.0</version>
</dependency><dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>code-star-alpha</artifactId>
    <version>2.0.0-alpha.1</version>
</dependency>
<dependency>
    <groupId>software.constructs</groupId>
    <artifactId>constructs</artifactId>
    <version>10.0.0</version>
</dependency>
```

Instal dependensi baru dengan menjalankan. mvn package

Ubah kode Anda untuk melakukan hal berikut:

- Impor Construct dari software.constructs perpustakaan baru
- Impor kelas inti, seperti Stack danApp, dari software.amazon.awscdk
- Impor konstruksi layanan dari software.amazon.awscdk.services

```
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
```

```
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.App;
import software.amazon.awscdk.services.s3.Bucket;
import software.amazon.awscdk.services.codestar.alpha.GitHubRepository;
```

C#

Cara paling mudah untuk meningkatkan dependensi aplikasi C# CDK adalah dengan mengedit file secara manual. `.csproj` Hapus semua referensi `Amazon.CDK.*` paket stabil dan ganti dengan referensi ke `Constructs` paket `Amazon.CDK.Lib` dan.

Konstruksi eksperimental disediakan dalam paket terpisah, berversi independen dengan nama yang diakhiri `alpha` dan nomor versi alfa. Nomor versi alfa sesuai `aws-cdk-lib` dengan rilis pertama yang kompatibel. Di sini, kami telah menyematkan `aws-codestar` ke v2.0.0-alpha.1.

```
<PackageReference Include="Amazon.CDK.Lib" Version="2.0.0" />
<PackageReference Include="Amazon.CDK.AWS.Codestar.Alpha" Version="2.0.0-alpha.1" />
<PackageReference Include="Constructs" Version="10.0.0" />
```

Instal dependensi baru dengan menjalankan. `dotnet restore`

Ubah impor di file sumber Anda sebagai berikut.

```
using Constructs;                   // for Construct class
using Amazon.CDK;                   // for core classes like App and Stack
using Amazon.CDK.AWS.S3;            // for stable constructs like Bucket
using Amazon.CDK.Codestar.Alpha;    // for experimental constructs
```

Go

Masalah go get untuk memperbarui dependensi Anda ke versi terbaru dan memperbarui file proyek Anda. `.mod`

Menguji aplikasi yang dimigrasi sebelum menerapkan

Sebelum menerapkan tumpukan Anda, gunakan `cdk diff` untuk memeriksa perubahan tak terduga pada sumber daya. Perubahan pada ID logis (menyebabkan penggantian sumber daya) tidak diharapkan.

Perubahan yang diharapkan termasuk tetapi tidak terbatas pada:

- Perubahan `CDKMetadata` sumber daya.
- Hash aset yang diperbarui.
- Perubahan yang terkait dengan sintesis tumpukan gaya baru. Berlaku jika aplikasi Anda menggunakan synthesizer stack lama di v1. (CDK v2 tidak mendukung synthesizer tumpukan lama.)
- Penambahan `CheckBootstrapVersion` aturan.

Perubahan tak terduga biasanya tidak disebabkan oleh peningkatan ke AWS CDK v2 itu sendiri. Biasanya, ini adalah hasil dari perilaku usang yang sebelumnya diubah oleh flag fitur. Ini adalah gejala peningkatan dari versi CDK lebih awal dari sekitar 1.85.x. Anda akan melihat perubahan yang sama ditingkatkan ke rilis v1.x terbaru. Anda biasanya dapat menyelesaikan ini dengan melakukan hal berikut:

1. Tingkatkan aplikasi Anda ke rilis v1.x terbaru
2. Hapus bendera fitur
3. Merevisi kode Anda seperlunya
4. Deploy
5. Tingkatkan ke v2

> **Note**
>
> Jika aplikasi yang ditingkatkan tidak dapat diterapkan setelah pemutakhiran dua tahap, laporkan masalah tersebut.

Saat Anda siap untuk menerapkan tumpukan di aplikasi Anda, pertimbangkan untuk menerapkan salinan terlebih dahulu sehingga Anda dapat mengujinya. Cara termudah untuk melakukannya adalah dengan menyebarkannya ke Wilayah yang berbeda. Namun, Anda juga dapat mengubah ID tumpukan Anda. Setelah pengujian, pastikan untuk menghancurkan salinan pengujian dengancdk destroy.

Pemecahan Masalah

TypeScript **`'from' expected`**atau **`';' expected`** kesalahan dalam impor

Upgrade ke TypeScript 3.8 atau lebih baru.

Jalankan 'cdk bootstrap'

Jika Anda melihat kesalahan seperti berikut:

```
#  MyStack failed: Error: MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not
 found. Has the environment been bootstrapped? Please run 'cdk bootstrap' (see https://
docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html)
    at CloudFormationDeployments.validateBootstrapStackVersion (.../aws-cdk/lib/api/
cloudformation-deployments.ts:323:13)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment
 been bootstrapped? Please run 'cdk bootstrap' (see https://docs.aws.amazon.com/cdk/
latest/guide/bootstrapping.html)
```

AWS CDK v2 membutuhkan tumpukan bootstrap yang diperbarui, dan selanjutnya, semua penerapan v2 memerlukan sumber daya bootstrap. (Dengan v1, Anda dapat menerapkan tumpukan sederhana tanpa bootstrap.) Untuk detail lengkap, lihat the section called “Bootstrapping”.

Menemukan tumpukan v1

Saat memigrasikan aplikasi CDK Anda dari v1 ke v2, Anda mungkin ingin mengidentifikasi AWS CloudFormation tumpukan yang diterapkan yang dibuat menggunakan v1. Untuk melakukan ini, jalankan perintah berikut:

```
npx awscdk-v1-stack-finder
```

Untuk detail penggunaan, lihat awscdk-v1-stack-finder README.

Migrasikan sumber daya dan AWS CloudFormation templat yang ada ke AWS CDK

Fitur CDK Migrate ada dalam rilis pratinjau untuk AWS CDK dan dapat berubah sewaktu-waktu.

Gunakan AWS Cloud Development Kit (AWS CDK) Command Line Interface (AWS CDK CLI) untuk memigrasikan AWS sumber daya yang diterapkan, AWS CloudFormation tumpukan yang diterapkan, dan templat lokal ke. AWS CloudFormation AWS CDK

Topik

Cara kerja migrasi

Gunakan AWS CDK CLI `cdk migrate` perintah untuk bermigrasi dari sumber berikut:

- AWS Sumber daya yang digunakan.
- Tumpukan yang AWS CloudFormation dikerahkan.
- AWS CloudFormation Template lokal.

Sumber daya yang digunakan AWS

Anda dapat memigrasikan AWS sumber daya yang diterapkan dari lingkungan tertentu (Akun AWS dan Wilayah AWS) yang tidak terkait dengan tumpukan. AWS CloudFormation

AWS CDK CLIIni menggunakan layanan generator IAc untuk memindai sumber daya di AWS lingkungan Anda untuk mengumpulkan detail sumber daya. Untuk mempelajari lebih lanjut tentang generator IAC, lihat Membuat templat untuk sumber daya yang ada di Panduan AWS CloudFormation Pengguna.

Setelah mengumpulkan detail sumber daya, Anda akan AWS CDK CLI membuat aplikasi CDK baru yang menyertakan satu tumpukan yang berisi sumber daya yang dimigrasi.

Tumpukan yang dikerahkan AWS CloudFormation

Anda dapat memigrasikan satu AWS CloudFormation tumpukan ke AWS CDK aplikasi baru. Ini AWS CDK CLI akan mengambil AWS CloudFormation template tumpukan Anda dan membuat aplikasi CDK baru. Aplikasi CDK akan terdiri dari satu tumpukan yang berisi tumpukan migrasi AWS CloudFormation Anda.

AWS CloudFormation Template lokal

Anda dapat bermigrasi dari AWS CloudFormation template lokal. Template lokal mungkin atau mungkin tidak berisi sumber daya yang digunakan. Ini AWS CDK CLI akan membuat aplikasi CDK baru yang berisi satu tumpukan dengan sumber daya Anda.

Setelah bermigrasi, Anda dapat mengelola, memodifikasi, dan menerapkan aplikasi CDK AWS CloudFormation untuk menyediakan atau memperbarui sumber daya Anda.

Manfaat CDK Migrate

Migrasi sumber daya ke dalam secara AWS CDK historis merupakan proses manual yang membutuhkan waktu dan keahlian dengan AWS CloudFormation dan bahkan AWS CDK untuk memulai. Dengan CDK Migrate, AWS CDK CLI kami memfasilitasi sebagian besar upaya migrasi untuk Anda dalam waktu yang singkat. CDK Migrate akan segera membantu Anda mulai menggunakan aplikasi AWS CDK untuk mengembangkan dan mengelola aplikasi baru dan yang sudah ada. AWS

Pertimbangan

Pertimbangan umum

CDK Migrasi vs Impor CDK

`cdk import`Perintah dapat mengimpor sumber daya yang diterapkan ke aplikasi CDK baru atau yang sudah ada. Saat mengimpor, setiap sumber daya harus didefinisikan secara manual sebagai konstruksi L1 di aplikasi Anda. Sebaiknya gunakan `cdk import` untuk mengimpor satu atau lebih sumber daya sekaligus ke aplikasi CDK baru atau yang sudah ada. Untuk mempelajari selengkapnya, lihat Mengimpor sumber daya yang ada ke dalam tumpukan.

`cdk migrate`Perintah bermigrasi dari sumber daya yang diterapkan, AWS CloudFormation tumpukan yang diterapkan, atau AWS CloudFormation templat lokal ke aplikasi CDK baru. Selama migrasi, AWS CDK CLI penggunaan `cdk import` untuk mengimpor sumber daya Anda ke aplikasi CDK baru. Ini AWS CDK CLI juga menghasilkan konstruksi L1 untuk setiap sumber daya untuk Anda. Sebaiknya gunakan `cdk migrate` saat mengimpor dari sumber migrasi yang didukung ke AWS CDK aplikasi baru.

CDK Migrate hanya membuat konstruksi L1

Aplikasi CDK yang baru dibuat hanya akan menyertakan konstruksi L1. Anda dapat menambahkan konstruksi tingkat yang lebih tinggi ke aplikasi Anda setelah migrasi.

CDK Migrate membuat aplikasi CDK yang berisi satu tumpukan

Aplikasi CDK yang baru dibuat akan berisi satu tumpukan.

Saat memigrasikan sumber daya yang diterapkan, semua sumber daya yang dimigrasi akan terkandung dalam satu tumpukan di aplikasi CDK baru.

Saat memigrasikan AWS CloudFormation tumpukan, Anda hanya dapat memigrasikan satu AWS CloudFormation tumpukan ke dalam satu tumpukan di aplikasi CDK baru.

Migrasi aset

Aset proyek, seperti AWS Lambda kode, tidak akan langsung bermigrasi ke aplikasi CDK baru. Setelah migrasi, Anda dapat menentukan nilai aset untuk memasukkannya ke dalam aplikasi CDK.

Migrasi sumber daya stateful

Saat memigrasikan sumber daya stateful, seperti database dan bucket Amazon Simple Storage Service (Amazon S3), Anda paling sering ingin memigrasikan sumber daya yang ada alih-alih membuat sumber daya baru.

Untuk memigrasi dan melestarikan sumber daya stateful, lakukan hal berikut:

- Verifikasi bahwa sumber daya stateful Anda mendukung impor. Untuk informasi selengkapnya, lihat Dukungan jenis sumber daya di Panduan AWS CloudFormation Pengguna.
- Setelah migrasi, verifikasi bahwa ID logis sumber daya yang dimigrasi di aplikasi CDK baru cocok dengan ID logis sumber daya yang digunakan.
- Jika bermigrasi dari AWS CloudFormation tumpukan, verifikasi bahwa nama tumpukan di aplikasi CDK baru cocok dengan tumpukan. AWS CloudFormation
- Menerapkan aplikasi CDK menggunakan AWS akun yang sama dan sumber daya Wilayah AWS yang dimigrasi.

Pertimbangan saat bermigrasi dari template AWS CloudFormation

CDK Migrate mendukung migrasi template tunggal

Saat memigrasi AWS CloudFormation templat, Anda dapat memilih satu templat untuk dimigrasi. Template bersarang tidak didukung.

Memigrasi template dengan fungsi intrinsik

Saat bermigrasi dari AWS CloudFormation template yang menggunakan fungsi intrinsik, AWS CDK CLI akan mencoba memigrasikan logika Anda ke aplikasi CDK dengan kelas. Fn Untuk mempelajari lebih lanjut, lihat kelas Fn di Referensi AWS Cloud Development Kit (AWS CDK) API.

Pertimbangan saat bermigrasi dari sumber daya yang digunakan

Batasan pemindaian

Saat memindai lingkungan Anda untuk sumber daya, generator IAC memiliki batasan khusus pada data yang dapat diambil dan batasan kuota saat memindai. Untuk mempelajari lebih lanjut, lihat Pertimbangan dalam Panduan AWS CloudFormation Pengguna.

Prasyarat

Sebelum menggunakan `cdk migrate` perintah, lakukan hal berikut:

1. Membangun otentikasi dengan AWS. Untuk petunjuk, lihat Langkah 2: Konfigurasikan akses terprogram.
2. Instal atau tingkatkan AWS CDK CLI. Untuk instruksi instalasi, lihat Langkah 3: Instal AWS CDKCLI.

Memulai dengan CDK Migrate

Untuk memulai, jalankan AWS CDK CLI `cdk migrate` perintah dari direktori pilihan Anda. Berikan opsi wajib dan opsional, tergantung pada jenis migrasi yang Anda lakukan.

Untuk daftar lengkap dan deskripsi opsi yang dapat Anda gunakan`cdk migrate`, lihatcdk migratereferensi perintah.

Berikut ini adalah beberapa opsi penting yang mungkin ingin Anda berikan.

Nama tumpukan

Satu-satunya pilihan yang diperlukan adalah`--stack-name`. Gunakan opsi ini untuk menentukan nama tumpukan yang akan dibuat dalam AWS CDK aplikasi setelah migrasi. Nama tumpukan juga akan digunakan sebagai nama AWS CloudFormation tumpukan Anda saat penerapan.

Bahasa

Gunakan `--language` untuk menentukan bahasa pemrograman aplikasi CDK baru.

AWS akun dan Wilayah AWS

AWS CDK CLIMengambil AWS akun dan Wilayah AWS informasi dari sumber default. Untuk informasi selengkapnya, lihat Langkah 2: Konfigurasikan akses terprogram. Anda dapat menggunakan `--account` dan `--region` opsi `cdk migrate` untuk memberikan nilai lain.

Direktori keluaran proyek CDK baru Anda

Secara default, AWS CDK CLI akan membuat proyek CDK baru di direktori kerja Anda dan menggunakan nilai yang Anda berikan `--stack-name` untuk memberi nama folder proyek. Jika folder dengan nama yang sama saat ini ada, folder AWS CDK CLI akan menimpa folder itu.

Anda dapat menentukan jalur keluaran yang berbeda untuk folder proyek CDK baru dengan `--output-path` opsi.

Sumber migrasi

Berikan opsi untuk menentukan sumber tempat Anda bermigrasi.

- `--from-path`— Migrasi dari AWS CloudFormation template lokal.
- `--from-scan`— Migrasi dari sumber daya yang diterapkan di AWS akun dan. Wilayah AWS
- `--from-stack`— Migrasi dari AWS CloudFormation tumpukan.

Bergantung pada sumber migrasi, Anda dapat memberikan opsi tambahan untuk menyesuaikan `cdk migrate` perintah.

Migrasi dari tumpukan AWS CloudFormation

Untuk bermigrasi dari AWS CloudFormation tumpukan yang diterapkan, berikan opsi. `--from-stack` Berikan nama AWS CloudFormation tumpukan yang Anda gunakan. `--stack-name` Berikut ini adalah contohnya:

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

Yang AWS CDK CLI akan melakukan hal berikut:

1. Ambil AWS CloudFormation template tumpukan yang Anda gunakan.
2. Jalankan `cdk init` untuk menginisialisasi aplikasi CDK baru.
3. Buat tumpukan dalam aplikasi CDK yang berisi tumpukan yang dimigrasi. AWS CloudFormation

Saat Anda bermigrasi dari AWS CloudFormation tumpukan yang diterapkan, AWS CDK CLI upaya untuk mencocokkan ID logis sumber daya yang diterapkan dan nama AWS CloudFormation tumpukan yang diterapkan ke sumber daya yang dimigrasi dan tumpukan di aplikasi CDK baru.

Setelah migrasi, Anda dapat mengelola dan memodifikasi aplikasi CDK secara normal. Saat Anda menerapkan, AWS CloudFormation akan mengidentifikasi penerapan sebagai pembaruan AWS CloudFormation tumpukan karena nama tumpukan yang cocok AWS CloudFormation . Sumber daya dengan ID logis yang cocok akan diperbarui. Untuk informasi selengkapnya tentang penerapan, lihatKelola dan terapkan aplikasi CDK Anda.

Migrasi dari template AWS CloudFormation

CDK Migrate mendukung migrasi dari AWS CloudFormation templat yang diformat dalam atau. `JSON` `YAML`

Untuk bermigrasi dari AWS CloudFormation templat lokal, gunakan `--from-path` opsi dan berikan jalur ke templat lokal. Anda juga harus memberikan `--stack-name` opsi yang diperlukan. Berikut ini adalah contohnya:

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

Yang AWS CDK CLI akan melakukan hal berikut:

1. Ambil AWS CloudFormation template lokal Anda.
2. Jalankan `cdk init` untuk menginisialisasi aplikasi CDK baru.
3. Buat tumpukan dalam aplikasi CDK yang berisi templat yang dimigrasi. AWS CloudFormation

Setelah migrasi, Anda dapat mengelola dan memodifikasi aplikasi CDK secara normal. Saat penerapan, Anda memiliki opsi berikut:

- Perbarui AWS CloudFormation tumpukan — Jika AWS CloudFormation template lokal sebelumnya digunakan, Anda dapat memperbarui tumpukan yang diterapkan AWS CloudFormation .
- Menerapkan AWS CloudFormation tumpukan baru — Jika AWS CloudFormation templat lokal tidak pernah digunakan, atau jika Anda ingin membuat tumpukan baru dari templat yang digunakan sebelumnya, Anda dapat menerapkan tumpukan baru. AWS CloudFormation

Migrasi dari template AWS SAM

Untuk bermigrasi dari templat AWS Serverless Application Model (AWS SAM), Anda harus terlebih dahulu mengonversinya menjadi AWS CloudFormation templat atau menerapkan untuk membuat AWS CloudFormation tumpukan.

Untuk mengonversi AWS SAM template ke AWS CloudFormation, Anda dapat menggunakan AWS SAM CLI `sam validate --debug` perintah. Anda mungkin harus mengatur `lint` ke `false` dalam `samconfig.toml` file Anda sebelum menjalankan perintah ini.

Untuk mengonversi ke AWS CloudFormation tumpukan, gunakan AWS SAM templat menggunakan file. AWS SAM CLI Kemudian bermigrasi dari tumpukan yang diterapkan.

Migrasi dari sumber daya yang digunakan

Untuk bermigrasi dari AWS sumber daya yang diterapkan, berikan opsi. `--from-scan` Anda juga harus memberikan `--stack-name` opsi yang diperlukan. Berikut ini adalah contohnya:

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

Yang AWS CDK CLI akan melakukan hal berikut:

1. Pindai akun Anda untuk informasi sumber daya dan properti — AWS CDK CLI Ini menggunakan generator IAc untuk memindai akun Anda dan mengumpulkan detail.
2. Hasilkan AWS CloudFormation template — Setelah pemindaian, generator AWS CDK CLI menggunakan IAc untuk membuat AWS CloudFormation template.
3. Inisialisasi aplikasi CDK baru dan migrasi template Anda — Ini AWS CDK CLI berjalan `cdk init` untuk menginisialisasi AWS CDK aplikasi baru dan memigrasikan AWS CloudFormation template Anda ke aplikasi CDK sebagai satu tumpukan.

Gunakan filter

Secara default, AWS CDK CLI akan memindai seluruh AWS lingkungan dan memigrasikan sumber daya hingga batas kuota maksimum generator IAC. Anda dapat memberikan filter AWS CDK CLI untuk menentukan kriteria sumber daya yang dimigrasi dari akun Anda ke aplikasi CDK baru. Untuk mempelajari selengkapnya, lihat `--filter`.

Memindai sumber daya dengan generator IAc

Bergantung pada jumlah sumber daya di akun Anda, pemindaian mungkin memakan waktu beberapa menit. Bilah kemajuan akan ditampilkan selama proses pemindaian.

Jenis sumber daya yang mendukung

AWS CDK CLIAkan memigrasikan sumber daya yang didukung oleh generator IAc. Untuk daftar lengkap, lihat Dukungan jenis sumber daya di Panduan AWS CloudFormation Pengguna.

Selesaikan properti hanya tulis

Beberapa sumber daya yang didukung berisi properti write-only. Properti ini dapat ditulis ke, untuk mengkonfigurasi properti, tetapi tidak dapat dibaca oleh generator IAc atau AWS CloudFormation

untuk mendapatkan nilai. Misalnya, properti yang digunakan untuk menentukan kata sandi database mungkin hanya ditulis untuk alasan keamanan.

Saat memindai sumber daya selama migrasi, generator IAc akan mendeteksi sumber daya yang mungkin berisi properti hanya tulis dan akan mengkategorikannya ke dalam salah satu jenis berikut:

- `MUTUALLY_EXCLUSIVE_PROPERTIES`— Ini adalah properti hanya tulis untuk sumber daya tertentu yang dapat dipertukarkan dan melayani tujuan yang sama. Salah satu properti yang saling eksklusif diperlukan untuk mengonfigurasi sumber daya Anda. Misalnya, properti `S3BucketImageUri`, dan untuk `AWS::Lambda::Function` sumber daya adalah `ZipFile` properti khusus tulis yang saling eksklusif. Salah satu dari mereka dapat digunakan untuk menentukan aset fungsi Anda, tetapi Anda harus menggunakannya.
- `MUTUALLY_EXCLUSIVE_TYPES`— Ini adalah properti khusus tulis yang diperlukan yang menerima beberapa jenis konfigurasi. Misalnya, `Body` properti `AWS::ApiGateway::RestApi` sumber daya menerima objek atau tipe string.
- `UNSUPPORTED_PROPERTIES`— Ini adalah properti hanya tulis yang tidak termasuk dalam dua kategori lainnya. Mereka adalah properti opsional atau properti wajib yang menerima array objek.

Untuk informasi selengkapnya tentang properti khusus tulis dan cara IAc generator mengelolanya saat memindai sumber daya yang digunakan dan membuat AWS CloudFormation templat, lihat generator IAc dan properti khusus tulis di Panduan Pengguna.AWS CloudFormation

Setelah migrasi, Anda harus menentukan nilai properti hanya-tulis di aplikasi CDK baru. AWS CDK CLIAkan menambahkan bagian Peringatan ke `ReadMe` file proyek CDK untuk mendokumentasikan properti hanya tulis apa pun yang diidentifikasi oleh generator IAc. Berikut ini adalah contohnya:

```
# Welcome to your CDK TypeScript project
...
## Warnings
### Write-only properties
Write-only properties are resource property values that can be written to but can't be
 read by AWS CloudFormation or CDK Migrate. For more information, see [IaC generator
 and write-only properties](https://docs.aws.amazon.com/AWSCloudFormation/latest/
UserGuide/generate-IaC-write-only-properties.html).

Write-only properties discovered during migration are organized here by resource ID and
 categorized by write-only property type. Resolve write-only properties by providing
 property values in your CDK app. For guidance, see [Resolve write-only properties]
(https://docs.aws.amazon.com/cdk/v2/guide/migrate.html#migrate-resources-writeonly).
```

```
### MyLambdaFunction
- **UNSUPPORTED_PROPERTIES**:
  - SnapStart/ApplyOn: Applying SnapStart setting on function resource type.Possible
 values: [PublishedVersions, None]
This property can be replaced with other types
  - Code/S3ObjectVersion: For versioned objects, the version of the deployment package
 object to use.
This property can be replaced with other exclusive properties
- **MUTUALLY_EXCLUSIVE_PROPERTIES**:
  - Code/S3Bucket: An Amazon S3 bucket in the same AWS Region as your function. The
 bucket can be in a different AWS account.
This property can be replaced with other exclusive properties
  - Code/S3Key: The Amazon S3 key of the deployment package.
This property can be replaced with other exclusive properties
```

- Peringatan diatur di bawah judul yang mengidentifikasi ID logis sumber daya yang terkait dengannya.
- Peringatan dikategorikan berdasarkan jenis. Jenis ini datang langsung dari generator IAc.

Untuk menyelesaikan properti hanya tulis

1. Identifikasi properti hanya-tulis untuk diselesaikan dari bagian Peringatan pada file proyek CDK Anda. `ReadMe` Di sini, Anda dapat mencatat sumber daya di aplikasi CDK Anda yang mungkin berisi properti khusus tulis dan mengidentifikasi jenis properti hanya-tulis yang ditemukan.

   a. Untuk`MUTUALLY_EXCLUSIVE_PROPERTIES`, tentukan properti yang saling eksklusif untuk dikonfigurasi di AWS CDK aplikasi Anda.

   b. Untuk`MUTUALLY_EXCLUSIVE_TYPES`, tentukan jenis yang diterima yang akan Anda gunakan untuk mengkonfigurasi properti.

   c. Untuk`UNSUPPORTED_PROPERTIES`, tentukan apakah properti itu opsional atau wajib. Kemudian, konfigurasikan seperlunya.

2. Gunakan panduan dari generator IAc dan properti tulis saja untuk mereferensikan arti jenis peringatan.
3. Di aplikasi CDK Anda, nilai properti hanya-tulis yang akan diselesaikan juga akan ditentukan di `Props` bagian aplikasi Anda. Berikan nilai yang benar di sini. Untuk deskripsi dan panduan properti, Anda dapat mereferensikan Referensi AWS CDK API.

Berikut ini adalah contoh `Props` bagian dalam aplikasi CDK yang dimigrasi dengan dua properti khusus tulis yang harus diselesaikan:

```
export interface MyTestAppStackProps extends cdk.StackProps {
  /**
   * The Amazon S3 key of the deployment package.
   */
  readonly lambdaFunction00asdfasdfsadf008grk1CodeS3Keym8P82: string;
  /**
   * An Amazon S3 bucket in the same AWS Region as your function. The bucket can be
 in a different AWS account.
   */
  readonly lambdaFunction00asdfasdfsadf008grk1CodeS3Bucketzidw8: string;
}
```

Setelah Anda menyelesaikan semua nilai properti write-only, Anda siap untuk mempersiapkan penerapan.

Berkas.json yang bermigrasi

AWS CDK CLIMembuat `migrate.json` file dalam AWS CDK proyek Anda selama migrasi. File ini berisi informasi referensi tentang sumber daya yang Anda gunakan. Saat Anda menerapkan aplikasi CDK untuk pertama kalinya, file ini akan AWS CDK CLI digunakan untuk mereferensikan sumber daya yang Anda gunakan, mengaitkan sumber daya Anda dengan AWS CloudFormation tumpukan baru, dan menghapus file.

Kelola dan terapkan aplikasi CDK Anda

Saat bermigrasi ke AWS CDK, aplikasi CDK baru mungkin tidak segera siap penerapan. Topik ini menjelaskan item tindakan yang perlu dipertimbangkan saat mengelola dan menerapkan aplikasi CDK baru Anda.

Mempersiapkan penyebaran

Sebelum menerapkan, Anda harus menyiapkan aplikasi CDK Anda.

Sintesis aplikasi Anda

Gunakan `cdk synth` perintah untuk mensintesis tumpukan di aplikasi CDK Anda ke dalam template. AWS CloudFormation

Jika Anda bermigrasi dari AWS CloudFormation tumpukan atau templat yang diterapkan, Anda dapat membandingkan templat yang disintesis dengan templat yang dimigrasi untuk memverifikasi nilai sumber daya dan properti.

Untuk mempelajari lebih lanjut tentang `cdk synth`, lihat Mensintesis tumpukan.

Lakukan diff

Jika Anda bermigrasi dari AWS CloudFormation tumpukan yang diterapkan, Anda dapat menggunakan perintah cdk diff untuk membandingkan dengan tumpukan di aplikasi CDK baru Anda.

Untuk mempelajari lebih lanjut tentang cdk diff, lihat. Membandingkan tumpukan

Bootstrap lingkungan Anda

Jika Anda menerapkan dari AWS lingkungan untuk pertama kalinya, gunakan `cdk bootstrap` untuk mempersiapkan lingkungan Anda. Untuk mempelajari selengkapnya, lihat Bootstrapping.

Menerapkan aplikasi CDK Anda

Saat Anda menerapkan aplikasi CDK, layanan ini AWS CDK CLI menggunakan AWS CloudFormation layanan untuk menyediakan sumber daya Anda. Sumber daya dibundel ke dalam satu tumpukan di aplikasi CDK dan digunakan sebagai tumpukan tunggal. AWS CloudFormation

Bergantung dari mana Anda bermigrasi, Anda dapat menerapkan untuk membuat AWS CloudFormation tumpukan baru atau memperbarui tumpukan yang ada AWS CloudFormation .

Terapkan untuk membuat tumpukan baru AWS CloudFormation

Jika Anda bermigrasi dari sumber daya yang diterapkan, secara otomatis AWS CDK CLI akan membuat AWS CloudFormation tumpukan baru saat penerapan. Sumber daya yang Anda gunakan akan disertakan dalam AWS CloudFormation tumpukan baru.

Jika Anda bermigrasi dari AWS CloudFormation template lokal yang tidak pernah digunakan, AWS CDK CLI maka secara otomatis akan membuat AWS CloudFormation tumpukan baru saat penerapan.

Jika Anda bermigrasi dari AWS CloudFormation tumpukan yang diterapkan atau AWS CloudFormation templat lokal yang sebelumnya digunakan, Anda dapat menerapkan untuk membuat tumpukan baru. AWS CloudFormation Untuk membuat tumpukan baru, lakukan hal berikut:

- Menyebarkan ke AWS lingkungan baru. Ini terdiri dari menggunakan AWS akun yang berbeda atau menyebarkan ke yang berbeda Wilayah AWS.
- Jika Anda ingin menerapkan tumpukan baru ke AWS lingkungan tumpukan atau templat yang dimigrasi yang sama, Anda harus mengubah nama tumpukan di aplikasi CDK ke nilai baru. Anda juga harus memodifikasi semua ID logis sumber daya di aplikasi CDK Anda. Kemudian, Anda dapat menerapkan ke lingkungan yang sama untuk membuat tumpukan baru dan sumber daya baru.

Terapkan untuk memperbarui tumpukan yang ada AWS CloudFormation

Jika Anda bermigrasi dari AWS CloudFormation tumpukan yang diterapkan atau AWS CloudFormation templat lokal yang sebelumnya digunakan, Anda dapat menerapkan untuk memperbarui tumpukan yang ada. AWS CloudFormation

Verifikasi bahwa nama tumpukan di aplikasi CDK Anda cocok dengan nama tumpukan tumpukan yang diterapkan AWS CloudFormation dan terapkan ke lingkungan yang sama. AWS

Bekerja dengan AWS CDK dalam bahasa pemrograman yang didukung

Gunakan AWS Cloud Development Kit (AWS CDK) untuk menentukan AWS Cloud infrastruktur Anda dengan bahasa pemrograman yang didukung.

Topik

Mengimpor Perpustakaan AWS Konstruksi

AWS CDK Termasuk AWS Construct Library, kumpulan konstruksi yang diselenggarakan oleh AWS layanan. Konstruksi stabil perpustakaan ditawarkan dalam satu modul, dipanggil dengan nama TypeScript paketnya:`aws-cdk-lib`. Nama paket sebenarnya bervariasi menurut bahasa.

TypeScript

Menginstal	`instal npm aws-cdk-lib`
Impor	`const cdk = membutuhkan ('aws-cdk -lib');`

JavaScript

Menginstal	`instal npm aws-cdk-lib`

Impor	`const cdk = membutuhkan ('aws-cdk-lib');`

Python

Menginstal	`instalasi python -m pip aws-cdk-lib`
Impor	`impor aws_cdk sebagai cdk`

Java

Tambahkan ke `pom.xml`	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Impor	`impor software.amazon.awscdk.app;` (for example)

C#

Menginstal	`dotnet tambahkan paket Amazon.cdk.lib`
Impor	`menggunakan Amazon.cdk;`

Kelas `construct` dasar dan kode pendukung ada di `constructs` modul. Konstruksi eksperimental, di mana API masih mengalami penyempurnaan, didistribusikan sebagai modul terpisah.

Referensi AWS CDK API

Referensi AWS CDK API menyediakan dokumentasi rinci tentang konstruksi (dan komponen lainnya) di pustaka. Versi Referensi API disediakan untuk setiap bahasa pemrograman yang didukung.

Bahan referensi setiap modul dipecah menjadi bagian-bagian berikut.

- Ikhtisar: Materi pengantar yang perlu Anda ketahui untuk bekerja dengan layanan di AWS CDK, termasuk konsep dan contoh.
- Konstruksi: Kelas perpustakaan yang mewakili satu atau lebih AWS sumber daya konkret. Ini adalah sumber daya atau pola “dikurasi” (L2) (sumber daya L3) yang menyediakan antarmuka tingkat tinggi dengan default yang waras.
- Kelas: Kelas non-konstruksi yang menyediakan fungsionalitas yang digunakan oleh konstruksi dalam modul.
- Structs: Struktur data (bundel atribut) yang mendefinisikan struktur nilai komposit seperti properti (`props`argumen konstruksi) dan opsi.
- Antarmuka: Antarmuka, yang namanya semua dimulai dengan “I”, mendefinisikan fungsionalitas minimum absolut untuk konstruksi yang sesuai atau kelas lainnya. CDK menggunakan antarmuka konstruksi untuk merepresentasikan AWS sumber daya yang ditentukan di luar AWS CDK aplikasi Anda dan direferensikan dengan metode seperti. `Bucket.fromBucketArn()`
- Enum: Koleksi nilai bernama untuk digunakan dalam menentukan parameter konstruksi tertentu. Menggunakan nilai yang disebutkan memungkinkan CDK untuk memeriksa nilai-nilai ini untuk validitas selama sintesis.
- CloudFormation Sumber daya: Konstruksi L1 ini, yang namanya dimulai dengan “Cfn”, mewakili sumber daya yang ditentukan dalam spesifikasi. CloudFormation Mereka secara otomatis dihasilkan dari spesifikasi itu dengan setiap rilis CDK. Setiap konstruksi L2 atau L3 merangkum satu atau lebih sumber daya. CloudFormation
- CloudFormation Jenis Properti: Kumpulan nilai bernama yang menentukan properti untuk setiap CloudFormation Sumber Daya.

Antarmuka dibandingkan dengan kelas konstruksi

AWS CDK Penggunaan antarmuka dengan cara tertentu yang mungkin tidak jelas bahkan jika Anda terbiasa dengan antarmuka sebagai konsep pemrograman.

AWS CDK Dukungan menggunakan sumber daya yang didefinisikan di luar aplikasi CDK menggunakan metode seperti`Bucket.fromBucketArn()`. Sumber daya eksternal tidak dapat dimodifikasi dan mungkin tidak memiliki semua fungsionalitas yang tersedia dengan sumber daya yang ditentukan dalam aplikasi CDK Anda menggunakan misalnya `Bucket` kelas. Antarmuka, kemudian, mewakili fungsionalitas minimum yang tersedia di CDK untuk jenis AWS sumber daya tertentu, termasuk sumber daya eksternal.

Saat membuat instance resource di aplikasi CDK, Anda harus selalu menggunakan class konkret seperti. `Bucket` Saat menentukan jenis argumen yang Anda terima di salah satu konstruksi Anda sendiri, gunakan jenis antarmuka seperti `IBucket` jika Anda siap untuk berurusan dengan sumber daya eksternal (yaitu, Anda tidak perlu mengubahnya). Jika Anda memerlukan konstruksi yang ditentukan CDK, tentukan jenis paling umum yang dapat Anda gunakan.

Beberapa antarmuka adalah versi minimum properti atau bundel opsi yang terkait dengan kelas tertentu, bukan konstruksi. Antarmuka seperti itu dapat berguna saat mensubklasifikasikan untuk menerima argumen yang akan Anda sampaikan ke kelas induk Anda. Jika Anda memerlukan satu atau lebih properti tambahan, Anda akan ingin menerapkan atau menurunkan dari antarmuka ini, atau dari jenis yang lebih spesifik.

> **Note**
>
> Beberapa bahasa pemrograman yang didukung oleh AWS CDK tidak memiliki fitur antarmuka. Dalam bahasa-bahasa ini, antarmuka hanyalah kelas biasa. Anda dapat mengidentifikasi mereka dengan nama mereka, yang mengikuti pola awal “I” diikuti dengan nama beberapa konstruksi lain (misalnya`IBucket`). Aturan yang sama berlaku.

Mengelola dependensi

Dependensi untuk AWS CDK aplikasi atau library Anda dikelola menggunakan alat manajemen paket. Alat-alat ini biasanya digunakan dengan bahasa pemrograman.

Biasanya, AWS CDK mendukung standar bahasa atau alat manajemen paket resmi jika ada. Jika tidak, AWS CDK akan mendukung bahasa yang paling populer atau didukung secara luas. Anda mungkin juga dapat menggunakan alat lain, terutama jika mereka bekerja dengan alat yang didukung. Namun, dukungan resmi untuk alat lain terbatas.

AWS CDK Mendukung manajer paket berikut:

Bahasa	Alat manajemen paket yang didukung
TypeScript/JavaScript	NPM (Node Package Manager) atau Yarn
Python	PIP (Package Installer untuk Python)
Java	Maven

Bahasa	Alat manajemen paket yang didukung
C#	NuGet
Go	Modul Go

Saat Anda membuat proyek baru menggunakan AWS CDK CLI `cdk init` perintah, dependensi untuk pustaka inti CDK dan konstruksi stabil ditentukan secara otomatis.

Untuk informasi selengkapnya tentang mengelola dependensi untuk bahasa pemrograman yang didukung, lihat berikut ini:

- Mengelola dependensi di TypeScript.
- Mengelola dependensi di JavaScript.
- Mengelola dependensi di Python.
- Mengelola dependensi di Java.
- Mengelola dependensi di C#.
- Mengelola dependensi di Go.

AWS CDK Membandingkan TypeScript dengan bahasa lain

TypeScript adalah bahasa pertama yang didukung untuk mengembangkan AWS CDK aplikasi. Oleh karena itu, sejumlah besar contoh kode CDK ditulis dalam TypeScript. Jika Anda mengembangkan dalam bahasa lain, mungkin berguna untuk membandingkan bagaimana AWS CDK kode diimplementasikan TypeScript dibandingkan dengan bahasa pilihan Anda. Ini dapat membantu Anda menggunakan contoh di seluruh dokumentasi.

Mengimpor modul

TypeScript/JavaScript

TypeScript mendukung mengimpor seluruh namespace, atau objek individual dari namespace. Setiap namespace mencakup konstruksi dan kelas lain untuk digunakan dengan layanan yang diberikan. AWS

```
// Import main CDK library as cdk
```

```
import * as cdk from 'aws-cdk-lib';   // ES6 import preferred in TS
const cdk = require('aws-cdk-lib');   // Node.js require() preferred in JS

// Import specific core CDK classes
import { Stack, App } from 'aws-cdk-lib';
const { Stack, App } = require('aws-cdk-lib');

// Import AWS S3 namespace as s3 into current namespace
import { aws_s3 as s3 } from 'aws-cdk-lib';   // TypeScript
const s3 = require('aws-cdk-lib/aws-s3');     // JavaScript

// Having imported cdk already as above, this is also valid
const s3 = cdk.aws_s3;

// Now use s3 to access the S3 types
const bucket = s3.Bucket(...);

// Selective import of s3.Bucket
import { Bucket } from 'aws-cdk-lib/aws-s3';        // TypeScript
const { Bucket } = require('aws-cdk-lib/aws-s3');   // JavaScript

// Now use Bucket to instantiate an S3 bucket
const bucket = Bucket(...);
```

Python

Seperti TypeScript, Python mendukung impor modul namespaced dan impor selektif. Ruang nama dengan Python terlihat seperti aws_cdk. xxx, di mana xxx mewakili nama AWS layanan, seperti s3 untuk Amazon S3. (Amazon S3 digunakan dalam contoh-contoh ini).

```
# Import main CDK library as cdk
import aws_cdk as cdk

# Selective import of specific core classes
from aws_cdk import Stack, App

# Import entire module as s3 into current namespace
import aws_cdk.aws_s3 as s3

# s3 can now be used to access classes it contains
bucket = s3.Bucket(...)
```

```
# Selective import of s3.Bucket into current namespace
from aws_cdk.s3 import Bucket

# Bucket can now be used to instantiate a bucket
bucket = Bucket(...)
```

Java

Impor Java bekerja secara berbeda dari TypeScript's. Setiap pernyataan impor mengimpor nama kelas tunggal dari paket tertentu, atau semua kelas yang didefinisikan dalam paket itu (menggunakan*). Kelas dapat diakses menggunakan nama kelas dengan sendirinya jika telah diimpor, atau nama kelas yang memenuhi syarat termasuk paketnya.

Library dinamai seperti `software.amazon.awscdk.services.xxx` untuk AWS Construct Library (perpustakaan utamanya adalah`software.amazon.awscdk`). ID grup Maven untuk AWS CDK paket adalah. `software.amazon.awscdk`

```
// Make certain core classes available
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.App;

// Make all Amazon S3 construct library classes available
import software.amazon.awscdk.services.s3.*;

// Make only Bucket and EventType classes available
import software.amazon.awscdk.services.s3.Bucket;
import software.amazon.awscdk.services.s3.EventType;

// An imported class may now be accessed using the simple class name (assuming that
 name
// does not conflict with another class)
Bucket bucket = Bucket.Builder.create(...).build();

// We can always use the qualified name of a class (including its package) even
 without an
// import directive
software.amazon.awscdk.services.s3.Bucket bucket =
    software.amazon.awscdk.services.s3.Bucket.Builder.create(...)
        .build();

// Java 10 or later can use var keyword to avoid typing the type twice
var bucket =
    software.amazon.awscdk.services.s3.Bucket.Builder.create(...)
```

```
        .build();
```

C#

Di C #, Anda mengimpor tipe dengan `using` arahan. Ada dua gaya. Satu memberi Anda akses ke semua jenis di namespace yang ditentukan dengan menggunakan nama polos mereka. Dengan yang lain, Anda dapat merujuk ke namespace itu sendiri dengan menggunakan alias.

Paket diberi nama seperti `Amazon.CDK.AWS.xxx` untuk paket AWS Construct Library. (Modul inti adalah`Amazon.CDK`.)

```
// Make CDK base classes available under cdk
using cdk = Amazon.CDK;

// Make all Amazon S3 construct library classes available
using Amazon.CDK.AWS.S3;

// Now we can access any S3 type using its name
var bucket = new Bucket(...);

// Import the S3 namespace under an alias
using s3 = Amazon.CDK.AWS.S3;

// Now we can access an S3 type through the namespace alias
var bucket = new s3.Bucket(...);

// We can always use the qualified name of a type (including its namespace) even
 without a
// using directive
var bucket = new Amazon.CDK.AWS.S3.Bucket(...)
```

Go

Setiap modul AWS Construct Library disediakan sebagai paket Go.

```
import (
    "github.com/aws/aws-cdk-go/awscdk/v2"           // CDK core package
    "github.com/aws/aws-cdk-go/awscdk/v2/awss3"     // AWS S3 construct library
 module
)

// now instantiate a bucket
```

```
bucket := awss3.NewBucket(...)

// use aliases for brevity/clarity
import (
    cdk "github.com/aws/aws-cdk-go/awscdk/v2"          // CDK core package
    s3  "github.com/aws/aws-cdk-go/awscdk/v2/awss3"    // AWS S3 construct library
 module
)

bucket := s3.NewBucket(...)
```

Membuat instantiasi sebuah konstruksi

AWS CDK kelas construct memiliki nama yang sama di semua bahasa yang didukung. Sebagian besar bahasa menggunakan `new` kata kunci untuk membuat instance kelas (Python dan Go tidak). Juga, dalam sebagian besar bahasa, kata kunci `this` mengacu pada contoh saat ini. (Python menggunakan `self` berdasarkan konvensi.) Anda harus meneruskan referensi ke instance saat ini sebagai `scope` parameter untuk setiap konstruksi yang Anda buat.

Argumen ketiga untuk AWS CDK konstruksi adalah`props`, objek yang berisi atribut yang diperlukan untuk membangun konstruksi. Argumen ini mungkin opsional, tetapi ketika diperlukan, bahasa yang didukung menanganinya dengan cara idiomatik. Nama-nama atribut juga disesuaikan dengan pola penamaan standar bahasa.

TypeScript/JavaScript

```
// Instantiate default Bucket
const bucket = new s3.Bucket(this, 'MyBucket');

// Instantiate Bucket with bucketName and versioned properties
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
   versioned: true,
});

// Instantiate Bucket with websiteRedirect, which has its own sub-properties
const bucket = new s3.Bucket(this, 'MyBucket', {
  websiteRedirect: {host: 'aws.amazon.com'}});
```

Python

Python tidak menggunakan `new` kata kunci saat membuat instance kelas. Argumen properti direpresentasikan menggunakan argumen kata kunci, dan argumen diberi nama menggunakan`snake_case`.

Jika nilai props itu sendiri merupakan bundel atribut, itu diwakili oleh kelas dinamai setelah properti, yang menerima argumen kata kunci untuk subproperti.

Dalam Python, instance saat ini diteruskan ke metode sebagai argumen pertama, yang dinamai `self` oleh konvensi.

```
# Instantiate default Bucket
bucket = s3.Bucket(self, "MyBucket")

# Instantiate Bucket with bucket_name and versioned properties
bucket = s3.Bucket(self, "MyBucket", bucket_name="my-bucket", versioned=true)

# Instantiate Bucket with website_redirect, which has its own sub-properties
bucket = s3.Bucket(self, "MyBucket", website_redirect=s3.WebsiteRedirect(
            host_name="aws.amazon.com"))
```

Java

Di Jawa, argumen props diwakili oleh kelas bernama `XxxxProps` (misalnya, `BucketProps` untuk alat peraga `Bucket` konstruksi). Anda membangun argumen props menggunakan pola pembangun.

Setiap `XxxxProps` kelas memiliki pembangun. Ada juga pembangun yang nyaman untuk setiap konstruksi yang membangun alat peraga dan konstruksi dalam satu langkah, seperti yang ditunjukkan pada contoh berikut.

Alat peraga diberi nama sama seperti di TypeScript, menggunakan`camelCase`.

```
// Instantiate default Bucket
Bucket bucket = Bucket(self, "MyBucket");

// Instantiate Bucket with bucketName and versioned properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .bucketName("my-bucket").versioned(true)
                      .build();
```

```
# Instantiate Bucket with websiteRedirect, which has its own sub-properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .websiteRedirect(new websiteRedirect.Builder()
                          .hostName("aws.amazon.com").build())
                      .build();
```

C#

Dalam C #, alat peraga ditentukan menggunakan penginisialisasi objek ke kelas bernama `XxxxProps` (misalnya, `BucketProps` untuk alat peraga `Bucket` konstruksi).

Alat peraga diberi nama mirip dengan TypeScript, kecuali menggunakan`PascalCase`.

Lebih mudah menggunakan `var` kata kunci saat membuat instance konstruksi, jadi Anda tidak perlu mengetikkan nama kelas dua kali. Namun, panduan gaya kode lokal Anda mungkin berbeda.

```
// Instantiate default Bucket
var bucket = Bucket(self, "MyBucket");

// Instantiate Bucket with BucketName and Versioned properties
var bucket =  Bucket(self, "MyBucket", new BucketProps {
                      BucketName = "my-bucket",
                      Versioned  = true});

// Instantiate Bucket with WebsiteRedirect, which has its own sub-properties
var bucket = Bucket(self, "MyBucket", new BucketProps {
                      WebsiteRedirect = new WebsiteRedirect {
                              HostName = "aws.amazon.com"
                      }});
```

Go

Untuk membuat konstruksi di Go, panggil fungsi di `NewXxxxxx` mana `Xxxxxxx` adalah nama konstruksi. Properti konstruksi didefinisikan sebagai struct.

Di Go, semua parameter konstruksi adalah pointer, termasuk nilai seperti angka, Boolean, dan string. Gunakan fungsi kenyamanan seperti `jsii.String` untuk membuat pointer ini.

```
 // Instantiate default Bucket
 bucket := awss3.NewBucket(stack, jsii.String("MyBucket"), nil)
```

```
// Instantiate Bucket with BucketName and Versioned properties
bucket1 := awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
 BucketName: jsii.String("my-bucket"),
 Versioned:  jsii.Bool(true),
})

// Instantiate Bucket with WebsiteRedirect, which has its own sub-properties
bucket2 := awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
 WebsiteRedirect: &awss3.RedirectTarget{
  HostName: jsii.String("aws.amazon.com"),
 }})
```

Mengakses anggota

Adalah umum untuk merujuk pada atribut atau properti konstruksi dan AWS CDK kelas lain dan menggunakan nilai-nilai ini sebagai, misalnya, input untuk membangun konstruksi lain. Perbedaan penamaan yang dijelaskan sebelumnya untuk metode berlaku di sini juga. Selain itu, di Jawa, tidak mungkin untuk mengakses anggota secara langsung. Sebagai gantinya, metode pengambil disediakan.

TypeScript/JavaScript

Nama-nama adalah`camelCase`.

```
bucket.bucketArn
```

Python

Nama-nama adalah`snake_case`.

```
bucket.bucket_arn
```

Java

Metode getter disediakan untuk setiap properti; nama-nama ini adalah`camelCase`.

```
bucket.getBucketArn()
```

C#

Nama-nama adalah`PascalCase`.

```
bucket.BucketArn
```

Go

Nama-nama adalah`PascalCase`.

```
bucket.BucketArn
```

Konstanta enum

Konstanta enum dicakup ke kelas, dan memiliki nama huruf besar dengan garis bawah dalam semua bahasa (kadang-kadang disebut sebagai). `SCREAMING_SNAKE_CASE` Karena nama kelas juga menggunakan casing yang sama di semua bahasa yang didukung kecuali Go, nama enum yang memenuhi syarat juga sama dalam bahasa ini.

```
s3.BucketEncryption.KMS_MANAGED
```

Di Go, konstanta enum adalah atribut dari namespace modul dan ditulis sebagai berikut.

```
awss3.BucketEncryption_KMS_MANAGED
```

Antarmuka objek

AWS CDK Menggunakan antarmuka TypeScript objek untuk menunjukkan bahwa kelas mengimplementasikan serangkaian metode dan properti yang diharapkan. Anda dapat mengenali antarmuka objek karena namanya dimulai dengan`I`. Kelas konkret menunjukkan antarmuka yang diimplementasikan dengan menggunakan kata kunci. `implements`

TypeScript/JavaScript

> **Note**
>
> JavaScript tidak memiliki fitur antarmuka. Anda dapat mengabaikan `implements` kata kunci dan nama kelas yang mengikutinya.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';

class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

Python tidak memiliki fitur antarmuka. Namun, untuk AWS CDK Anda dapat menunjukkan implementasi antarmuka dengan mendekorasi kelas Anda dengan`@jsii.implements(interface)`.

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
  def visit(self, node: IConstruct) -> None:
    print("Visited", node.node.path)
```

Java

```
import software.amazon.awscdk.IAspect;
import software.amazon.awscdk.IConstruct;

public class MyAspect implements IAspect {
    public void visit(IConstruct node) {
        System.out.format("Visited %s", node.getNode().getPath());
    }
}
```

C#

```
using Amazon.CDK;

public class MyAspect : IAspect
{
    public void Visit(IConstruct node)
```

```
    {
        System.Console.WriteLine($"Visited ${node.Node.Path}");
    }
}
```

Go

Go struct tidak perlu secara eksplisit mendeklarasikan antarmuka mana yang mereka terapkan. Kompiler Go menentukan implementasi berdasarkan metode dan properti yang tersedia pada struktur. Misalnya, dalam kode berikut, `MyAspect` mengimplementasikan `IAspect` antarmuka karena menyediakan `Visit` metode yang mengambil konstruksi.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
 fmt.Println("Visited", *node.Node().Path())
}
```

Bekerja dengan AWS CDK di TypeScript

TypeScript adalah bahasa klien yang didukung penuh untuk AWS Cloud Development Kit (AWS CDK) dan dianggap stabil. Bekerja dengan AWS CDK in TypeScript menggunakan alat yang sudah dikenal, termasuk TypeScript compiler Microsoft (`tsc`), Node.js dan Node Package Manager (`npm`). Anda juga dapat menggunakan Yarn jika Anda mau, meskipun contoh dalam Panduan ini menggunakan NPM. Modul yang terdiri dari AWS Construct Library didistribusikan melalui repositori NPM, npmjs.org.

Anda dapat menggunakan editor atau IDE apa pun. Banyak AWS CDK pengembang menggunakan Visual Studio Code (atau VScodium yang setara dengan sumber terbuka), yang memiliki dukungan yang sangat baik untuk. TypeScript

Topik

- Memulai dengan TypeScript
- Membuat proyek
- Menggunakan lokal tsc dan cdk
- Mengelola AWS modul Construct Library

- Mengelola dependensi di TypeScript
- AWS CDK idiom di TypeScript
- Membangun, mensintesis, dan menyebarkan

Memulai dengan TypeScript

Untuk bekerja dengan AWS CDK, Anda harus memiliki AWS akun dan kredensional dan telah menginstal Node.js dan Toolkit. AWS CDK Lihat Memulai dengan AWS CDK.

Anda juga membutuhkan TypeScript dirinya sendiri (versi 3.8 atau yang lebih baru). Jika Anda belum memilikinya, Anda dapat menginstalnya menggunakan`npm`.

```
npm install -g typescript
```

> **Note**
>
> Jika Anda mendapatkan kesalahan izin, dan memiliki akses administrator di sistem Anda, coba`sudo npm install -g typescript`.

Tetap TypeScript up to date dengan regulernya`npm update -g typescript`.

> **Note**
>
> Pengakhiran bahasa pihak ketiga: versi bahasa hanya didukung hingga EOL (End Of Life) dibagikan oleh vendor atau komunitas dan dapat berubah sewaktu-waktu dengan pemberitahuan sebelumnya.

Membuat proyek

Anda membuat AWS CDK proyek baru dengan memanggil `cdk init` dalam direktori kosong. Gunakan `--language` opsi dan tentukan`typescript`:

```
mkdir my-project
cd my-project
cdk init app --language typescript
```

Membuat proyek juga menginstal `aws-cdk-lib` modul dan dependensinya.

`cdk init` menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung dalam nama folder diubah menjadi garis bawah. Namun, nama tersebut harus mengikuti bentuk TypeScript pengenal; misalnya, tidak boleh dimulai dengan angka atau berisi spasi.

Menggunakan lokal `tsc` dan `cdk`

Untuk sebagian besar, panduan ini mengasumsikan Anda menginstal TypeScript dan CDK Toolkit global (`npm install -g typescript aws-cdk`), dan contoh perintah yang disediakan (seperti `cdk synth`) mengikuti asumsi ini. Pendekatan ini memudahkan untuk menjaga kedua komponen tetap up to date, dan karena keduanya mengambil pendekatan ketat untuk kompatibilitas mundur, umumnya ada sedikit risiko untuk selalu menggunakan versi terbaru.

Beberapa tim lebih memilih untuk menentukan semua dependensi dalam setiap proyek, termasuk alat seperti TypeScript kompiler dan CDK Toolkit. Praktik ini memungkinkan Anda menyematkan komponen ini ke versi tertentu dan memastikan bahwa semua pengembang di tim Anda (dan lingkungan CI/CD Anda) menggunakan versi tersebut dengan tepat. Ini menghilangkan kemungkinan sumber perubahan, membantu membuat build dan penerapan lebih konsisten dan berulang.

CDK menyertakan dependensi untuk keduanya TypeScript dan CDK Toolkit di template TypeScript proyek `package.json`, jadi jika Anda ingin menggunakan pendekatan ini, Anda tidak perlu membuat perubahan apa pun pada proyek Anda. Yang perlu Anda lakukan adalah menggunakan perintah yang sedikit berbeda untuk membangun aplikasi Anda dan untuk mengeluarkan `cdk` perintah.

Operasi	Gunakan alat global	Gunakan alat lokal
Inisialisasi proyek	`cdk init --language typescript`	`npx aws-cdk init --language typescript`
Membangun	`tsc`	`npm run build`
Jalankan perintah CDK Toolkit	`cdk ...`	`npm run cdk ...` or `npx aws-cdk ...`

`npx aws-cdk` menjalankan versi CDK Toolkit yang diinstal secara lokal di proyek saat ini, jika ada, kembali ke instalasi global, jika ada. Jika tidak ada instalasi global, `npx` unduh salinan sementara

CDK Toolkit dan jalankan itu. Anda dapat menentukan versi arbitrer dari CDK Toolkit menggunakan @ sintaks: prints. `npx aws-cdk@2.0 --version` `2.0.0`

> **Tip**
>
> Siapkan alias sehingga Anda dapat menggunakan `cdk` perintah dengan instalasi CDK Toolkit lokal.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Mengelola AWS modul Construct Library

Gunakan Node Package Manager (`npm`) untuk menginstal dan memperbarui modul AWS Construct Library untuk digunakan oleh aplikasi Anda, serta paket lain yang Anda butuhkan. (Anda dapat menggunakan `yarn` bukan `npm` jika Anda mau.) `npm`juga menginstal dependensi untuk modul-modul tersebut secara otomatis.

Sebagian besar AWS CDK konstruksi berada dalam paket CDK utama, bernama`aws-cdk-lib`, yang merupakan dependensi default dalam proyek baru yang dibuat oleh. cdk init Modul Perpustakaan AWS Konstruksi “Eksperimental”, di mana konstruksi tingkat yang lebih tinggi masih dalam pengembangan, diberi nama seperti. `@aws-cdk/`*`SERVICE-NAME`*`-alpha` Nama layanan memiliki awalan aws-. Jika Anda tidak yakin dengan nama modul, cari di NPM.

> **Note**
>
> Referensi API CDK juga menunjukkan nama paket.

Misalnya, perintah di bawah ini menginstal modul eksperimental untuk AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

Dukungan Construct Library beberapa layanan ada di lebih dari satu namespace. Misalnya, selain ituaws-route53, ada tiga ruang nama Amazon Route 53 tambahan,, aws-route53-targetsaws-route53-patterns, dan. aws-route53resolver

Dependensi proyek Anda dipertahankan di. package.json Anda dapat mengedit file ini untuk mengunci beberapa atau semua dependensi Anda ke versi tertentu atau untuk memungkinkan mereka diperbarui ke versi yang lebih baru di bawah kriteria tertentu. Untuk memperbarui dependensi NPM project Anda ke versi terbaru yang diizinkan sesuai dengan aturan yang Anda tentukan di: package.json

```
npm update
```

Di TypeScript, Anda mengimpor modul ke kode Anda dengan nama yang sama yang Anda gunakan untuk menginstalnya menggunakan NPM. Kami merekomendasikan praktik berikut saat mengimpor AWS CDK kelas dan modul AWS Construct Library dalam aplikasi Anda. Mengikuti panduan ini akan membantu membuat kode Anda konsisten dengan AWS CDK aplikasi lain serta lebih mudah dipahami.

- Gunakan import arahan gaya ES6, bukan. require()
- Umumnya, impor kelas individu dariaws-cdk-lib.

  ```
  import { App, Stack } from 'aws-cdk-lib';
  ```

- Jika Anda membutuhkan banyak kelasaws-cdk-lib, Anda dapat menggunakan alias namespace cdk alih-alih mengimpor kelas individual. Hindari melakukan keduanya.

  ```
  import * as cdk from 'aws-cdk-lib';
  ```

- Umumnya, impor konstruksi AWS layanan menggunakan alias namespace pendek.

  ```
  import { aws_s3 as s3 } from 'aws-cdk-lib';
  ```

Mengelola dependensi di TypeScript

Dalam proyek TypeScript CDK, dependensi ditentukan dalam package.json file di direktori utama proyek. AWS CDK Modul inti berada dalam satu NPM paket yang disebutaws-cdk-lib.

Saat Anda menginstal paket menggunakannpm install, NPM mencatat paket `package.json` untuk Anda.

Jika mau, Anda dapat menggunakan Benang sebagai pengganti NPM. Namun, CDK tidak mendukung plug-and-play mode Yarn, yang merupakan mode default di Yarn 2. Tambahkan berikut ini ke `.yarnrc.yml` file proyek Anda untuk mematikan fitur ini.

```
nodeLinker: node-modules
```

Aplikasi CDK

Berikut ini adalah contoh `package.json` file yang dihasilkan oleh `cdk init --language typescript` perintah:

```
{
  "name": "my-package",
  "version": "0.1.0",
  "bin": {
    "my-package": "bin/my-package.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^26.0.10",
    "@types/node": "10.17.27",
    "jest": "^26.4.2",
    "ts-jest": "^26.2.0",
    "aws-cdk": "2.16.0",
    "ts-node": "^9.0.0",
    "typescript": "~3.9.7"
  },
  "dependencies": {
    "aws-cdk-lib": "2.16.0",
    "constructs": "^10.0.0",
    "source-map-support": "^0.5.16"
  }
}
```

Untuk aplikasi CDK yang dapat di-deploy, `aws-cdk-lib` harus ditentukan di bagian. `dependencies` `package.json` Anda dapat menggunakan penentu nomor versi tanda sisipan (^) untuk menunjukkan bahwa Anda akan menerima versi yang lebih baru dari yang ditentukan selama mereka berada dalam versi utama yang sama.

Untuk konstruksi eksperimental, tentukan versi yang tepat untuk modul pustaka konstruksi alfa, yang memiliki API yang dapat berubah. Jangan gunakan ^ atau ~ karena versi modul ini yang lebih baru dapat membawa perubahan API yang dapat merusak aplikasi Anda.

Tentukan versi pustaka dan alat yang diperlukan untuk menguji aplikasi Anda (misalnya, kerangka `jest` pengujian) di `devDependencies` bagian. `package.json` Secara opsional, gunakan ^ untuk menentukan bahwa versi yang kompatibel selanjutnya dapat diterima.

Pustaka konstruksi pihak ketiga

Jika Anda mengembangkan pustaka konstruksi, tentukan dependensinya menggunakan kombinasi `devDependencies` bagian `peerDependencies` dan, seperti yang ditunjukkan pada file contoh berikut. `package.json`

```
{
  "name": "my-package",
  "version": "0.0.1",
  "peerDependencies": {
    "aws-cdk-lib": "^2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "^10.0.0"
  },
  "devDependencies": {
    "aws-cdk-lib": "2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "10.0.0",
    "jsii": "^1.50.0",
    "aws-cdk": "^2.14.0"
  }
}
```

Di`peerDependencies`, gunakan tanda sisipan (^) untuk menentukan versi terendah `aws-cdk-lib` yang digunakan perpustakaan Anda. Ini memaksimalkan kompatibilitas perpustakaan Anda dengan berbagai versi CDK. Tentukan versi yang tepat untuk modul pustaka konstruksi alfa, yang memiliki API yang dapat berubah. Menggunakan `peerDependencies` memastikan bahwa hanya ada satu salinan dari semua perpustakaan CDK di pohon. `node_modules`

Di`devDependencies`, tentukan alat dan pustaka yang Anda butuhkan untuk pengujian, secara opsional dengan ^ untuk menunjukkan bahwa versi kompatibel yang lebih baru dapat diterima. Tentukan dengan tepat (tanpa ^ atau ~) versi terendah `aws-cdk-lib` dan paket CDK lainnya yang kompatibel dengan perpustakaan Anda. Praktik ini memastikan bahwa pengujian Anda berjalan terhadap versi tersebut. Dengan cara ini, jika Anda secara tidak sengaja menggunakan fitur yang hanya ditemukan di versi yang lebih baru, pengujian Anda dapat menangkapnya.

> **Warning**
>
> `peerDependencies`diinstal secara otomatis hanya oleh NPM 7 dan yang lebih baru. Jika Anda menggunakan NPM 6 atau sebelumnya, atau jika Anda menggunakan Yarn, Anda harus menyertakan dependensi dependensi Anda di. `devDependencies` Jika tidak, mereka tidak akan diinstal, dan Anda akan menerima peringatan tentang dependensi rekan yang belum terselesaikan.

Menginstal dan memperbarui dependensi

Jalankan perintah berikut untuk menginstal dependensi proyek Anda.

NPM

```
# Install the latest version of everything that matches the ranges in 'package.json'
npm install

# Install the same exact dependency versions as recorded in 'package-lock.json'
npm ci
```

Yarn

```
# Install the latest version of everything that matches the ranges in 'package.json'
yarn upgrade

# Install the same exact dependency versions as recorded in 'yarn.lock'
yarn install --frozen-lockfile
```

Untuk memperbarui modul yang diinstal, yarn upgrade perintah sebelumnya npm install dan dapat digunakan. Perintah mana pun memperbarui paket `node_modules` ke versi terbaru yang memenuhi aturan di`package.json`. Namun, mereka tidak memperbarui `package.json` dirinya sendiri, yang

mungkin ingin Anda lakukan untuk menetapkan versi minimum baru. Jika Anda meng-host paket Anda GitHub, Anda dapat mengonfigurasi pembaruan versi Dependabot untuk memperbarui secara otomatis. `package.json` Alternatif lainnya, gunakan npm-check-updates.

> **Important**
>
> Secara desain, ketika Anda menginstal atau memperbarui dependensi, NPM dan Yarn memilih versi terbaru dari setiap paket yang memenuhi persyaratan yang ditentukan dalam. `package.json` Selalu ada risiko bahwa versi ini dapat rusak (baik secara tidak sengaja atau sengaja). Uji secara menyeluruh setelah memperbarui dependensi proyek Anda.

AWS CDK idiom di TypeScript

Alat Peraga

Semua kelas AWS Construct Library dipakai menggunakan tiga argumen: lingkup di mana konstruksi sedang didefinisikan (induknya di pohon konstruksi), id, dan alat peraga. Argumen props adalah bundel pasangan kunci/nilai yang digunakan konstruksi untuk mengkonfigurasi sumber daya yang dibuatnya. AWS Kelas dan metode lain juga menggunakan pola “bundel atribut” untuk argumen.

Dalam TypeScript, bentuk `props` didefinisikan menggunakan antarmuka yang memberi tahu Anda argumen yang diperlukan dan opsional serta tipenya. Antarmuka seperti didefinisikan untuk setiap jenis `props` argumen, biasanya khusus untuk konstruksi tunggal atau metode. Misalnya, konstruksi Bucket (in the`aws-cdk-lib/aws-s3 module`) menentukan `props` argumen yang sesuai dengan antarmuka. BucketProps

Jika properti itu sendiri merupakan objek, misalnya properti WebsiteRedirect dari`BucketProps`, objek itu akan memiliki antarmuka sendiri yang bentuknya harus sesuai, dalam hal ini. RedirectTarget

Jika Anda mensubklasifikasikan kelas AWS Construct Library (atau mengganti metode yang menggunakan argumen seperti props), Anda dapat mewarisi dari antarmuka yang ada untuk membuat yang baru yang menentukan alat peraga baru yang diperlukan kode Anda. Saat memanggil kelas induk atau metode dasar, umumnya Anda dapat meneruskan seluruh argumen props yang Anda terima, karena atribut apa pun yang disediakan dalam objek tetapi tidak ditentukan dalam antarmuka akan diabaikan.

Rilis future AWS CDK dapat secara kebetulan menambahkan properti baru dengan nama yang Anda gunakan untuk properti Anda sendiri. Melewati nilai yang Anda terima ke rantai pewarisan kemudian

dapat menyebabkan perilaku yang tidak terduga. Lebih aman untuk meneruskan salinan dangkal alat peraga yang Anda terima dengan properti Anda dihapus atau disetel keundefined. Sebagai contoh:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Atau, beri nama properti Anda sehingga jelas bahwa mereka milik konstruksi Anda. Dengan cara ini, kecil kemungkinan mereka akan bertabrakan dengan properti di AWS CDK rilis mendatang. Jika ada banyak dari mereka, gunakan satu objek dengan nama yang tepat untuk menahannya.

Nilai yang hilang

Nilai yang hilang dalam suatu objek (seperti alat peraga) memiliki nilai `undefined` dalam TypeScript. Versi 3.7 dari bahasa memperkenalkan operator yang menyederhanakan bekerja dengan nilai-nilai ini, sehingga lebih mudah untuk menentukan default dan rantai "hubung singkat" ketika nilai yang tidak ditentukan tercapai. Untuk informasi lebih lanjut tentang fitur-fitur ini, lihat Catatan Rilis TypeScript 3.7, khususnya dua fitur pertama, Optional Chaining dan Nullish Coalescing.

Membangun, mensintesis, dan menyebarkan

Umumnya, Anda harus berada di direktori root proyek saat membangun dan menjalankan aplikasi Anda.

Node.js tidak dapat berjalan TypeScript secara langsung; sebagai gantinya, aplikasi Anda dikonversi untuk JavaScript menggunakan TypeScript kompiler,`tsc`. JavaScript Kode yang dihasilkan kemudian dieksekusi.

AWS CDK Secara otomatis melakukan ini kapan pun perlu menjalankan aplikasi Anda. Namun, dapat berguna untuk mengkompilasi secara manual untuk memeriksa kesalahan dan menjalankan tes. Untuk mengompilasi TypeScript aplikasi Anda secara manual, masalah`npm run build`. Anda juga dapat mengeluarkan masalah `npm run watch` untuk masuk ke mode tontonan, di mana TypeScript kompiler secara otomatis membangun kembali aplikasi Anda setiap kali Anda menyimpan perubahan ke file sumber.

Tumpukan yang ditentukan dalam AWS CDK aplikasi Anda dapat disintesis dan diterapkan secara individual atau bersama-sama menggunakan perintah di bawah ini. Umumnya, Anda harus berada di direktori utama proyek Anda ketika Anda menerbitkannya.

- `cdk synth`: Mensintesis AWS CloudFormation template dari satu atau beberapa tumpukan di aplikasi Anda AWS CDK .

- `cdk deploy`: Menerapkan resource yang ditentukan oleh satu atau beberapa tumpukan di AWS CDK aplikasi Anda. AWS

Anda dapat menentukan nama beberapa tumpukan yang akan disintesis atau digunakan dalam satu perintah. Jika aplikasi Anda hanya mendefinisikan satu tumpukan, Anda tidak perlu menentukannya.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Anda juga dapat menggunakan wildcard * (sejumlah karakter) dan? (setiap karakter tunggal) untuk mengidentifikasi tumpukan berdasarkan pola. Saat menggunakan wildcard, lampirkan pola dalam tanda kutip. Jika tidak, shell mungkin mencoba memperluasnya ke nama-nama file di direktori saat ini sebelum diteruskan ke AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Anda tidak perlu mensintesis tumpukan secara eksplisit sebelum menerapkannya; `cdk deploy` melakukan langkah ini agar Anda memastikan kode terbaru Anda diterapkan.

Untuk dokumentasi lengkap dari `cdk` perintah, lihatthe section called “AWS CDK Toolkit”.

Bekerja dengan AWS CDK di JavaScript

JavaScriptadalah bahasa klien yang didukung penuh untuk AWS CDK dan dianggap stabil. Bekerja dengan AWS Cloud Development Kit (AWS CDK) in JavaScript menggunakan alat yang sudah dikenal, termasuk Node.js dan Node Package Manager (`npm`). Anda juga dapat menggunakan Yarn jika Anda mau, meskipun contoh dalam Panduan ini menggunakan NPM. Modul yang terdiri dari AWS Construct Library didistribusikan melalui repositori NPM, npmjs.org.

Anda dapat menggunakan editor atau IDE apa pun. Banyak AWS CDK pengembang menggunakan Visual Studio Code (atau VScodium yang setara dengan sumber terbuka), yang memiliki dukungan yang baik untuk. JavaScript

Topik

Memulai dengan JavaScript

Untuk bekerja dengan AWS CDK, Anda harus memiliki AWS akun dan kredensional dan telah menginstal Node.js dan Toolkit. AWS CDK Lihat Memulai dengan AWS CDK.

JavaScript AWS CDK aplikasi tidak memerlukan prasyarat tambahan di luar ini.

> **Note**
>
> Penghentian bahasa pihak ketiga: versi bahasa hanya didukung hingga EOL (End Of Life) dibagikan oleh vendor atau komunitas dan dapat berubah sewaktu-waktu dengan pemberitahuan sebelumnya.

Membuat proyek

Anda membuat AWS CDK proyek baru dengan memanggil `cdk init` dalam direktori kosong. Gunakan `--language` opsi dan tentukan`javascript`:

```
mkdir my-project
cd my-project
cdk init app --language javascript
```

Membuat proyek juga menginstal `aws-cdk-lib`modul dan dependensinya.

`cdk init`menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung dalam nama folder diubah menjadi garis bawah.

Namun, nama tersebut harus mengikuti bentuk JavaScript pengenal; misalnya, tidak boleh dimulai dengan angka atau berisi spasi.

Menggunakan lokal **cdk**

Untuk sebagian besar, panduan ini mengasumsikan Anda menginstal CDK Toolkit global (`npm install -g aws-cdk`), dan contoh perintah yang disediakan (seperti`cdk synth`) mengikuti asumsi ini. Pendekatan ini memudahkan untuk menjaga CDK Toolkit tetap up to date, dan karena CDK mengambil pendekatan ketat untuk kompatibilitas mundur, umumnya ada sedikit risiko untuk selalu menggunakan versi terbaru.

Beberapa tim lebih memilih untuk menentukan semua dependensi dalam setiap proyek, termasuk alat seperti CDK Toolkit. Praktik ini memungkinkan Anda menyematkan komponen tersebut ke versi tertentu dan memastikan bahwa semua pengembang di tim Anda (dan lingkungan CI/CD Anda) menggunakan versi tersebut dengan tepat. Ini menghilangkan kemungkinan sumber perubahan, membantu membuat build dan penerapan lebih konsisten dan berulang.

CDK menyertakan ketergantungan untuk CDK Toolkit dalam template JavaScript proyek`package.json`, jadi jika Anda ingin menggunakan pendekatan ini, Anda tidak perlu membuat perubahan apa pun pada proyek Anda. Yang perlu Anda lakukan adalah menggunakan perintah yang sedikit berbeda untuk membangun aplikasi Anda dan untuk mengeluarkan `cdk` perintah.

Operasi	Gunakan Toolkit CDK global	Gunakan Toolkit CDK lokal
Inisialisasi proyek	`cdk init --language javascript`	`npx aws-cdk init --language javascript`
Jalankan perintah CDK Toolkit	`cdk...`	`npm jalankan cdk...` or `npx aws-cdk...`

`npx aws-cdk`menjalankan versi CDK Toolkit yang diinstal secara lokal di proyek saat ini, jika ada, kembali ke instalasi global, jika ada. Jika tidak ada instalasi global, `npx` unduh salinan sementara CDK Toolkit dan jalankan itu. Anda dapat menentukan versi arbitrer dari CDK Toolkit menggunakan `@` sintaks: prints. `npx aws-cdk@1.120 --version` `1.120.0`

> **Tip**
>
> Siapkan alias sehingga Anda dapat menggunakan `cdk` perintah dengan instalasi CDK Toolkit lokal.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Mengelola AWS modul Construct Library

Gunakan Node Package Manager (`npm`) untuk menginstal dan memperbarui modul AWS Construct Library untuk digunakan oleh aplikasi Anda, serta paket lain yang Anda butuhkan. (Anda dapat menggunakan `yarn` bukan `npm` jika Anda mau.) `npm`juga menginstal dependensi untuk modul-modul tersebut secara otomatis.

Sebagian besar AWS CDK konstruksi berada dalam paket CDK utama, bernama`aws-cdk-lib`, yang merupakan dependensi default dalam proyek baru yang dibuat oleh. `cdk init` Modul Perpustakaan AWS Konstruksi “Eksperimental”, di mana konstruksi tingkat yang lebih tinggi masih dalam pengembangan, diberi nama seperti. `aws-cdk-lib/`*`SERVICE-NAME`*`-alpha` Nama layanan memiliki awalan aws-. Jika Anda tidak yakin dengan nama modul, cari di NPM.

> **Note**
>
> Referensi API CDK juga menunjukkan nama paket.

Misalnya, perintah di bawah ini menginstal modul eksperimental untuk AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

Dukungan Construct Library beberapa layanan ada di lebih dari satu namespace. Misalnya, selain ituaws-route53, ada tiga ruang nama Amazon Route 53 tambahan,, aws-route53-targetsaws-route53-patterns, dan. aws-route53resolver

Dependensi proyek Anda dipertahankan di. package.json Anda dapat mengedit file ini untuk mengunci beberapa atau semua dependensi Anda ke versi tertentu atau untuk memungkinkan mereka diperbarui ke versi yang lebih baru di bawah kriteria tertentu. Untuk memperbarui dependensi NPM project Anda ke versi terbaru yang diizinkan sesuai dengan aturan yang Anda tentukan di: package.json

```
npm update
```

Di JavaScript, Anda mengimpor modul ke kode Anda dengan nama yang sama yang Anda gunakan untuk menginstalnya menggunakan NPM. Kami merekomendasikan praktik berikut saat mengimpor AWS CDK kelas dan modul AWS Construct Library dalam aplikasi Anda. Mengikuti panduan ini akan membantu membuat kode Anda konsisten dengan AWS CDK aplikasi lain serta lebih mudah dipahami.

- Gunakanrequire(), bukan arahan gaya ES6import. Versi Node.js yang lebih lama tidak mendukung impor ES6, jadi menggunakan sintaks yang lebih lama lebih kompatibel secara luas. (Jika Anda benar-benar ingin menggunakan impor ES6, gunakan esm untuk memastikan proyek Anda kompatibel dengan semua versi Node.js yang didukung.)
- Umumnya, impor kelas individu dariaws-cdk-lib.

  ```
  const { App, Stack } = require('aws-cdk-lib');
  ```

- Jika Anda membutuhkan banyak kelasaws-cdk-lib, Anda dapat menggunakan alias namespace cdk alih-alih mengimpor kelas individual. Hindari melakukan keduanya.

  ```
  const cdk = require('aws-cdk-lib');
  ```

- Umumnya, impor AWS Construct Libraries menggunakan alias namespace pendek.

  ```
  const { s3 } = require('aws-cdk-lib/aws-s3');
  ```

Mengelola dependensi di JavaScript

Dalam proyek JavaScript CDK, dependensi ditentukan dalam `package.json` file di direktori utama proyek. AWS CDK Modul inti berada dalam satu NPM paket yang disebut`aws-cdk-lib`.

Saat Anda menginstal paket menggunakannpm install, NPM mencatat paket `package.json` untuk Anda.

Jika mau, Anda dapat menggunakan Benang sebagai pengganti NPM. Namun, CDK tidak mendukung plug-and-play mode Yarn, yang merupakan mode default di Yarn 2. Tambahkan berikut ini ke `.yarnrc.yml` file proyek Anda untuk mematikan fitur ini.

```
nodeLinker: node-modules
```

Aplikasi CDK

Berikut ini adalah contoh `package.json` file yang dihasilkan oleh `cdk init --language typescript` perintah. File yang JavaScript dihasilkan untuk serupa, hanya tanpa entri TypeScript terkait.

```
{
  "name": "my-package",
  "version": "0.1.0",
  "bin": {
    "my-package": "bin/my-package.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^26.0.10",
    "@types/node": "10.17.27",
    "jest": "^26.4.2",
    "ts-jest": "^26.2.0",
    "aws-cdk": "2.16.0",
    "ts-node": "^9.0.0",
    "typescript": "~3.9.7"
  },
```

```
  "dependencies": {
    "aws-cdk-lib": "2.16.0",
    "constructs": "^10.0.0",
    "source-map-support": "^0.5.16"
  }
}
```

Untuk aplikasi CDK yang dapat di-deploy, `aws-cdk-lib` harus ditentukan di bagian. `dependencies` `package.json` Anda dapat menggunakan penentu nomor versi tanda sisipan (^) untuk menunjukkan bahwa Anda akan menerima versi yang lebih baru dari yang ditentukan selama mereka berada dalam versi utama yang sama.

Untuk konstruksi eksperimental, tentukan versi yang tepat untuk modul pustaka konstruksi alfa, yang memiliki API yang dapat berubah. Jangan gunakan ^ atau ~ karena versi modul ini yang lebih baru dapat membawa perubahan API yang dapat merusak aplikasi Anda.

Tentukan versi pustaka dan alat yang diperlukan untuk menguji aplikasi Anda (misalnya, kerangka `jest` pengujian) di `devDependencies` bagian. `package.json` Secara opsional, gunakan ^ untuk menentukan bahwa versi yang kompatibel selanjutnya dapat diterima.

Pustaka konstruksi pihak ketiga

Jika Anda mengembangkan pustaka konstruksi, tentukan dependensinya menggunakan kombinasi `devDependencies` bagian `peerDependencies` dan, seperti yang ditunjukkan pada file contoh berikut. `package.json`

```
{
  "name": "my-package",
  "version": "0.0.1",
  "peerDependencies": {
    "aws-cdk-lib": "^2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "^10.0.0"
  },
  "devDependencies": {
    "aws-cdk-lib": "2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "10.0.0",
    "jsii": "^1.50.0",
    "aws-cdk": "^2.14.0"
  }
```

```
}
```

Di`peerDependencies`, gunakan tanda sisipan (^) untuk menentukan versi terendah `aws-cdk-lib` yang digunakan perpustakaan Anda. Ini memaksimalkan kompatibilitas perpustakaan Anda dengan berbagai versi CDK. Tentukan versi yang tepat untuk modul pustaka konstruksi alfa, yang memiliki API yang dapat berubah. Menggunakan `peerDependencies` memastikan bahwa hanya ada satu salinan dari semua perpustakaan CDK di pohon. `node_modules`

Di`devDependencies`, tentukan alat dan pustaka yang Anda butuhkan untuk pengujian, secara opsional dengan ^ untuk menunjukkan bahwa versi kompatibel yang lebih baru dapat diterima. Tentukan dengan tepat (tanpa ^ atau ~) versi terendah `aws-cdk-lib` dan paket CDK lainnya yang kompatibel dengan perpustakaan Anda. Praktik ini memastikan bahwa pengujian Anda berjalan terhadap versi tersebut. Dengan cara ini, jika Anda secara tidak sengaja menggunakan fitur yang hanya ditemukan di versi yang lebih baru, pengujian Anda dapat menangkapnya.

> **Warning**
>
> `peerDependencies`diinstal secara otomatis hanya oleh NPM 7 dan yang lebih baru. Jika Anda menggunakan NPM 6 atau sebelumnya, atau jika Anda menggunakan Yarn, Anda harus menyertakan dependensi dependensi Anda di. `devDependencies` Jika tidak, mereka tidak akan diinstal, dan Anda akan menerima peringatan tentang dependensi rekan yang belum terselesaikan.

Menginstal dan memperbarui dependensi

Jalankan perintah berikut untuk menginstal dependensi proyek Anda.

NPM

```
# Install the latest version of everything that matches the ranges in 'package.json'
npm install

# Install the same exact dependency versions as recorded in 'package-lock.json'
npm ci
```

Yarn

```
# Install the latest version of everything that matches the ranges in 'package.json'
yarn upgrade
```

```
# Install the same exact dependency versions as recorded in 'yarn.lock'
yarn install --frozen-lockfile
```

Untuk memperbarui modul yang diinstal, yarn upgrade perintah sebelumnya npm install dan dapat digunakan. Perintah mana pun memperbarui paket `node_modules` ke versi terbaru yang memenuhi aturan di`package.json`. Namun, mereka tidak memperbarui `package.json` dirinya sendiri, yang mungkin ingin Anda lakukan untuk menetapkan versi minimum baru. Jika Anda meng-host paket Anda GitHub, Anda dapat mengonfigurasi pembaruan versi Dependabot untuk memperbarui secara otomatis. `package.json` Alternatif lainnya, gunakan npm-check-updates.

> **Important**
>
> Secara desain, ketika Anda menginstal atau memperbarui dependensi, NPM dan Yarn memilih versi terbaru dari setiap paket yang memenuhi persyaratan yang ditentukan dalam. `package.json` Selalu ada risiko bahwa versi ini dapat rusak (baik secara tidak sengaja atau sengaja). Uji secara menyeluruh setelah memperbarui dependensi proyek Anda.

AWS CDK idiom di JavaScript

Alat Peraga

Semua kelas AWS Construct Library dipakai menggunakan tiga argumen: ruang lingkup di mana konstruksi sedang didefinisikan (induknya di pohon konstruksi), id, dan props, bundel pasangan kunci/nilai yang digunakan konstruksi untuk mengkonfigurasi sumber daya yang dibuatnya. AWS Kelas dan metode lain juga menggunakan pola “bundel atribut” untuk argumen.

Menggunakan IDE atau editor yang memiliki JavaScript pelengkapan otomatis yang baik akan membantu menghindari salah eja nama properti. Jika sebuah konstruksi mengharapkan `encryptionKeys` properti, dan Anda mengejanya, saat membuat instance konstruksi`encryptionkeys`, Anda belum melewati nilai yang Anda inginkan. Ini dapat menyebabkan kesalahan pada waktu sintesis jika properti diperlukan, atau menyebabkan properti diabaikan secara diam-diam jika bersifat opsional. Dalam kasus terakhir, Anda mungkin mendapatkan perilaku default yang ingin Anda timpa. Berhati-hatilah di sini.

Saat mensubklasifikasikan class AWS Construct Library (atau mengganti metode yang menggunakan argumen seperti props), Anda mungkin ingin menerima properti tambahan untuk Anda gunakan

sendiri. Nilai-nilai ini akan diabaikan oleh kelas induk atau metode overridden, karena mereka tidak pernah diakses dalam kode itu, sehingga Anda umumnya dapat meneruskan semua props yang Anda terima.

Rilis future AWS CDK dapat secara kebetulan menambahkan properti baru dengan nama yang Anda gunakan untuk properti Anda sendiri. Melewati nilai yang Anda terima ke rantai pewarisan kemudian dapat menyebabkan perilaku yang tidak terduga. Lebih aman untuk meneruskan salinan dangkal alat peraga yang Anda terima dengan properti Anda dihapus atau disetel keundefined. Sebagai contoh:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Atau, beri nama properti Anda sehingga jelas bahwa mereka milik konstruksi Anda. Dengan cara ini, kecil kemungkinan mereka akan bertabrakan dengan properti di AWS CDK rilis mendatang. Jika ada banyak dari mereka, gunakan satu objek dengan nama yang tepat untuk menahannya.

Nilai yang hilang

Nilai yang hilang dalam suatu objek (sepertiprops) memiliki nilai undefined dalam JavaScript. Teknik yang biasa berlaku untuk menangani ini. Misalnya, idiom umum untuk mengakses properti dari nilai yang mungkin tidak terdefinisi adalah sebagai berikut:

```
// a may be undefined, but if it is not, it may have an attribute b
// c is undefined if a is undefined, OR if a doesn't have an attribute b
let c = a && a.b;
```

Namun, jika a bisa memiliki nilai “palsu” lain selain ituundefined, lebih baik membuat tes lebih eksplisit. Di sini, kita akan memanfaatkan fakta bahwa null dan undefined sama dengan menguji keduanya sekaligus:

```
let c = a == null ? a : a.b;
```

> **Tip**
>
> Node.js 14.0 dan yang lebih baru mendukung operator baru yang dapat menyederhanakan penanganan nilai yang tidak ditentukan. Untuk informasi lebih lanjut, lihat proposal chaining opsional dan nullish coalescing.

Mensintesis dan menyebarkan

Tumpukan yang ditentukan dalam AWS CDK aplikasi Anda dapat disintesis dan diterapkan secara individual atau bersama-sama menggunakan perintah di bawah ini. Umumnya, Anda harus berada di direktori utama proyek Anda ketika Anda menerbitkannya.

- `cdk synth`: Mensintesis AWS CloudFormation template dari satu atau beberapa tumpukan di aplikasi Anda AWS CDK .
- `cdk deploy`: Menerapkan resource yang ditentukan oleh satu atau beberapa tumpukan di AWS CDK aplikasi Anda. AWS

Anda dapat menentukan nama beberapa tumpukan yang akan disintesis atau digunakan dalam satu perintah. Jika aplikasi Anda hanya mendefinisikan satu tumpukan, Anda tidak perlu menentukannya.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Anda juga dapat menggunakan wildcard * (sejumlah karakter) dan? (setiap karakter tunggal) untuk mengidentifikasi tumpukan berdasarkan pola. Saat menggunakan wildcard, lampirkan pola dalam tanda kutip. Jika tidak, shell mungkin mencoba memperluasnya ke nama-nama file di direktori saat ini sebelum diteruskan ke AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Anda tidak perlu mensintesis tumpukan secara eksplisit sebelum menerapkannya; `cdk deploy` melakukan langkah ini agar Anda memastikan kode terbaru Anda diterapkan.

Untuk dokumentasi lengkap dari `cdk` perintah, lihatthe section called “AWS CDK Toolkit”.

Menggunakan TypeScript contoh dengan JavaScript

TypeScriptadalah bahasa yang kita gunakan untuk mengembangkan AWS CDK, dan itu adalah bahasa pertama yang didukung untuk mengembangkan aplikasi, begitu banyak contoh AWS CDK kode yang tersedia ditulis TypeScript. Contoh kode ini dapat menjadi sumber daya yang baik untuk

JavaScript pengembang; Anda hanya perlu menghapus bagian-bagian kode yang TypeScript spesifik.

TypeScript cuplikan sering menggunakan ECMAScript `import` dan `export` kata kunci yang lebih baru untuk mengimpor objek dari modul lain dan untuk mendeklarasikan objek yang akan tersedia di luar modul saat ini. Node.js baru saja mulai mendukung kata kunci ini dalam rilis terbarunya. Bergantung pada versi Node.js yang Anda gunakan (atau ingin mendukung), Anda dapat menulis ulang impor dan ekspor untuk menggunakan sintaks yang lebih lama.

Impor dapat diganti dengan panggilan ke `require()` fungsi.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { Bucket, BucketPolicy } from 'aws-cdk-lib/aws-s3';
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const { Bucket, BucketPolicy } = require('aws-cdk-lib/aws-s3');
```

Ekspor dapat ditugaskan ke `module.exports` objek.

TypeScript

```
export class Stack1 extends cdk.Stack {
  // ...
}

export class Stack2 extends cdk.Stack {
  // ...
}
```

JavaScript

```
class Stack1 extends cdk.Stack {
  // ...
}

class Stack2 extends cdk.Stack {
  // ...
```

```
}

module.exports = { Stack1, Stack2 }
```

> **Note**
>
> Alternatif untuk menggunakan impor dan ekspor gaya lama adalah dengan menggunakan modul. `esm`

Setelah impor dan ekspor diurutkan, Anda dapat menggali kode yang sebenarnya. Anda mungkin mengalami fitur-fitur yang umum digunakan ini TypeScript :

- Ketik anotasi
- Definisi antarmuka
- Jenis konversi/gips
- Pengubah akses

Jenis anotasi dapat disediakan untuk variabel, anggota kelas, parameter fungsi, dan tipe pengembalian fungsi. Untuk variabel, parameter, dan anggota, jenis ditentukan dengan mengikuti pengidentifikasi dengan titik dua dan tipe. Nilai pengembalian fungsi mengikuti tanda tangan fungsi dan terdiri dari titik dua dan tipe.

Untuk mengonversi kode beranotasi tipe menjadi JavaScript, hapus titik dua dan jenisnya. Anggota kelas harus memiliki beberapa nilai JavaScript; atur mereka ke `undefined` jika mereka hanya memiliki anotasi tipe di TypeScript.

TypeScript

```
var encrypted: boolean = true;

class myStack extends cdk.Stack {
    bucket: s3.Bucket;
    // ...
}

function makeEnv(account: string, region: string) : object {
    // ...
```

```
}
```

JavaScript

```
var encrypted = true;

class myStack extends cdk.Stack {
    bucket = undefined;
    // ...
}

function makeEnv(account, region) {
    // ...
}
```

Dalam TypeScript, antarmuka digunakan untuk memberikan bundel properti wajib dan opsional, dan jenisnya, nama. Anda kemudian dapat menggunakan nama antarmuka sebagai anotasi tipe. TypeScript akan memastikan bahwa objek yang Anda gunakan sebagai, misalnya, argumen ke fungsi memiliki properti yang diperlukan dari tipe yang tepat.

```
interface myFuncProps {
    code: lambda.Code,
    handler?: string
}
```

JavaScript tidak memiliki fitur antarmuka, jadi setelah Anda menghapus anotasi tipe, hapus deklarasi antarmuka sepenuhnya.

Ketika fungsi atau metode mengembalikan tipe tujuan umum (seperti`object`), tetapi Anda ingin memperlakukan nilai itu sebagai tipe anak yang lebih spesifik untuk mengakses properti atau metode yang bukan bagian dari antarmuka tipe yang lebih umum, TypeScript memungkinkan Anda mentransmisikan nilai menggunakan `as` diikuti oleh jenis atau nama antarmuka. JavaScript tidak mendukung (atau membutuhkan) ini, jadi cukup hapus `as` dan pengenal berikut. Sintaks cast yang kurang umum adalah menggunakan nama tipe dalam tanda kurung,`<LikeThis>`; gips ini juga harus dihapus.

Akhirnya, TypeScript mendukung pengubah akses`public`,`protected`, dan `private` untuk anggota kelas. Semua anggota kelas JavaScript adalah publik. Cukup hapus pengubah ini di mana pun Anda melihatnya.

Mengetahui cara mengidentifikasi dan menghapus TypeScript fitur-fitur ini sangat membantu mengadaptasi TypeScript cuplikan pendek. JavaScript Tetapi mungkin tidak praktis untuk mengonversi TypeScript contoh yang lebih panjang dengan cara ini, karena mereka lebih cenderung menggunakan TypeScript fitur lain. Untuk situasi ini, kami merekomendasikan Sucrase. Sucrase tidak akan mengeluh jika kode menggunakan variabel yang tidak ditentukan, misalnya, seperti yang akan terjadi. `tsc` Jika valid secara sintaksis, maka dengan beberapa pengecualian, Sucrase dapat menerjemahkannya ke. JavaScript Ini membuatnya sangat berharga untuk mengonversi cuplikan yang mungkin tidak dapat dijalankan sendiri.

Migrasi ke TypeScript

Banyak JavaScript pengembang pindah ke TypeScriptproyek mereka menjadi lebih besar dan lebih kompleks. TypeScript adalah superset dari JavaScript —semua JavaScript kode adalah kode yang valid, jadi tidak ada perubahan pada TypeScript kode Anda yang diperlukan—dan itu juga merupakan bahasa yang didukung. AWS CDK Jenis anotasi dan TypeScript fitur lainnya bersifat opsional dan dapat ditambahkan ke AWS CDK aplikasi Anda saat Anda menemukan nilai di dalamnya. TypeScript juga memberi Anda akses awal ke JavaScript fitur-fitur baru, seperti rantai opsional dan penggabungan nullish, sebelum selesai — dan tanpa mengharuskan Anda meningkatkan Node.js.

TypeScriptantarmuka “berbasis bentuk”, yang mendefinisikan bundel properti wajib dan opsional (dan jenisnya) dalam suatu objek, memungkinkan kesalahan umum ditangkap saat Anda menulis kode, dan memudahkan IDE Anda untuk memberikan pelengkapan otomatis yang kuat dan saran pengkodean waktu nyata lainnya.

Coding in TypeScript memang melibatkan langkah tambahan: mengompilasi aplikasi Anda dengan TypeScript compiler,. `tsc` Untuk AWS CDK aplikasi biasa, kompilasi membutuhkan waktu paling lama beberapa detik.

Cara termudah untuk memigrasikan JavaScript AWS CDK aplikasi yang sudah ada TypeScript adalah dengan membuat TypeScript proyek baru menggunakan`cdk init app --language typescript`, lalu menyalin file sumber Anda (dan file lain yang diperlukan, seperti aset seperti kode sumber AWS Lambda fungsi) ke proyek baru. Ganti nama JavaScript file Anda untuk mengakhiri `.ts` dan mulai mengembangkan. TypeScript

Bekerja dengan AWS CDK in Python

Python adalah bahasa klien yang sepenuhnya didukung untuk AWS Cloud Development Kit (AWS CDK) dan dianggap stabil. Bekerja dengan AWS CDK in Python menggunakan alat yang sudah

dikenal, termasuk implementasi Python standar (CPython), lingkungan virtual dengan, `virtualenv` dan penginstal paket Python. `pip` Modul yang terdiri dari AWS Construct Library didistribusikan melalui pypi.org. Versi Python AWS CDK bahkan menggunakan pengidentifikasi gaya Python (misalnya, nama metode). `snake_case`

Anda dapat menggunakan editor atau IDE apa pun. Banyak AWS CDK pengembang menggunakan Visual Studio Code (atau VScodium yang setara dengan sumber terbuka), yang memiliki dukungan yang baik untuk Python melalui ekstensi resmi. Editor IDLE yang disertakan dengan Python sudah cukup untuk memulai. Modul Python untuk AWS CDK do memiliki petunjuk tipe, yang berguna untuk alat linting atau IDE yang mendukung validasi tipe.

Topik

Memulai dengan Python

Untuk bekerja dengan AWS CDK, Anda harus memiliki AWS akun dan kredensional dan telah menginstal Node.js dan Toolkit. AWS CDK Lihat Memulai dengan AWS CDK.

AWS CDK Aplikasi Python membutuhkan Python 3.6 atau yang lebih baru. Jika Anda belum menginstalnya, unduh versi yang kompatibel untuk sistem operasi Anda di python.org. Jika Anda menjalankan Linux, sistem Anda mungkin datang dengan versi yang kompatibel, atau Anda dapat menginstalnya menggunakan manajer paket distro Anda (`yum`,`apt`, dll.). Pengguna Mac mungkin tertarik dengan Homebrew, pengelola paket bergaya Linux untuk macOS.

> **Note**
>
> Pengakhiran bahasa pihak ketiga: versi bahasa hanya didukung hingga EOL (End Of Life) dibagikan oleh vendor atau komunitas dan dapat berubah sewaktu-waktu dengan pemberitahuan sebelumnya.

Penginstal paket Python,`pip`, dan manajer lingkungan virtual`virtualenv`, juga diperlukan. Instalasi Windows dari versi Python yang kompatibel termasuk alat-alat ini. Di Linux, `pip` dan `virtualenv` dapat disediakan sebagai paket terpisah di manajer paket Anda. Atau, Anda dapat menginstalnya dengan perintah berikut:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

Jika Anda mengalami kesalahan izin, jalankan perintah di atas dengan `--user` bendera sehingga modul diinstal di direktori pengguna Anda, atau gunakan `sudo` untuk mendapatkan izin untuk menginstal modul di seluruh sistem.

> **Note**
>
> Hal ini umum untuk distro Linux untuk menggunakan nama executable untuk `python3` Python 3.x, dan telah merujuk `python` ke instalasi Python 2.x. Beberapa distro memiliki paket opsional yang dapat Anda instal yang membuat `python` perintah mengacu pada Python 3. Jika gagal, Anda dapat menyesuaikan perintah yang digunakan untuk menjalankan aplikasi Anda dengan mengedit `cdk.json` di direktori utama proyek.

> **Note**
>
> Di Windows, Anda mungkin ingin memanggil Python (pipdan) menggunakan executable, py peluncur >Python untuk Windows. Antara lain, peluncur memungkinkan Anda untuk dengan mudah menentukan versi Python yang diinstal yang ingin Anda gunakan.
> Jika mengetik python di baris perintah menghasilkan pesan tentang menginstal Python dari Windows Store, bahkan setelah menginstal Python versi Windows, buka panel pengaturan Alias Eksekusi Aplikasi Windows dan matikan dua entri Penginstal Aplikasi untuk Python.

Membuat proyek

Anda membuat AWS CDK proyek baru dengan memanggil `cdk init` dalam direktori kosong. Gunakan `--language` opsi dan tentukan`python`:

```
mkdir my-project
```

```
cd my-project
cdk init app --language python
```

`cdk init`menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung dalam nama folder diubah menjadi garis bawah. Namun, nama tersebut harus mengikuti bentuk pengenal Python; misalnya, seharusnya tidak dimulai dengan angka atau berisi spasi.

Untuk bekerja dengan proyek baru, aktifkan lingkungan virtualnya. Ini memungkinkan dependensi proyek diinstal secara lokal di folder proyek, bukan secara global.

```
source .venv/bin/activate
```

> **Note**
>
> Anda mungkin mengenali ini sebagai perintah Mac/Linux untuk mengaktifkan lingkungan virtual. Templat Python menyertakan file batch, `source.bat`, yang memungkinkan perintah yang sama untuk digunakan pada Windows. Perintah Windows tradisional `.venv\Scripts\activate.bat`,, juga berfungsi.
> Jika Anda menginisialisasi AWS CDK proyek Anda menggunakan CDK Toolkit v1.70.0 atau yang lebih lama, lingkungan virtual Anda ada di direktori, bukan. `.env` `.venv`

> **Important**
>
> Aktifkan lingkungan virtual proyek setiap kali Anda mulai mengerjakannya. Jika tidak, Anda tidak akan memiliki akses ke modul yang diinstal di sana, dan modul yang Anda instal akan masuk ke direktori modul global Python (atau akan menghasilkan kesalahan izin).

Setelah mengaktifkan lingkungan virtual Anda untuk pertama kalinya, instal dependensi standar aplikasi:

```
python -m pip install -r requirements.txt
```

Mengelola AWS modul Construct Library

Gunakan penginstal paket Python,pip, untuk menginstal dan memperbarui modul AWS Construct Library untuk digunakan oleh aplikasi Anda, serta paket lain yang Anda butuhkan. pipjuga menginstal dependensi untuk modul-modul tersebut secara otomatis. Jika sistem Anda tidak mengenali pip sebagai perintah mandiri, panggil pip sebagai modul Python, seperti ini:

```
python -m pip PIP-COMMAND
```

Sebagian besar AWS CDK konstruksi ada di`aws-cdk-lib`. Modul eksperimental berada dalam modul terpisah bernama like`aws-cdk.SERVICE-NAME.alpha`. Nama layanan menyertakan awalan aws. Jika Anda tidak yakin dengan nama modul, cari di PyPI. Misalnya, perintah di bawah ini menginstal AWS CodeStar perpustakaan.

```
python -m pip install aws-cdk.aws-codestar-alpha
```

Beberapa konstruksi layanan berada di lebih dari satu namespace. Misalnya, selain itu`aws-cdk.aws-route53`, ada tiga ruang nama Amazon Route 53 tambahan, bernama `aws-route53-targets`, `aws-route53-patterns`, dan. `aws-route53resolver`

> **Note**
>
> Edisi Python dari Referensi API CDK juga menunjukkan nama paket.

Nama-nama yang digunakan untuk mengimpor modul AWS Construct Library ke dalam kode Python Anda terlihat seperti berikut ini.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

Kami merekomendasikan praktik berikut saat mengimpor AWS CDK kelas dan modul AWS Construct Library dalam aplikasi Anda. Mengikuti panduan ini akan membantu membuat kode Anda konsisten dengan AWS CDK aplikasi lain serta lebih mudah dipahami.

- Umumnya, impor kelas individu dari tingkat atas`aws_cdk`.

  ```
  from aws_cdk import App, Construct
  ```

- Jika Anda membutuhkan banyak kelas dari`aws_cdk`, Anda dapat menggunakan alias namespace `cdk` alih-alih mengimpor kelas individual. Hindari melakukan keduanya.

```
import aws_cdk as cdk
```

- Umumnya, impor AWS Construct Libraries menggunakan alias namespace pendek.

```
import aws_cdk.aws_s3 as s3
```

Setelah menginstal modul, perbarui `requirements.txt` file proyek Anda, yang mencantumkan dependensi proyek Anda. Yang terbaik adalah melakukan ini secara manual daripada menggunakan`pip freeze`. `pip freeze`menangkap versi saat ini dari semua modul yang diinstal di lingkungan virtual Python Anda, yang dapat berguna saat menggabungkan proyek untuk dijalankan di tempat lain.

Namun, biasanya, Anda hanya `requirements.txt` boleh mencantumkan dependensi tingkat atas (modul yang bergantung langsung pada aplikasi Anda) dan bukan dependensi pustaka tersebut. Strategi ini membuat memperbarui dependensi Anda lebih sederhana.

Anda dapat mengedit `requirements.txt` untuk mengizinkan peningkatan; cukup ganti nomor versi == sebelumnya dengan ~= untuk memungkinkan peningkatan ke versi kompatibel yang lebih tinggi, atau hapus persyaratan versi sepenuhnya untuk menentukan versi modul terbaru yang tersedia.

Dengan `requirements.txt` diedit dengan tepat untuk memungkinkan peningkatan, keluarkan perintah ini untuk memutakhirkan modul yang diinstal proyek Anda kapan saja:

```
pip install --upgrade -r requirements.txt
```

Mengelola dependensi di Python

Dengan Python, Anda menentukan dependensi dengan memasukkannya ke dalam aplikasi atau `requirements.txt` `setup.py` untuk membangun pustaka. Dependensi kemudian dikelola dengan alat PIP. PIP dipanggil dengan salah satu cara berikut:

```
pip command options
python -m pip command options
```

python -m pipPemanggilan bekerja pada sebagian besar sistem; pip mengharuskan PIP yang dapat dieksekusi berada di jalur sistem. Jika pip tidak berhasil, coba ganti denganpython -m pip.

cdk init --language pythonPerintah menciptakan lingkungan virtual untuk proyek baru Anda. Ini memungkinkan setiap proyek memiliki versi dependensi sendiri, dan juga file dasar`requirements.txt`. Anda harus mengaktifkan lingkungan virtual ini dengan menjalankan source .venv/bin/activate setiap kali Anda mulai bekerja dengan proyek.

Aplikasi CDK

Berikut ini adalah contoh `requirements.txt` file. Karena PIP tidak memiliki fitur penguncian ketergantungan, kami menyarankan Anda menggunakan operator == untuk menentukan versi yang tepat untuk semua dependensi, seperti yang ditunjukkan di sini.

```
aws-cdk-lib==2.14.0
aws-cdk.aws-appsync-alpha==2.10.0a0
```

Menginstal modul dengan pip install tidak secara otomatis menambahkannya`requirements.txt`. Anda harus melakukannya sendiri. Jika Anda ingin meningkatkan ke versi dependensi yang lebih baru, edit nomor versinya di`requirements.txt`.

Untuk menginstal atau memperbarui dependensi proyek Anda setelah membuat atau mengedit`requirements.txt`, jalankan yang berikut ini:

```
python -m pip install -r requirements.txt
```

> **Tip**
>
> pip freezePerintah mengeluarkan versi semua dependensi yang diinstal dalam format yang dapat ditulis ke file teks. Ini dapat digunakan sebagai file persyaratan dengan`pip install -r`. File ini nyaman untuk menyematkan semua dependensi (termasuk yang transitif) ke versi persis yang Anda uji. Untuk menghindari masalah saat memutakhirkan paket nanti, gunakan file terpisah untuk ini, seperti `freeze.txt` (not`requirements.txt`). Kemudian, buat ulang saat Anda memutakhirkan dependensi proyek Anda.

Pustaka konstruksi pihak ketiga

Di pustaka, dependensi ditentukan dalam`setup.py`, sehingga dependensi transitif diunduh secara otomatis ketika paket dikonsumsi oleh aplikasi. Jika tidak, setiap aplikasi yang ingin menggunakan paket Anda perlu menyalin dependensi Anda ke dalamnya. `requirements.txt` Contoh `setup.py` ditampilkan di sini.

```
from setuptools import setup

setup(
  name='my-package',
  version='0.0.1',
  install_requires=[
    'aws-cdk-lib==2.14.0',
  ],
  ...
)
```

Untuk mengerjakan paket untuk pengembangan, buat atau aktifkan lingkungan virtual, lalu jalankan perintah berikut.

```
python -m pip install -e .
```

Meskipun PIP secara otomatis menginstal dependensi transitif, hanya ada satu salinan yang diinstal dari satu paket. Versi yang ditentukan tertinggi di pohon ketergantungan dipilih; aplikasi selalu memiliki kata terakhir dalam versi paket apa yang diinstal.

AWS CDK idiom dalam Python

Konflik bahasa

Dalam Python, `lambda` adalah kata kunci bahasa, sehingga Anda tidak dapat menggunakannya sebagai nama untuk modul pustaka AWS Lambda konstruksi atau fungsi Lambda. Konvensi Python untuk konflik tersebut adalah dengan menggunakan garis bawah, seperti pada`lambda_`, dalam nama variabel.

Menurut konvensi, argumen kedua untuk AWS CDK konstruksi diberi nama`id`. Saat menulis tumpukan dan konstruksi Anda sendiri, memanggil parameter `id` “bayangan” fungsi bawaan Python`id()`, yang mengembalikan pengenal unik objek. Fungsi ini tidak sering digunakan, tetapi jika Anda membutuhkannya dalam konstruksi Anda, ganti nama argumen, misalnya. `construct_id`

Argumen dan properti

Semua kelas AWS Construct Library dipakai menggunakan tiga argumen: lingkup di mana konstruksi sedang didefinisikan (induknya di pohon konstruksi), id, dan props, bundel pasangan kunci/nilai yang digunakan konstruksi untuk mengkonfigurasi sumber daya yang dibuatnya. Kelas dan metode lain juga menggunakan pola “bundel atribut” untuk argumen.

scope dan id harus selalu diteruskan sebagai argumen posisi, bukan argumen kata kunci, karena namanya berubah jika konstruksi menerima properti bernama scope atau id.

Dalam Python, props dinyatakan sebagai argumen kata kunci. Jika argumen berisi struktur data bersarang, ini diekspresikan menggunakan kelas yang mengambil argumen kata kunci sendiri di instantiation. Pola yang sama diterapkan pada panggilan metode lain yang mengambil argumen terstruktur.

Misalnya, dalam `add_lifecycle_rule` metode bucket Amazon S3, `transitions` properti adalah daftar instance. `Transition`

```
bucket.add_lifecycle_rule(
  transitions=[
    Transition(
      storage_class=StorageClass.GLACIER,
      transition_after=Duration.days(10)
    )
  ]
)
```

Saat memperluas kelas atau mengganti metode, Anda mungkin ingin menerima argumen tambahan untuk tujuan Anda sendiri yang tidak dipahami oleh kelas induk. Dalam hal ini Anda harus menerima argumen yang tidak Anda pedulikan menggunakan `**kwargs` idiom, dan menggunakan argumen khusus kata kunci untuk menerima argumen yang Anda minati. Saat memanggil konstruktor induk atau metode yang diganti, berikan hanya argumen yang diharapkannya (seringkali hanya). `**kwargs` Melewati argumen bahwa kelas induk atau metode tidak mengharapkan hasil kesalahan.

```
class MyConstruct(Construct):
    def __init__(self, id, *, MyProperty=42, **kwargs):
        super().__init__(self, id, **kwargs)
        # ...
```

Rilis future AWS CDK dapat secara kebetulan menambahkan properti baru dengan nama yang Anda gunakan untuk properti Anda sendiri. Ini tidak akan menyebabkan masalah teknis apa pun bagi pengguna konstruksi atau metode Anda (karena properti Anda tidak melewati "rantai", kelas induk atau metode yang diganti hanya akan menggunakan nilai default) tetapi dapat menyebabkan kebingungan. Anda dapat menghindari masalah potensial ini dengan memberi nama properti Anda sehingga mereka jelas milik konstruksi Anda. Jika ada banyak properti baru, bundel mereka ke dalam kelas yang diberi nama tepat dan meneruskannya sebagai argumen kata kunci tunggal.

Nilai yang hilang

AWS CDK Penggunaan `None` untuk mewakili nilai yang hilang atau tidak terdefinisi. Saat bekerja dengan`**kwargs`, gunakan `get()` metode kamus untuk memberikan nilai default jika properti tidak disediakan. Hindari penggunaan`kwargs[...]`, karena ini menimbulkan nilai `KeyError` yang hilang.

```
encrypted = kwargs.get("encrypted")         # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)  # specify default of False if property is
 missing
```

Beberapa AWS CDK metode (seperti `tryGetContext()` untuk mendapatkan nilai konteks runtime) mungkin kembali`None`, yang perlu Anda periksa secara eksplisit.

Menggunakan antarmuka

Python tidak memiliki fitur antarmuka seperti beberapa bahasa lain, meskipun memang memiliki kelas dasar abstrak, yang serupa. (Jika Anda tidak terbiasa dengan antarmuka, Wikipedia memiliki pengantar yang bagus.) TypeScript, bahasa di mana AWS CDK diimplementasikan, memang menyediakan antarmuka, dan konstruksi dan AWS CDK objek lainnya sering memerlukan objek yang melekat pada antarmuka tertentu, daripada mewarisi dari kelas tertentu. Jadi AWS CDK menyediakan fitur antarmuka sendiri sebagai bagian dari lapisan JSII.

Untuk menunjukkan bahwa kelas mengimplementasikan antarmuka tertentu, Anda dapat menggunakan `@jsii.implements` dekorator:

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Jenis jebakan

Python menggunakan pengetikan dinamis, di mana semua variabel dapat merujuk ke nilai dari jenis apa pun. Parameter dan nilai pengembalian dapat dianotasi dengan tipe, tetapi ini adalah “petunjuk” dan tidak diberlakukan. Ini berarti bahwa dengan Python, mudah untuk meneruskan jenis nilai yang salah ke konstruksi. AWS CDK Alih-alih mendapatkan kesalahan tipe selama pembuatan, seperti yang Anda lakukan dari bahasa yang diketik secara statis, Anda mungkin mendapatkan kesalahan runtime ketika lapisan JSII (yang menerjemahkan antara Python dan inti) tidak dapat menangani tipe yang tidak AWS CDK terduga TypeScript .

Menurut pengalaman kami, kesalahan tipe yang dilakukan programmer Python cenderung masuk dalam kategori ini.

- Melewati satu nilai di mana konstruksi mengharapkan wadah (daftar Python atau kamus) atau sebaliknya.
- Melewati nilai tipe yang terkait dengan konstruksi layer 1 (`CfnXxxxxx`) ke konstruksi L2 atau L3, atau sebaliknya.

Modul AWS CDK Python memang menyertakan anotasi tipe, sehingga Anda dapat menggunakan alat yang mendukungnya untuk membantu tipe. Jika Anda tidak menggunakan IDE yang mendukung ini, misalnya PyCharm, Anda mungkin ingin memanggil validator MyPytipe sebagai langkah dalam proses build Anda. Ada juga pemeriksa tipe runtime yang dapat meningkatkan pesan kesalahan untuk kesalahan terkait tipe.

Mensintesis dan menyebarkan

Tumpukan yang ditentukan dalam AWS CDK aplikasi Anda dapat disintesis dan diterapkan secara individual atau bersama-sama menggunakan perintah di bawah ini. Umumnya, Anda harus berada di direktori utama proyek Anda ketika Anda menerbitkannya.

- `cdk synth`: Mensintesis AWS CloudFormation template dari satu atau beberapa tumpukan di aplikasi Anda AWS CDK .
- `cdk deploy`: Menerapkan resource yang ditentukan oleh satu atau beberapa tumpukan di AWS CDK aplikasi Anda. AWS

Anda dapat menentukan nama beberapa tumpukan yang akan disintesis atau digunakan dalam satu perintah. Jika aplikasi Anda hanya mendefinisikan satu tumpukan, Anda tidak perlu menentukannya.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

Anda juga dapat menggunakan wildcard * (sejumlah karakter) dan? (setiap karakter tunggal) untuk mengidentifikasi tumpukan berdasarkan pola. Saat menggunakan wildcard, lampirkan pola dalam tanda kutip. Jika tidak, shell mungkin mencoba memperluasnya ke nama-nama file di direktori saat ini sebelum diteruskan ke AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Anda tidak perlu mensintesis tumpukan secara eksplisit sebelum menerapkannya; `cdk deploy` melakukan langkah ini agar Anda memastikan kode terbaru Anda diterapkan.

Untuk dokumentasi lengkap dari `cdk` perintah, lihatthe section called “AWS CDK Toolkit”.

Bekerja dengan AWS CDK di Jawa

Java adalah bahasa klien yang sepenuhnya didukung untuk AWS CDK dan dianggap stabil. Anda dapat mengembangkan AWS CDK aplikasi di Java menggunakan alat yang sudah dikenal, termasuk JDK (Oracle's, atau distribusi OpenJDK seperti Amazon Corretto) dan Apache Maven.

AWS CDK Mendukung Java 8 dan yang lebih baru. Kami merekomendasikan untuk menggunakan versi terbaru yang Anda bisa, karena versi bahasa yang lebih baru mencakup peningkatan yang sangat nyaman untuk mengembangkan AWS CDK aplikasi. Misalnya, Java 9 memperkenalkan `Map.of()` metode (cara mudah untuk mendeklarasikan hashmap yang akan ditulis sebagai literal objek). TypeScript Java 10 memperkenalkan inferensi tipe lokal menggunakan kata kunci. `var`

> **Note**
>
> Sebagian besar contoh kode dalam Panduan Pengembang ini bekerja dengan Java 8. Beberapa contoh digunakan`Map.of()`; contoh-contoh ini termasuk komentar yang mencatat bahwa mereka memerlukan Java 9.

Anda dapat menggunakan editor teks apa pun, atau IDE Java yang dapat membaca proyek Maven, untuk mengerjakan aplikasi Anda AWS CDK . Kami menyediakan petunjuk Eclipse dalam Panduan ini, tetapi IntelliJ IDEA, NetBeans, dan IDE lainnya dapat mengimpor proyek Maven dan dapat digunakan untuk mengembangkan aplikasi di Jawa. AWS CDK

Dimungkinkan untuk menulis AWS CDK aplikasi dalam bahasa yang dihosting JVM selain Java (misalnya, Kotlin, Groovy, Clojure, atau Scala), tetapi pengalamannya mungkin tidak terlalu idiomatis, dan kami tidak dapat memberikan dukungan apa pun untuk bahasa-bahasa ini.

Topik

Memulai dengan Java

Untuk bekerja dengan AWS CDK, Anda harus memiliki AWS akun dan kredensional dan telah menginstal Node.js dan Toolkit. AWS CDK Lihat Memulai dengan AWS CDK.

AWS CDK Aplikasi Java membutuhkan Java 8 (v1.8) atau yang lebih baru. Kami merekomendasikan Amazon Corretto, tetapi Anda dapat menggunakan distribusi OpenJDK atau JDK Oracle. Anda juga membutuhkan Apache Maven 3.5 atau yang lebih baru. Anda juga dapat menggunakan alat seperti Gradle, tetapi kerangka aplikasi yang dihasilkan oleh AWS CDK Toolkit adalah proyek Maven.

> **Note**
>
> Pengakhiran bahasa pihak ketiga: versi bahasa hanya didukung hingga EOL (End Of Life) dibagikan oleh vendor atau komunitas dan dapat berubah sewaktu-waktu dengan pemberitahuan sebelumnya.

Membuat proyek

Anda membuat AWS CDK proyek baru dengan memanggil `cdk init` dalam direktori kosong. Gunakan `--language` opsi dan tentukan`java`:

```
mkdir my-project
cd my-project
cdk init app --language java
```

`cdk init`menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung dalam nama folder diubah menjadi garis bawah. Namun, nama tersebut harus mengikuti bentuk pengenal Java; misalnya, seharusnya tidak dimulai dengan angka atau berisi spasi.

Proyek yang dihasilkan mencakup referensi ke paket `software.amazon.awscdk` Maven. Ini dan dependensinya diinstal secara otomatis oleh Maven.

Jika Anda menggunakan IDE, Anda sekarang dapat membuka atau mengimpor proyek. Di Eclipse, misalnya, pilih File > Import > Maven > Existing Maven Projects. Pastikan bahwa pengaturan proyek diatur untuk menggunakan Java 8 (1.8).

Mengelola AWS modul Construct Library

Gunakan Maven untuk menginstal paket AWS Construct Library, yang ada di grup. `software.amazon.awscdk` Sebagian besar konstruksi berada di artefak`aws-cdk-lib`, yang ditambahkan ke proyek Java baru secara default. Modul untuk layanan yang dukungan CDK tingkat tinggi masih dikembangkan berada dalam paket “eksperimental” terpisah, dinamai dengan versi pendek (tidak ada atau awalan AWS Amazon) dari nama layanan mereka. Cari Repositori Pusat Maven untuk menemukan nama semua AWS CDK dan AWS Membangun pustaka Modul.

> **Note**
>
> Edisi Java dari Referensi API CDK juga menunjukkan nama paket.

Dukungan AWS Construct Library beberapa layanan ada di lebih dari satu namespace. Misalnya, Amazon Route 53 memiliki fungsinya dibagi menjadi`software.amazon.awscdk.route53`,`route53-patterns`,`route53resolver`, dan`route53-targets`.

AWS CDK Paket utama diimpor dalam kode Java sebagai`software.amazon.awscdk`. Modul untuk berbagai layanan di Perpustakaan AWS Konstruksi hidup di bawah `software.amazon.awscdk.services` dan diberi nama yang mirip dengan nama paket Maven mereka. Misalnya, namespace modul Amazon S3 adalah. `software.amazon.awscdk.services.s3`

Sebaiknya tulis `import` pernyataan Java terpisah untuk setiap kelas AWS Construct Library yang Anda gunakan di setiap file sumber Java Anda, dan menghindari impor wildcard. Anda selalu dapat menggunakan nama tipe yang sepenuhnya memenuhi syarat (termasuk namespace-nya) tanpa pernyataan. `import`

Jika aplikasi Anda bergantung pada paket eksperimental, edit proyek Anda `pom.xml` dan tambahkan `<dependency>` elemen baru dalam `<dependencies>` wadah. Misalnya, `<dependency>` elemen berikut menentukan modul perpustakaan konstruksi CodeStar eksperimental:

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>codestar-alpha</artifactId>
    <version>2.0.0-alpha.10</version>
</dependency>
```

> **Tip**
>
> Jika Anda menggunakan IDE Java, mungkin memiliki fitur untuk mengelola dependensi Maven. Kami merekomendasikan untuk mengedit `pom.xml` secara langsung, kecuali Anda benar-benar yakin fungsionalitas IDE cocok dengan apa yang akan Anda lakukan dengan tangan.

Mengelola dependensi di Java

Di Java, dependensi ditentukan `pom.xml` dan diinstal menggunakan Maven. `<dependencies>`Wadah mencakup `<dependency>` elemen untuk setiap paket. Berikut ini adalah bagian dari `pom.xml` untuk aplikasi CDK Java yang khas.

```
<dependencies>
    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>aws-cdk-lib</artifactId>
```

```
        <version>2.14.0</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>appsync-alpha</artifactId>
        <version>2.10.0-alpha.0</version>
    </dependency>
</dependencies>
```

> **Tip**
>
> Banyak IDE Java telah mengintegrasikan dukungan Maven dan `pom.xml` editor visual, yang mungkin Anda temukan nyaman untuk mengelola dependensi.

Maven tidak mendukung penguncian ketergantungan. Meskipun dimungkinkan untuk menentukan rentang versi`pom.xml`, kami sarankan Anda selalu menggunakan versi yang tepat agar build Anda tetap dapat diulang.

Maven secara otomatis menginstal dependensi transitif, tetapi hanya ada satu salinan yang diinstal dari setiap paket. Versi yang ditentukan tertinggi di pohon POM dipilih; aplikasi selalu memiliki kata terakhir dalam versi paket apa yang diinstal.

Maven secara otomatis menginstal atau memperbarui dependensi Anda setiap kali Anda membangun (mvn compile) atau package () proyek Anda. mvn package CDK Toolkit melakukan ini secara otomatis setiap kali Anda menjalankannya, jadi umumnya tidak perlu memanggil Maven secara manual.

AWS CDK idiom di Jawa

Alat Peraga

Semua kelas AWS Construct Library dipakai menggunakan tiga argumen: lingkup di mana konstruksi sedang didefinisikan (induknya di pohon konstruksi), id, dan props, bundel pasangan kunci/nilai yang digunakan konstruksi untuk mengkonfigurasi sumber daya yang dibuatnya. Kelas dan metode lain juga menggunakan pola “bundel atribut” untuk argumen.

Di Jawa, alat peraga diekspresikan menggunakan pola Builder. Setiap tipe konstruksi memiliki tipe props yang sesuai; misalnya, `Bucket` konstruksi (yang mewakili bucket Amazon S3) mengambil sebagai props sebagai instance dari. `BucketProps`

`BucketProps`Kelas (seperti setiap kelas props AWS Construct Library) memiliki kelas dalam yang disebut. `Builder` `BucketProps.Builder`Jenis ini menawarkan metode untuk mengatur berbagai properti `BucketProps` instance. Setiap metode mengembalikan `Builder` instance, sehingga panggilan metode dapat dirantai untuk mengatur beberapa properti. Di akhir rantai, Anda memanggil `build()` untuk benar-benar menghasilkan `BucketProps` objek.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build());
```

Konstruksi, dan kelas lain yang mengambil objek seperti alat peraga sebagai argumen terakhir mereka, menawarkan jalan pintas. Kelas memiliki sendiri `Builder` yang membuat instance dan objek props dalam satu langkah. Dengan cara ini, Anda tidak perlu secara eksplisit membuat instance (misalnya) keduanya `BucketProps` dan `Bucket` —dan Anda tidak memerlukan impor untuk jenis alat peraga.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build();
```

Saat menurunkan konstruksi Anda sendiri dari konstruksi yang ada, Anda mungkin ingin menerima properti tambahan. Kami menyarankan Anda mengikuti pola pembangun ini. Namun, ini tidak sesederhana mensubklasifikasikan kelas konstruksi. Anda harus menyediakan sendiri bagian yang bergerak dari dua `Builder` kelas baru. Anda mungkin lebih suka membuat konstruksi Anda menerima satu atau lebih argumen tambahan. Anda harus memberikan konstruktor tambahan ketika argumen bersifat opsional.

Struktur generik

Di beberapa API, AWS CDK menggunakan JavaScript array atau objek yang tidak diketik sebagai input ke metode. (Lihat, misalnya, AWS CodeBuild`BuildSpec.fromObject()`metode.) Di Jawa, objek-objek ini direpresentasikan sebagai`java.util.Map<String, Object>`. Dalam kasus di mana nilai-nilai semua string, Anda dapat menggunakan`Map<String, String>`.

Java tidak menyediakan cara untuk menulis literal untuk wadah seperti yang dilakukan beberapa bahasa lain. Di Java 9 dan yang lebih baru, Anda dapat menggunakan `java.util.Map.of()`untuk dengan mudah menentukan peta hingga sepuluh entri sebaris dengan salah satu panggilan ini.

```
java.util.Map.of(
    "base-directory", "dist",
    "files", "LambdaStack.template.json"
 )
```

Untuk membuat peta dengan lebih dari sepuluh entri, gunakan `java.util.Map.ofEntries()`.

Jika Anda menggunakan Java 8, Anda dapat memberikan metode Anda sendiri yang mirip dengan ini.

JavaScript array direpresentasikan sebagai `List<Object>` atau `List<String>` di Jawa. Metode `java.util.Arrays.asList` ini nyaman untuk mendefinisikan `List` s pendek.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

Nilai yang hilang

Di Jawa, nilai yang hilang dalam AWS CDK objek seperti alat peraga diwakili oleh`null`. Anda harus secara eksplisit menguji nilai apa pun yang bisa `null` untuk memastikannya berisi nilai sebelum melakukan apa pun dengannya. Java tidak memiliki "gula sintaksis" untuk membantu menangani nilai nol seperti yang dilakukan beberapa bahasa lain. Anda mungkin menemukan Apache ObjectUtil defaultIfNulldan firstNonNullberguna dalam beberapa situasi. Atau, tulis metode pembantu statis Anda sendiri untuk membuatnya lebih mudah menangani nilai nol yang berpotensi dan membuat kode Anda lebih mudah dibaca.

Membangun, mensintesis, dan menyebarkan

AWS CDK Secara otomatis mengkompilasi aplikasi Anda sebelum menjalankannya. Namun, membangun aplikasi Anda secara manual dapat berguna untuk memeriksa kesalahan dan menjalankan pengujian. Anda dapat melakukan ini di IDE Anda (misalnya, tekan Control-B di Eclipse) atau dengan mengeluarkan `mvn compile` pada prompt perintah saat berada di direktori root proyek Anda.

Jalankan pengujian apa pun yang Anda tulis dengan menjalankan `mvn test` pada prompt perintah.

Tumpukan yang ditentukan dalam AWS CDK aplikasi Anda dapat disintesis dan diterapkan secara individual atau bersama-sama menggunakan perintah di bawah ini. Umumnya, Anda harus berada di direktori utama proyek Anda ketika Anda menerbitkannya.

- `cdk synth`: Mensintesis AWS CloudFormation template dari satu atau beberapa tumpukan di aplikasi Anda AWS CDK .
- `cdk deploy`: Menerapkan resource yang ditentukan oleh satu atau beberapa tumpukan di AWS CDK aplikasi Anda. AWS

Anda dapat menentukan nama beberapa tumpukan yang akan disintesis atau digunakan dalam satu perintah. Jika aplikasi Anda hanya mendefinisikan satu tumpukan, Anda tidak perlu menentukannya.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Anda juga dapat menggunakan wildcard * (sejumlah karakter) dan? (setiap karakter tunggal) untuk mengidentifikasi tumpukan berdasarkan pola. Saat menggunakan wildcard, lampirkan pola dalam tanda kutip. Jika tidak, shell mungkin mencoba memperluasnya ke nama-nama file di direktori saat ini sebelum diteruskan ke AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Anda tidak perlu mensintesis tumpukan secara eksplisit sebelum menerapkannya; `cdk deploy` melakukan langkah ini agar Anda memastikan kode terbaru Anda diterapkan.

Untuk dokumentasi lengkap `cdk` perintah, lihatthe section called “AWS CDK Toolkit”.

Bekerja dengan AWS CDK di C

.NET adalah bahasa klien yang sepenuhnya didukung untuk AWS CDK dan dianggap stabil. C # adalah bahasa.NET utama yang kami berikan contoh dan dukungannya. Anda dapat memilih untuk menulis AWS CDK aplikasi dalam bahasa.NET lainnya, seperti Visual Basic atau F #, tetapi AWS menawarkan dukungan terbatas untuk menggunakan bahasa ini dengan CDK.

Anda dapat mengembangkan AWS CDK aplikasi di C # menggunakan alat yang sudah dikenal termasuk Visual Studio, Visual Studio Code, `dotnet` perintah, dan manajer NuGet paket. Modul yang terdiri dari AWS Construct Library didistribusikan melalui nuget.org.

Kami menyarankan menggunakan Visual Studio 2019 (edisi apa pun) di Windows untuk mengembangkan AWS CDK aplikasi di C #.

Topik

Memulai dengan C#

Untuk bekerja dengan AWS CDK, Anda harus memiliki AWS akun dan kredensional dan telah menginstal Node.js dan Toolkit. AWS CDK Lihat Memulai dengan AWS CDK.

AWS CDK Aplikasi C # membutuhkan .NET Core v3.1 atau yang lebih baru, tersedia di sini.

Rantai alat .NET mencakup`dotnet`, alat baris perintah untuk membangun dan menjalankan aplikasi.NET dan mengelola paket. NuGet Bahkan jika Anda bekerja terutama di Visual Studio, perintah ini dapat berguna untuk operasi batch dan untuk menginstal paket AWS Construct Library.

Membuat proyek

Anda membuat AWS CDK proyek baru dengan memanggil `cdk init` dalam direktori kosong. Gunakan `--language` opsi dan tentukan`csharp`:

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init`menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung dalam nama folder diubah menjadi garis bawah. Namun, nama tersebut harus mengikuti bentuk pengenal C#; misalnya, seharusnya tidak dimulai dengan angka atau berisi spasi.

Proyek yang dihasilkan mencakup referensi ke `Amazon.CDK.Lib` NuGet paket. Itu dan dependensinya diinstal secara otomatis oleh. NuGet

Mengelola AWS modul Construct Library

Ekosistem .NET menggunakan manajer NuGet paket. Paket CDK utama, yang berisi kelas inti dan semua konstruksi layanan yang stabil, adalah. `Amazon.CDK.Lib` Modul eksperimental, di mana fungsionalitas baru sedang dalam pengembangan aktif`Amazon.CDK.AWS.SERVICE-NAME.Alpha`, diberi nama seperti, di mana nama layanan adalah nama pendek tanpa awalan AWS atau Amazon. Misalnya, nama NuGet paket untuk AWS IoT modul adalah`Amazon.CDK.AWS.IoT.Alpha`. Jika Anda tidak dapat menemukan paket yang Anda inginkan, cari Nuget.org.

> **Note**
>
> Edisi .NET dari Referensi API CDK juga menunjukkan nama paket.

Dukungan AWS Construct Library beberapa layanan ada di lebih dari satu modul. Misalnya, AWS IoT memiliki modul kedua bernama`Amazon.CDK.AWS.IoT.Actions.Alpha`.

Modul utama, yang Anda perlukan di sebagian besar AWS CDK aplikasi, diimpor dalam kode C # sebagai`Amazon.CDK`. AWS CDK Modul untuk berbagai layanan di Perpustakaan AWS Konstruksi hidup di bawah`Amazon.CDK.AWS`. Misalnya, namespace modul Amazon S3 adalah. `Amazon.CDK.AWS.S3`

Kami merekomendasikan menulis `using` arahan C# untuk konstruksi inti CDK dan untuk setiap AWS layanan yang Anda gunakan di setiap file sumber C# Anda. Anda mungkin merasa nyaman menggunakan alias untuk namespace atau ketik untuk membantu menyelesaikan konflik nama. Anda selalu dapat menggunakan nama tipe yang sepenuhnya berkualitas (termasuk namespace-nya) tanpa pernyataan. `using`

Mengelola dependensi di C#

Di AWS CDK aplikasi C#, Anda mengelola dependensi menggunakan. NuGet NuGet memiliki empat antarmuka standar, sebagian besar setara. Gunakan salah satu yang sesuai dengan kebutuhan dan gaya kerja Anda. Anda juga dapat menggunakan alat yang kompatibel, seperti Paket atau MyGetatau bahkan mengedit `.csproj` file secara langsung.

NuGet tidak membiarkan Anda menentukan rentang versi untuk dependensi. Setiap dependensi disematkan ke versi tertentu.

Setelah memperbarui dependensi Anda, Visual Studio akan digunakan NuGet untuk mengambil versi yang ditentukan dari setiap paket saat Anda membangun berikutnya. Jika Anda tidak menggunakan Visual Studio, gunakan dotnet restore perintah untuk memperbarui dependensi Anda.

Mengedit file proyek secara langsung

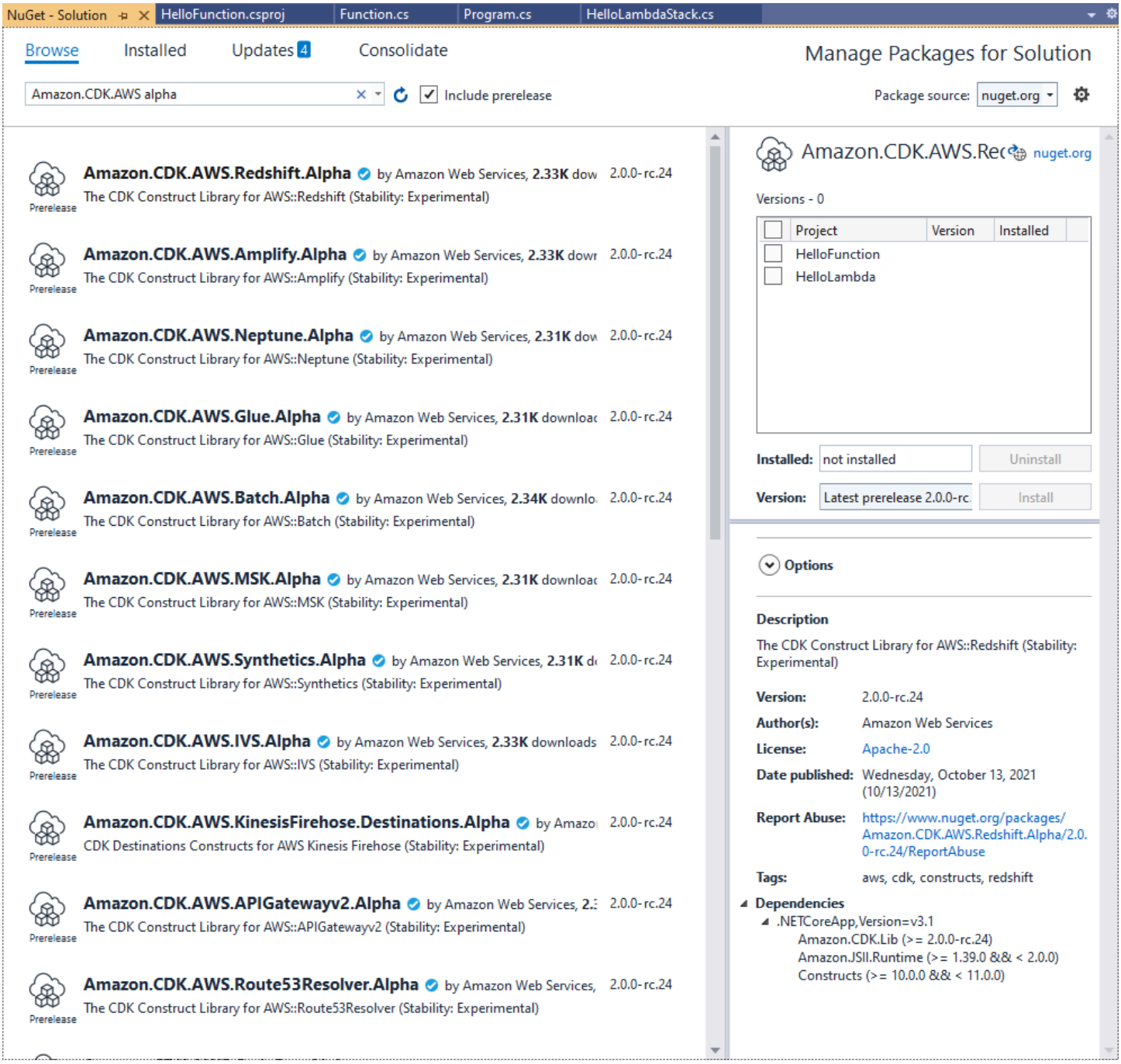
`.csproj`File proyek Anda berisi `<ItemGroup>` wadah yang mencantumkan dependensi Anda sebagai `<PackageReference` elemen.

```
<ItemGroup>
    <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
    <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

NuGet GUI Visual Studio

Alat Visual Studio dapat diakses dari NuGet Tools > NuGet Package Manager > Manage NuGet Packages for Solution. Gunakan tab Browse untuk menemukan paket AWS Construct Library yang ingin Anda instal. Anda dapat memilih versi yang diinginkan, termasuk versi prarilis modul Anda, dan menambahkannya ke salah satu proyek terbuka.

> **Note**
>
> Semua modul AWS Construct Library yang dianggap “eksperimental” (lihatthe section called “Versioning”) ditandai sebagai prarilis dan memiliki akhiran nama. NuGet `alpha`



Lihat di halaman Pembaruan untuk menginstal versi baru paket Anda.

NuGet Konsol

NuGet Konsol adalah antarmuka PowerShell berbasis NuGet yang berfungsi dalam konteks proyek Visual Studio. Anda dapat membukanya di Visual Studio dengan memilih Tools > NuGet Package Manager > Package Manager Console. Untuk informasi selengkapnya tentang penggunaan alat ini, lihat Menginstal dan Mengelola Paket dengan Package Manager Console di Visual Studio.

`dotnet`Perintah

`dotnet`Perintah adalah alat baris perintah utama untuk bekerja dengan proyek Visual Studio C #. Anda dapat memanggilnya dari prompt perintah Windows apa pun. Di antara banyak kemampuannya, `dotnet` dapat menambahkan NuGet dependensi ke proyek Visual Studio.

Dengan asumsi Anda berada di direktori yang sama dengan file proyek Visual Studio (`.csproj`), keluarkan perintah seperti berikut untuk menginstal paket. Karena pustaka CDK utama disertakan saat Anda membuat proyek, Anda hanya perlu menginstal modul eksperimental secara eksplisit. Modul eksperimental mengharuskan Anda untuk menentukan nomor versi eksplisit.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Anda dapat mengeluarkan perintah dari direktori lain. Untuk melakukannya, sertakan jalur ke file proyek, atau ke direktori yang berisi itu, setelah `add` kata kunci. Contoh berikut mengasumsikan bahwa Anda berada di direktori utama AWS CDK proyek Anda.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Untuk menginstal versi paket tertentu, sertakan `-v` bendera dan versi yang diinginkan.

Untuk memperbarui paket, keluarkan `dotnet add` perintah yang sama yang Anda gunakan untuk menginstalnya. Untuk modul eksperimental, sekali lagi, Anda harus menentukan nomor versi eksplisit.

Untuk informasi selengkapnya tentang mengelola paket menggunakan `dotnet` perintah, lihat Menginstal dan Mengelola Paket Menggunakan CLI dotnet.

`nuget`Perintah

Alat baris `nuget` perintah dapat menginstal dan memperbarui NuGet paket. Namun, proyek Visual Studio Anda harus diatur secara berbeda dari cara `cdk init` menyiapkan proyek. (Detail teknis: `nuget` bekerja dengan `Packages.config` proyek, sambil `cdk init` membuat `PackageReference` proyek bergaya baru.)

Kami tidak merekomendasikan penggunaan `nuget` alat dengan AWS CDK proyek yang dibuat oleh`cdk init`. Jika Anda menggunakan jenis proyek lain, dan ingin menggunakannya`nuget`, lihat Referensi NuGet CLI.

AWS CDK idiom dalam C

Alat Peraga

Semua kelas AWS Construct Library dipakai menggunakan tiga argumen: lingkup di mana konstruksi sedang didefinisikan (induknya di pohon konstruksi), id, dan props, bundel pasangan kunci/nilai yang digunakan konstruksi untuk mengkonfigurasi sumber daya yang dibuatnya. Kelas dan metode lain juga menggunakan pola “bundel atribut” untuk argumen.

Dalam C #, alat peraga diekspresikan menggunakan tipe alat peraga. Dengan gaya C# idiomatik, kita dapat menggunakan penginisialisasi objek untuk mengatur berbagai properti. Di sini kita membuat bucket Amazon S3 menggunakan `Bucket` konstruksinya; jenis alat peraga yang sesuai adalah. `BucketProps`

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
    Versioned = true
});
```

> **Tip**
>
> Tambahkan paket ke proyek Anda `Amazon.JSII.Analyzers` untuk mendapatkan pemeriksaan nilai yang diperlukan dalam definisi alat peraga Anda di dalam Visual Studio.

Saat memperluas kelas atau mengganti metode, Anda mungkin ingin menerima alat peraga tambahan untuk tujuan Anda sendiri yang tidak dipahami oleh kelas induk. Untuk melakukan ini, subkelas jenis alat peraga yang sesuai dan tambahkan atribut baru.

```
// extend BucketProps for use with MimeBucket
class MimeBucketProps : BucketProps {
    public string MimeType { get; set; }
}

// hypothetical bucket that enforces MIME type of objects inside it
class MimeBucket : Bucket {
     public MimeBucket( readonly Construct scope, readonly string id, readonly
 MimeBucketProps props=null) : base(scope, id, props) {
         // ...
     }
```

```
}

// instantiate our MimeBucket class
var bucket = new MimeBucket(this, "MyBucket", new MimeBucketProps {
    Versioned = true,
    MimeType = "image/jpeg"
});
```

Saat memanggil initializer kelas induk atau metode overridden, Anda biasanya dapat meneruskan props yang Anda terima. Tipe baru kompatibel dengan induknya, dan alat peraga tambahan yang Anda tambahkan diabaikan.

Rilis future AWS CDK dapat secara kebetulan menambahkan properti baru dengan nama yang Anda gunakan untuk properti Anda sendiri. Ini tidak akan menyebabkan masalah teknis apa pun menggunakan konstruksi atau metode Anda (karena properti Anda tidak melewati "naik rantai", kelas induk atau metode yang diganti hanya akan menggunakan nilai default) tetapi dapat menyebabkan kebingungan bagi pengguna konstruksi Anda. Anda dapat menghindari masalah potensial ini dengan memberi nama properti Anda sehingga mereka jelas milik konstruksi Anda. Jika ada banyak properti baru, bundel mereka ke dalam kelas dengan nama yang tepat dan meneruskannya sebagai properti tunggal.

Struktur generik

Di beberapa API, AWS CDK menggunakan JavaScript array atau objek yang tidak diketik sebagai masukan ke metode. (Lihat, misalnya, AWS CodeBuild`BuildSpec.fromObject()`metode.) Dalam C #, objek-objek ini direpresentasikan sebagai`System.Collections.Generic.Dictionary<String, Object>`. Dalam kasus di mana nilai-nilai semua string, Anda dapat menggunakan`Dictionary<String, String>`. JavaScript array direpresentasikan sebagai `object[]` atau tipe `string[]` array di C #.

> **Tip**
>
> Anda dapat menentukan alias pendek untuk membuatnya lebih mudah untuk bekerja dengan jenis kamus tertentu.
>
> ```
> using StringDict = System.Collections.Generic.Dictionary<string, string>;
> using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
> ```

Nilai yang hilang

Dalam C #, nilai yang hilang dalam AWS CDK objek seperti alat peraga diwakili oleh. `null` Operator akses anggota bersyarat nol `?.` dan operator penggabungan nol nyaman untuk bekerja dengan nilai-nilai ini. `??`

```
// mimeType is null if props is null or if props.MimeType is null
string mimeType = props?.MimeType;

// mimeType defaults to text/plain. either props or props.MimeType can be null
string MimeType = props?.MimeType ?? "text/plain";
```

Membangun, mensintesis, dan menyebarkan

AWS CDK Secara otomatis mengompilasi aplikasi Anda sebelum menjalankannya. Namun, membangun aplikasi Anda secara manual dapat berguna untuk memeriksa kesalahan dan menjalankan pengujian. Anda dapat melakukan ini dengan menekan F6 di Visual Studio atau dengan mengeluarkan `dotnet build src` dari baris perintah, di mana `src` direktori di direktori proyek Anda yang berisi file Visual Studio Solution (`.sln`).

Tumpukan yang ditentukan dalam AWS CDK aplikasi Anda dapat disintesis dan diterapkan secara individual atau bersama-sama menggunakan perintah di bawah ini. Umumnya, Anda harus berada di direktori utama proyek Anda ketika Anda menerbitkannya.

- `cdk synth`: Mensintesis AWS CloudFormation template dari satu atau beberapa tumpukan di aplikasi Anda AWS CDK .
- `cdk deploy`: Menerapkan resource yang ditentukan oleh satu atau beberapa tumpukan di AWS CDK aplikasi Anda. AWS

Anda dapat menentukan nama beberapa tumpukan yang akan disintesis atau digunakan dalam satu perintah. Jika aplikasi Anda hanya mendefinisikan satu tumpukan, Anda tidak perlu menentukannya.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

Anda juga dapat menggunakan wildcard * (sejumlah karakter) dan? (setiap karakter tunggal) untuk mengidentifikasi tumpukan berdasarkan pola. Saat menggunakan wildcard, lampirkan pola dalam

tanda kutip. Jika tidak, shell mungkin mencoba memperluasnya ke nama-nama file di direktori saat ini sebelum diteruskan ke AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Anda tidak perlu mensintesis tumpukan secara eksplisit sebelum menerapkannya; `cdk deploy` melakukan langkah ini agar Anda memastikan kode terbaru Anda diterapkan.

Untuk dokumentasi lengkap dari `cdk` perintah, lihatthe section called “AWS CDK Toolkit”.

Bekerja dengan AWS CDK in Go

Go adalah bahasa klien yang didukung penuh untuk AWS Cloud Development Kit (AWS CDK) dan dianggap stabil. Bekerja dengan AWS CDK in Go menggunakan alat yang sudah dikenal. Versi Go AWS CDK bahkan menggunakan pengidentifikasi gaya GO.

Berbeda dengan bahasa lain yang didukung CDK, Go bukanlah bahasa pemrograman berorientasi objek tradisional. Go menggunakan komposisi di mana bahasa lain sering memanfaatkan pewarisan. Kami telah mencoba menggunakan pendekatan Go idiomatik sebanyak mungkin, tetapi ada tempat di mana CDK mungkin berbeda.

Topik ini memberikan panduan saat bekerja dengan AWS CDK in Go. Lihat posting blog pengumuman untuk panduan proyek Go sederhana untuk. AWS CDK

Topik

- Memulai dengan Go
- Membuat proyek
- Mengelola AWS modul Construct Library
- Mengelola dependensi di Go
- AWS CDK idiom di Go
- Membangun, mensintesis, dan menyebarkan

Memulai dengan Go

Untuk bekerja dengan AWS CDK, Anda harus memiliki AWS akun dan kredensional dan telah menginstal Node.js dan Toolkit. AWS CDK Lihat Memulai dengan AWS CDK.

Ikatan Go untuk AWS CDK menggunakan toolchain Go standar, v1.18 atau yang lebih baru. Anda dapat menggunakan editor pilihan Anda.

> **Note**
>
> Pengakhiran bahasa pihak ketiga: versi bahasa hanya didukung hingga EOL (End Of Life) dibagikan oleh vendor atau komunitas dan dapat berubah sewaktu-waktu dengan pemberitahuan sebelumnya.

Membuat proyek

Anda membuat AWS CDK proyek baru dengan memanggil `cdk init` dalam direktori kosong. Gunakan `--language` opsi dan tentukango:

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init`menggunakan nama folder proyek untuk memberi nama berbagai elemen proyek, termasuk kelas, subfolder, dan file. Tanda hubung dalam nama folder diubah menjadi garis bawah. Namun, nama tersebut harus mengikuti bentuk pengenal Go; misalnya, seharusnya tidak dimulai dengan angka atau berisi spasi.

Proyek yang dihasilkan mencakup referensi ke modul inti AWS CDK Go,`github.com/aws/aws-cdk-go/awscdk/v2`, digo`.mod`. Masalah go get untuk menginstal ini dan modul lain yang diperlukan.

Mengelola AWS modul Construct Library

Dalam sebagian besar AWS CDK dokumentasi dan contoh, kata “modul” sering digunakan untuk merujuk pada modul AWS Construct Library, satu atau lebih per AWS layanan, yang berbeda dari penggunaan istilah Go idiomatik. CDK Construct Library disediakan dalam satu modul Go dengan

modul Construct Library individual, yang mendukung berbagai AWS layanan, disediakan sebagai paket Go dalam modul itu.

Dukungan AWS Construct Library beberapa layanan ada di lebih dari satu modul Construct Library (paket Go). Misalnya, Amazon Route 53 memiliki tiga modul Construct Library selain `awsroute53` paket utama, bernama `awsroute53patternsawsroute53resolver`, dan`awsroute53targets`.

Paket AWS CDK inti, yang Anda perlukan di sebagian besar AWS CDK aplikasi, diimpor dalam kode Go sebagai`github.com/aws/aws-cdk-go/awscdk/v2`. Paket untuk berbagai layanan di Perpustakaan AWS Konstruksi hidup di bawah`github.com/aws/aws-cdk-go/awscdk/v2`. Misalnya, namespace modul Amazon S3 adalah. `github.com/aws/aws-cdk-go/awscdk/v2/awss3`

```
import (
        "github.com/aws/aws-cdk-go/awscdk/v2/awss3"
        // ...
)
```

Setelah Anda mengimpor modul Construct Library (paket Go) untuk layanan yang ingin Anda gunakan di aplikasi Anda, Anda mengakses konstruksi dalam modul tersebut menggunakan, misalnya,. `awss3.Bucket`

Mengelola dependensi di Go

Di Go, versi dependensi didefinisikan dalam. `go.mod` `go.mod`Defaultnya mirip dengan yang ditampilkan di sini.

```
module my-package

go 1.16

require (
  github.com/aws/aws-cdk-go/awscdk/v2 v2.16.0
  github.com/aws/constructs-go/constructs/v10 v10.0.5
  github.com/aws/jsii-runtime-go v1.29.0
)
```

Nama Package (modul, dalam bahasa Go) ditentukan oleh URL dengan nomor versi yang diperlukan ditambahkan. Sistem modul Go tidak mendukung rentang versi.

Keluarkan go get perintah untuk menginstal semua modul dan pembaruan yang diperlukan`go.mod`. Untuk melihat daftar pembaruan yang tersedia untuk dependensi Anda, masalah. go list -m -u all

AWS CDK idiom di Go

Nama bidang dan metode

Nama bidang dan metode menggunakan selubung unta (`likeThis`) di TypeScript, bahasa asal CDK. Di Go, ini mengikuti konvensi Go, begitu juga Pascal-cased (). `LikeThis`

Membersihkan

Dalam `main` metode Anda, gunakan `defer jsii.Close()` untuk memastikan aplikasi CDK Anda membersihkan setelahnya sendiri.

Nilai dan konversi pointer yang hilang

Di Go, nilai yang hilang dalam AWS CDK objek seperti bundel properti diwakili oleh`nil`. Go tidak memiliki tipe nullable; satu-satunya tipe yang dapat berisi `nil` adalah pointer. Untuk memungkinkan nilai menjadi opsional, maka, semua properti CDK, argumen, dan nilai pengembalian adalah pointer, bahkan untuk tipe primitif. Ini berlaku untuk nilai yang diperlukan serta nilai opsional, jadi jika nilai yang diperlukan nanti menjadi opsional, tidak diperlukan perubahan tipe yang melanggar.

Saat melewati nilai atau ekspresi literal, gunakan fungsi pembantu berikut untuk membuat pointer ke nilai.

- `jsii.String`
- `jsii.Number`
- `jsii.Bool`
- `jsii.Time`

Untuk konsistensi, kami menyarankan Anda menggunakan pointer dengan cara yang sama saat mendefinisikan konstruksi Anda sendiri, meskipun mungkin tampak lebih nyaman untuk, misalnya, menerima konstruksi Anda `id` sebagai string daripada penunjuk ke string.

Saat berhadapan dengan AWS CDK nilai opsional, termasuk nilai primitif serta tipe kompleks, Anda harus menguji pointer secara eksplisit untuk memastikannya tidak `nil` sebelum melakukan apa pun dengannya. Go tidak memiliki “gula sintaksis” untuk membantu menangani nilai kosong atau hilang

seperti yang dilakukan beberapa bahasa lain. Namun, nilai yang diperlukan dalam bundel properti dan struktur serupa dijamin ada (konstruksi gagal jika tidak), jadi nilai ini tidak perlu diperiksa`nil`.

Konstruksi dan Alat Peraga

Konstruksi, yang mewakili satu atau lebih AWS sumber daya dan atribut terkait, direpresentasikan dalam Go sebagai antarmuka. Misalnya, `awss3.Bucket` adalah antarmuka. Setiap konstruksi memiliki fungsi pabrik, sepertia`wss3.NewBucket`, untuk mengembalikan struct yang mengimplementasikan antarmuka yang sesuai.

Semua fungsi pabrik mengambil tiga argumen: `scope` di mana konstruksi sedang didefinisikan (induknya di pohon konstruksi), sebuah, dan `idprops`, bundel pasangan kunci/nilai yang digunakan konstruksi untuk mengonfigurasi sumber daya yang dibuatnya. Pola “bundel atribut” juga digunakan di tempat lain di AWS CDK.

Di Go, alat peraga diwakili oleh tipe struct tertentu untuk setiap konstruksi. Misalnya, `awss3.Bucket` mengambil argumen props tipe`awss3.BucketProps`. Gunakan struct literal untuk menulis argumen props.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

Struktur generik

Di beberapa tempat, AWS CDK menggunakan JavaScript array atau objek yang tidak diketik sebagai masukan ke metode. (Lihat, misalnya, AWS CodeBuild`BuildSpec.fromObject()`metode.) Di Go, objek-objek ini direpresentasikan sebagai irisan dan antarmuka kosong, masing-masing.

CDK menyediakan fungsi pembantu variadik seperti `jsii.Strings` untuk membangun irisan yang mengandung tipe primitif.

```
jsii.Strings("One", "Two", "Three")
```

Mengembangkan konstruksi khusus

Di Go, biasanya lebih mudah untuk menulis konstruksi baru daripada memperluas yang sudah ada. Pertama, tentukan tipe struct baru, sematkan secara anonim satu atau lebih tipe yang ada jika semantik seperti ekstensi diinginkan. Tulis metode untuk fungsionalitas baru apa pun yang Anda tambahkan dan bidang yang diperlukan untuk menyimpan data yang mereka butuhkan.

Tentukan antarmuka alat peraga jika konstruksi Anda membutuhkannya. Akhirnya, tulis fungsi pabrik `NewMyConstruct()` untuk mengembalikan instance konstruksi Anda.

Jika Anda hanya mengubah beberapa nilai default pada konstruksi yang ada atau menambahkan perilaku sederhana di instantiation, Anda tidak memerlukan semua pipa ledeng itu. Sebagai gantinya, tulis fungsi pabrik yang memanggil fungsi pabrik dari konstruksi yang Anda "perluas." Dalam bahasa CDK lainnya, misalnya, Anda dapat membuat `TypedBucket` konstruksi yang menerapkan tipe objek di bucket Amazon S3 dengan mengganti tipe dan, di penginisialisasi `s3.Bucket` tipe baru Anda, menambahkan kebijakan bucket yang hanya mengizinkan ekstensi nama file tertentu untuk ditambahkan ke bucket. Di Go, lebih mudah untuk hanya menulis `NewTypedBucket` yang mengembalikan `s3.Bucket` (menggunakan instantiated`s3.NewBucket`) yang telah Anda tambahkan kebijakan bucket yang sesuai. Tidak ada jenis konstruksi baru yang diperlukan karena fungsionalitas sudah tersedia dalam konstruksi bucket standar; "konstruksi" baru hanya menyediakan cara yang lebih sederhana untuk mengonfigurasinya.

Membangun, mensintesis, dan menyebarkan

AWS CDK Secara otomatis mengompilasi aplikasi Anda sebelum menjalankannya. Namun, membangun aplikasi Anda secara manual dapat berguna untuk memeriksa kesalahan dan menjalankan pengujian. Anda dapat melakukan ini dengan mengeluarkan `go build` pada prompt perintah saat berada di direktori root proyek Anda.

Jalankan pengujian apa pun yang telah Anda tulis dengan menjalankan `go test` pada prompt perintah.

Tumpukan yang ditentukan dalam AWS CDK aplikasi Anda dapat disintesis dan diterapkan secara individual atau bersama-sama menggunakan perintah di bawah ini. Umumnya, Anda harus berada di direktori utama proyek Anda ketika Anda menerbitkannya.

- `cdk synth`: Mensintesis AWS CloudFormation template dari satu atau beberapa tumpukan di aplikasi Anda AWS CDK .
- `cdk deploy`: Menerapkan resource yang ditentukan oleh satu atau beberapa tumpukan di AWS CDK aplikasi Anda. AWS

Anda dapat menentukan nama beberapa tumpukan yang akan disintesis atau digunakan dalam satu perintah. Jika aplikasi Anda hanya mendefinisikan satu tumpukan, Anda tidak perlu menentukannya.

```
cdk synth                # app defines single stack
```

```
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Anda juga dapat menggunakan wildcard * (sejumlah karakter) dan? (setiap karakter tunggal) untuk mengidentifikasi tumpukan berdasarkan pola. Saat menggunakan wildcard, lampirkan pola dalam tanda kutip. Jika tidak, shell mungkin mencoba memperluasnya ke nama-nama file di direktori saat ini sebelum diteruskan ke AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Anda tidak perlu mensintesis tumpukan secara eksplisit sebelum menerapkannya; `cdk deploy` melakukan langkah ini agar Anda memastikan kode terbaru Anda diterapkan.

Untuk dokumentasi lengkap dari `cdk` perintah, lihatthe section called “AWS CDK Toolkit”.

Mengembangkan AWS CDK aplikasi

Kembangkan AWS Cloud Development Kit (AWS CDK) aplikasi.

Topik

Menyesuaikan konstruksi dari Construct Library AWS

Sesuaikan konstruksi dari AWS Construct Library melalui escape hatch, raw override, dan custom resource.

Topik

Menggunakan palka pelarian

The AWS Construct Library menyediakan konstruksi dari berbagai tingkat abstraksi.

Pada tingkat tertinggi, AWS CDK aplikasi Anda dan tumpukan di dalamnya sendiri merupakan abstraksi dari seluruh infrastruktur cloud Anda, atau bagian signifikan darinya. Mereka dapat diparameterisasi untuk menerapkannya di lingkungan yang berbeda atau untuk kebutuhan yang berbeda.

Abstraksi adalah alat yang ampuh untuk merancang dan mengimplementasikan aplikasi cloud. AWS CDK Ini memberi Anda kekuatan tidak hanya untuk membangun dengan abstraksinya, tetapi juga untuk membuat abstraksi baru. Menggunakan konstruksi L2 dan L3 open-source yang ada sebagai panduan, Anda dapat membangun konstruksi L2 dan L3 Anda sendiri untuk mencerminkan praktik dan pendapat terbaik organisasi Anda sendiri.

Tidak ada abstraksi yang sempurna, dan bahkan abstraksi yang baik tidak dapat mencakup setiap kasus penggunaan yang mungkin. Selama pengembangan, Anda mungkin menemukan konstruksi yang hampir sesuai dengan kebutuhan Anda, membutuhkan kustomisasi kecil atau besar.

Untuk alasan ini, AWS CDK menyediakan cara untuk keluar dari model konstruksi. Ini termasuk pindah ke abstraksi tingkat yang lebih rendah atau ke model yang berbeda sepenuhnya. Melarikan diri memungkinkan Anda melarikan diri dari AWS CDK paradigma dan menyesuaikannya dengan cara yang sesuai dengan kebutuhan Anda. Kemudian, Anda dapat membungkus perubahan Anda dalam konstruksi baru untuk mengabstraksikan kompleksitas yang mendasarinya dan menyediakan API bersih untuk pengembang lain.

Berikut ini adalah contoh situasi di mana Anda dapat menggunakan palka pelarian:

- Fitur AWS layanan tersedia melalui AWS CloudFormation, tetapi tidak ada konstruksi L2 untuk itu.
- Fitur AWS layanan tersedia melalui AWS CloudFormation, dan ada konstruksi L2 untuk layanan ini, tetapi ini belum mengekspos fitur tersebut. Karena konstruksi L2 dikuratori oleh tim CDK, mereka mungkin tidak segera tersedia untuk fitur baru.
- Fitur ini belum tersedia AWS CloudFormation sama sekali.

  Untuk menentukan apakah fitur tersedia melalui AWS CloudFormation, lihat Referensi Jenis AWS Sumber Daya dan Properti.

Kembangkan lubang keluar untuk konstruksi L1

Jika konstruksi L2 tidak tersedia untuk layanan, Anda dapat menggunakan konstruksi L1 yang dihasilkan secara otomatis. Sumber daya ini dapat dikenali dari namanya dimulai dengan`Cfn`, seperti `CfnBucket` atau`CfnRole`. Anda membuat instance persis seperti Anda akan menggunakan sumber daya yang setara AWS CloudFormation .

Misalnya, untuk membuat instance bucket Amazon S3 L1 tingkat rendah dengan analitik diaktifkan, Anda akan menulis sesuatu seperti berikut ini.

TypeScript

```
new s3.CfnBucket(this, 'MyBucket', {
  analyticsConfigurations: [
    {
      id: 'Config',
      // ...
    }
  ]
});
```

JavaScript

```
new s3.CfnBucket(this, 'MyBucket', {
  analyticsConfigurations: [
    {
      id: 'Config'
      // ...
    }
  ]
});
```

Python

```
s3.CfnBucket(self, "MyBucket",
    analytics_configurations: [
        dict(id="Config",
             # ...
             )
    ]
)
```

Java

```
CfnBucket.Builder.create(this, "MyBucket")
    .analyticsConfigurations(Arrays.asList(java.util.Map.of(    // Java 9 or later
        "id", "Config", // ...
    ))).build();
```

C#

```
new CfnBucket(this, 'MyBucket', new CfnBucketProps {
```

```
    AnalyticsConfigurations = new Dictionary<string, string>
    {
        ["id"] = "Config",
        // ...
    }
});
```

Mungkin ada kasus yang jarang terjadi di mana Anda ingin mendefinisikan sumber daya yang tidak memiliki `CfnXxx` kelas yang sesuai. Ini bisa menjadi jenis sumber daya baru yang belum dipublikasikan dalam spesifikasi AWS CloudFormation sumber daya. Dalam kasus seperti ini, Anda dapat membuat instance `cdk.CfnResource` secara langsung dan menentukan jenis sumber daya dan properti. Seperti yang ditunjukkan dalam contoh berikut.

TypeScript

```
new cdk.CfnResource(this, 'MyBucket', {
  type: 'AWS::S3::Bucket',
  properties: {
    // Note the PascalCase here! These are CloudFormation identifiers.
    AnalyticsConfigurations: [
      {
        Id: 'Config',
        // ...
      }
    ]
  }
});
```

JavaScript

```
new cdk.CfnResource(this, 'MyBucket', {
  type: 'AWS::S3::Bucket',
  properties: {
    // Note the PascalCase here! These are CloudFormation identifiers.
    AnalyticsConfigurations: [
      {
        Id: 'Config'
        // ...
      }
    ]
  }
```

```
});
```

Python

```
cdk.CfnResource(self, 'MyBucket',
  type="AWS::S3::Bucket",
  properties=dict(
    # Note the PascalCase here! These are CloudFormation identifiers.
    "AnalyticsConfigurations": [
      {
        "Id": "Config",
        # ...
      }
    ]
  }
)
```

Java

```
CfnResource.Builder.create(this, "MyBucket")
        .type("AWS::S3::Bucket")
        .properties(java.util.Map.of(    // Map.of requires Java 9 or later
            // Note the PascalCase here! These are CloudFormation identifiers
            "AnalyticsConfigurations", Arrays.asList(
                    java.util.Map.of("Id", "Config", // ...
                    ))))
        .build();
```

C#

```
new CfnResource(this, "MyBucket", new CfnResourceProps
{
    Type = "AWS::S3::Bucket",
    Properties = new Dictionary<string, object>
    {   // Note the PascalCase here! These are CloudFormation identifiers
        ["AnalyticsConfigurations"] = new Dictionary<string, string>[]
        {
            new Dictionary<string, string> {
                ["Id"] = "Config"
            }
        }
    }
```

```
});
```

Kembangkan lubang keluar untuk konstruksi L2

Jika konstruksi L2 tidak memiliki fitur atau Anda mencoba mengatasi masalah, Anda dapat memodifikasi konstruksi L1 yang dienkapsulasi oleh konstruksi L2.

Semua konstruksi L2 berisi di dalamnya konstruksi L1 yang sesuai. Misalnya, `Bucket` konstruksi tingkat tinggi membungkus konstruksi tingkat rendah. `CfnBucket` Karena `CfnBucket` sesuai langsung dengan AWS CloudFormation sumber daya, ia mengekspos semua fitur yang tersedia melalui AWS CloudFormation.

Pendekatan dasar untuk mendapatkan akses ke konstruksi L1 adalah dengan menggunakan (`construct.node.defaultChild`Python:`default_child`), melemparkannya ke tipe yang tepat (jika perlu), dan memodifikasi propertinya. Sekali lagi, mari kita ambil contoh dari a`Bucket`.

TypeScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;

// Change its properties
cfnBucket.analyticsConfiguration = [
  {
    id: 'Config',
    // ...
  }
];
```

JavaScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild;

// Change its properties
cfnBucket.analyticsConfiguration = [
  {
    id: 'Config'
    // ...
  }
```

```
];
```

Python

```
# Get the CloudFormation resource
cfn_bucket = bucket.node.default_child

# Change its properties
cfn_bucket.analytics_configuration = [
    {
        "id": "Config",
        # ...
    }
]
```

Java

```
// Get the CloudFormation resource
CfnBucket cfnBucket = (CfnBucket)bucket.getNode().getDefaultChild();

cfnBucket.setAnalyticsConfigurations(
        Arrays.asList(java.util.Map.of(    // Java 9 or later
            "Id", "Config", // ...
        ));
```

C#

```
// Get the CloudFormation resource
var cfnBucket = (CfnBucket)bucket.Node.DefaultChild;

cfnBucket.AnalyticsConfigurations = new List<object> {
    new Dictionary<string, string>
    {
        ["Id"] = "Config",
        // ...
    }
};
```

Anda juga dapat menggunakan objek ini untuk mengubah AWS CloudFormation opsi seperti `Metadata` dan`UpdatePolicy`.

TypeScript

```
cfnBucket.cfnOptions.metadata = {
  MetadataKey: 'MetadataValue'
};
```

JavaScript

```
cfnBucket.cfnOptions.metadata = {
  MetadataKey: 'MetadataValue'
};
```

Python

```
cfn_bucket.cfn_options.metadata = {
    "MetadataKey": "MetadataValue"
}
```

Java

```
cfnBucket.getCfnOptions().setMetadata(java.util.Map.of(    // Java 9+
    "MetadataKey", "Metadatavalue"));
```

C#

```
cfnBucket.CfnOptions.Metadata = new Dictionary<string, object>
{
    ["MetadataKey"] = "Metadatavalue"
};
```

Palka yang tidak bisa melarikan diri

Ini AWS CDK juga menyediakan kemampuan untuk naik level abstraksi, yang mungkin kita sebut sebagai palka “un-escape”. Jika Anda memiliki konstruksi L1, seperti`CfnBucket`, Anda dapat membuat konstruksi L2 baru (`Bucket`dalam hal ini) untuk membungkus konstruksi L1.

Ini nyaman ketika Anda membuat sumber daya L1 tetapi ingin menggunakannya dengan konstruksi yang membutuhkan sumber daya L2. Ini juga membantu ketika Anda ingin menggunakan metode kenyamanan seperti `.grantXxxx()` itu tidak tersedia pada konstruksi L1.

Anda pindah ke tingkat abstraksi yang lebih tinggi menggunakan metode statis pada kelas L2 yang disebut `.fromCfnXxxxx()` —misalnya, untuk bucket Amazon `Bucket.fromCfnBucket()` S3. Sumber daya L1 adalah satu-satunya parameter.

TypeScript

```
b1 = new s3.CfnBucket(this, "buck09", { ... });
b2 = s3.Bucket.fromCfnBucket(b1);
```

JavaScript

```
b1 = new s3.CfnBucket(this, "buck09", { ...} );
b2 = s3.Bucket.fromCfnBucket(b1);
```

Python

```
b1 = s3.CfnBucket(self, "buck09", ...)
 b2 = s3.from_cfn_bucket(b1)
```

Java

```
CfnBucket b1 = CfnBucket.Builder.create(this, "buck09")
        // ....
                          .build();
IBucket b2 = Bucket.fromCfnBucket(b1);
```

C#

```
var b1 = new CfnBucket(this, "buck09", new CfnBucketProps { ... });
var v2 = Bucket.FromCfnBucket(b1);
```

Konstruksi L2 dibuat dari konstruksi L1 adalah objek proxy yang merujuk ke sumber daya L1, mirip dengan yang dibuat dari nama sumber daya, ARN, atau pencarian. Modifikasi pada konstruksi ini tidak memengaruhi AWS CloudFormation template akhir yang disintesis (karena Anda memiliki sumber daya L1, Anda dapat memodifikasinya sebagai gantinya). Untuk informasi selengkapnya tentang objek proxy, lihatthe section called “Mereferensikan sumber daya di akun Anda AWS”.

Untuk menghindari kebingungan, jangan membuat beberapa konstruksi L2 yang merujuk ke konstruksi L1 yang sama. Misalnya, jika Anda mengekstrak `CfnBucket` dari `Bucket`

menggunakan teknik di bagian sebelumnya, Anda tidak boleh membuat `Bucket` instance kedua `Bucket.fromCfnBucket()` dengan memanggilnya`CfnBucket`. Ini benar-benar berfungsi seperti yang Anda harapkan (hanya satu yang `AWS::S3::Bucket` disintesis) tetapi itu membuat kode Anda lebih sulit untuk dipertahankan.

Penggantian mentah

Jika ada properti yang hilang dari konstruksi L1, Anda dapat melewati semua pengetikan menggunakan penggantian mentah. Ini juga memungkinkan untuk menghapus properti yang disintesis.

Gunakan salah satu `addOverride` metode (Python:`add_override`) metode, seperti yang ditunjukkan pada contoh berikut.

TypeScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;

// Use dot notation to address inside the resource template fragment
cfnBucket.addOverride('Properties.VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addDeletionOverride('Properties.VersioningConfiguration.Status');

// use index (0 here) to address an element of a list
cfnBucket.addOverride('Properties.Tags.0.Value', 'NewValue');
cfnBucket.addDeletionOverride('Properties.Tags.0');

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.addPropertyOverride('VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addPropertyDeletionOverride('VersioningConfiguration.Status');
cfnBucket.addPropertyOverride('Tags.0.Value', 'NewValue');
cfnBucket.addPropertyDeletionOverride('Tags.0');
```

JavaScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild ;

// Use dot notation to address inside the resource template fragment
cfnBucket.addOverride('Properties.VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addDeletionOverride('Properties.VersioningConfiguration.Status');
```

```
// use index (0 here) to address an element of a list
cfnBucket.addOverride('Properties.Tags.0.Value', 'NewValue');
cfnBucket.addDeletionOverride('Properties.Tags.0');

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.addPropertyOverride('VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addPropertyDeletionOverride('VersioningConfiguration.Status');
cfnBucket.addPropertyOverride('Tags.0.Value', 'NewValue');
cfnBucket.addPropertyDeletionOverride('Tags.0');
```

Python

```
# Get the CloudFormation resource
cfn_bucket = bucket.node.default_child

# Use dot notation to address inside the resource template fragment
cfn_bucket.add_override("Properties.VersioningConfiguration.Status", "NewStatus")
cfn_bucket.add_deletion_override("Properties.VersioningConfiguration.Status")

# use index (0 here) to address an element of a list
cfn_bucket.add_override("Properties.Tags.0.Value", "NewValue")
cfn_bucket.add_deletion_override("Properties.Tags.0")

# addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfn_bucket.add_property_override("VersioningConfiguration.Status", "NewStatus")
cfn_bucket.add_property_deletion_override("VersioningConfiguration.Status")
cfn_bucket.add_property_override("Tags.0.Value", "NewValue")
cfn_bucket.add_property_deletion_override("Tags.0")
```

Java

```
// Get the CloudFormation resource
CfnBucket cfnBucket = (CfnBucket)bucket.getNode().getDefaultChild();

// Use dot notation to address inside the resource template fragment
cfnBucket.addOverride("Properties.VersioningConfiguration.Status", "NewStatus");
cfnBucket.addDeletionOverride("Properties.VersioningConfiguration.Status");

// use index (0 here) to address an element of a list
cfnBucket.addOverride("Properties.Tags.0.Value", "NewValue");
```

```
cfnBucket.addDeletionOverride("Properties.Tags.0");

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.addPropertyOverride("VersioningConfiguration.Status", "NewStatus");
cfnBucket.addPropertyDeletionOverride("VersioningConfiguration.Status");
cfnBucket.addPropertyOverride("Tags.0.Value", "NewValue");
cfnBucket.addPropertyDeletionOverride("Tags.0");
```

C#

```
// Get the CloudFormation resource
var cfnBucket = (CfnBucket)bucket.node.defaultChild;

// Use dot notation to address inside the resource template fragment
cfnBucket.AddOverride("Properties.VersioningConfiguration.Status", "NewStatus");
cfnBucket.AddDeletionOverride("Properties.VersioningConfiguration.Status");

// use index (0 here) to address an element of a list
cfnBucket.AddOverride("Properties.Tags.0.Value", "NewValue");
cfnBucket.AddDeletionOverride("Properties.Tags.0");

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.AddPropertyOverride("VersioningConfiguration.Status", "NewStatus");
cfnBucket.AddPropertyDeletionOverride("VersioningConfiguration.Status");
cfnBucket.AddPropertyOverride("Tags.0.Value", "NewValue");
cfnBucket.AddPropertyDeletionOverride("Tags.0");
```

Sumber daya khusus

Jika fitur tidak tersedia melalui AWS CloudFormation, tetapi hanya melalui panggilan API langsung, Anda harus menulis Sumber Daya AWS CloudFormation Kustom untuk membuat panggilan API yang Anda butuhkan. Anda dapat menggunakan AWS CDK untuk menulis sumber daya khusus dan membungkusnya menjadi antarmuka konstruksi biasa. Dari perspektif konsumen konstruksi Anda, pengalaman akan terasa asli.

Membangun sumber daya khusus melibatkan penulisan fungsi Lambda yang merespons peristiwa sumber daya `CREATE``UPDATE`, dan `DELETE` siklus hidup. Jika sumber daya kustom Anda hanya perlu melakukan satu panggilan API, pertimbangkan untuk menggunakan AwsCustomResource. Hal ini memungkinkan untuk melakukan panggilan SDK arbitrer selama penerapan. AWS CloudFormation

Jika tidak, Anda harus menulis fungsi Lambda Anda sendiri untuk melakukan pekerjaan yang perlu Anda selesaikan.

Subjek terlalu luas untuk dibahas sepenuhnya di sini, tetapi tautan berikut akan membantu Anda memulai:

- Sumber Daya Kustom
- Contoh Sumber Daya Kustom
- Untuk contoh yang lebih lengkap, lihat DnsValidatedCertificatekelas di pustaka standar CDK. Ini diimplementasikan sebagai sumber daya khusus.

Dapatkan nilai dari variabel lingkungan

Untuk mendapatkan nilai variabel lingkungan, gunakan kode seperti berikut ini. Kode ini mendapatkan nilai variabel lingkungan`MYBUCKET`.

TypeScript

```
// Sets bucket_name to undefined if environment variable not set
var bucket_name = process.env.MYBUCKET;

// Sets bucket_name to a default if env var doesn't exist
var bucket_name = process.env.MYBUCKET || "DefaultName";
```

JavaScript

```
// Sets bucket_name to undefined if environment variable not set
var bucket_name = process.env.MYBUCKET;

// Sets bucket_name to a default if env var doesn't exist
var bucket_name = process.env.MYBUCKET || "DefaultName";
```

Python

```
import os

# Raises KeyError if environment variable doesn't exist
bucket_name = os.environ["MYBUCKET"]
```

```
# Sets bucket_name to None if environment variable doesn't exist
bucket_name = os.getenv("MYBUCKET")

# Sets bucket_name to a default if env var doesn't exist
bucket_name = os.getenv("MYBUCKET", "DefaultName")
```

Java

```
// Sets bucketName to null if environment variable doesn't exist
String bucketName = System.getenv("MYBUCKET");

// Sets bucketName to a default if env var doesn't exist
String bucketName = System.getenv("MYBUCKET");
if (bucketName == null) bucketName = "DefaultName";
```

C#

```
using System;

// Sets bucket name to null if environment variable doesn't exist
string bucketName = Environment.GetEnvironmentVariable("MYBUCKET");

// Sets bucket_name to a default if env var doesn't exist
string bucketName = Environment.GetEnvironmentVariable("MYBUCKET") ?? "DefaultName";
```

Gunakan AWS CloudFormation nilai

Lihat the section called “Parameter-parameter” untuk informasi tentang menggunakan AWS CloudFormation parameter dengan AWS CDK.

Untuk mendapatkan referensi ke sumber daya dalam AWS CloudFormation template yang ada, lihatthe section called “Impor AWS CloudFormation template”.

Impor AWS CloudFormation template yang ada

Impor sumber daya dari AWS CloudFormation template ke AWS Cloud Development Kit (AWS CDK) aplikasi Anda dengan menggunakan `cloudformation-include.CfnInclude`konstruksi untuk mengonversi sumber daya ke konstruksi L1.

Setelah mengimpor, Anda dapat bekerja dengan resource ini di aplikasi Anda dengan cara yang sama seperti jika mereka awalnya didefinisikan dalam AWS CDK kode. Anda juga dapat menggunakan konstruksi L1 ini dalam konstruksi tingkat yang lebih tinggi AWS CDK . Misalnya, ini memungkinkan Anda menggunakan metode pemberian izin L2 dengan sumber daya yang mereka definisikan.

`cloudformation-include.CfnInclude`Konstruksi pada dasarnya menambahkan pembungkus AWS CDK API ke sumber daya apa pun di template Anda AWS CloudFormation . Gunakan kemampuan ini untuk mengimpor AWS CloudFormation templat AWS CDK yang ada ke bagian sekaligus. Dengan melakukan ini, Anda dapat mengelola sumber daya yang ada menggunakan AWS CDK konstruksi untuk memanfaatkan manfaat abstraksi tingkat yang lebih tinggi. Anda juga dapat menggunakan fitur ini untuk menjual AWS CloudFormation template Anda ke AWS CDK pengembang dengan menyediakan API AWS CDK konstruksi.

> **Note**
>
> AWS CDK v1 juga disertakan `aws-cdk-lib.CfnInclude`, yang sebelumnya digunakan untuk tujuan umum yang sama. Namun, ia tidak memiliki banyak fungsi dari. `cloudformation-include.CfnInclude`

Topik

Mengimpor template AWS CloudFormation

Berikut ini adalah contoh AWS CloudFormation template yang akan kita gunakan untuk memberikan contoh dalam topik ini. Salin dan simpan template `my-template.json` untuk diikuti. Setelah mengerjakan contoh-contoh ini, Anda dapat menjelajahi lebih jauh dengan menggunakan salah satu AWS CloudFormation templat yang sudah Anda gunakan. Anda bisa mendapatkannya dari AWS CloudFormation konsol.

```
{
```

```
  "Resources": {
    "MyBucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

Anda dapat bekerja dengan template JSON atau YAMG. Kami merekomendasikan JSON jika tersedia karena parser YAMG dapat sedikit berbeda dalam apa yang mereka terima.

Berikut ini adalah contoh cara mengimpor template sampel ke AWS CDK aplikasi Anda menggunakan`cloudformation-include`. Template diimpor dalam konteks tumpukan CDK.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as cfninc from 'aws-cdk-lib/cloudformation-include';
import { Construct } from 'constructs';

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

    const template = new cfninc.CfnInclude(this, 'Template', {
      templateFile: 'my-template.json',
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const cfninc = require('aws-cdk-lib/cloudformation-include');

class MyStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const template = new cfninc.CfnInclude(this, 'Template', {
```

```
      templateFile: 'my-template.json',
    });
  }
}

module.exports = { MyStack }
```

Python

```
import aws_cdk as cdk
from aws_cdk import cloudformation_include as cfn_inc
from constructs import Construct

class MyStack(cdk.Stack):

    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        template = cfn_inc.CfnInclude(self, "Template",
            template_file="my-template.json")
```

Java

```
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.cloudformation.include.CfnInclude;
import software.constructs.Construct;

public class MyStack extends Stack {
    public MyStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyStack(final Construct scope, final String id, final StackProps props) {
        super(scope, id, props);

        CfnInclude template = CfnInclude.Builder.create(this, "Template")
         .templateFile("my-template.json")
         .build();
    }
}
```

C#

```
using Amazon.CDK;
using Constructs;
using cfnInc = Amazon.CDK.CloudFormation.Include;

namespace MyApp
{
    public class MyStack : Stack
    {
        internal MyStack(Construct scope, string id, IStackProps props = null) :
 base(scope, id, props)
        {
            var template = new cfnInc.CfnInclude(this, "Template", new
 cfnInc.CfnIncludeProps
            {
                TemplateFile = "my-template.json"
            });
        }
    }
}
```

Secara default, mengimpor sumber daya mempertahankan ID logis asli sumber daya dari template. Perilaku ini cocok untuk mengimpor AWS CloudFormation template ke dalam AWS CDK, di mana ID logis harus dipertahankan. AWS CloudFormation membutuhkan informasi ini untuk mengenali sumber daya yang diimpor ini sebagai sumber daya yang sama dari AWS CloudFormation template.

Jika Anda mengembangkan pembungkus AWS CDK konstruksi untuk template sehingga dapat digunakan oleh AWS CDK pengembang lain, mintalah ID sumber daya AWS CDK baru sebagai gantinya. Dengan melakukan ini, konstruksi dapat digunakan beberapa kali dalam tumpukan tanpa konflik nama. Untuk melakukan ini, setel `preserveLogicalIds` properti ke `false` saat mengimpor template. Berikut ini adalah contohnya:

TypeScript

```
const template = new cfninc.CfnInclude(this, 'MyConstruct', {
  templateFile: 'my-template.json',
  preserveLogicalIds: false
});
```

JavaScript

```
const template = new cfninc.CfnInclude(this, 'MyConstruct', {
  templateFile: 'my-template.json',
  preserveLogicalIds: false
});
```

Python

```
template = cfn_inc.CfnInclude(self, "Template",
    template_file="my-template.json",
    preserve_logical_ids=False)
```

Java

```
CfnInclude template = CfnInclude.Builder.create(this, "Template")
 .templateFile("my-template.json")
 .preserveLogicalIds(false)
 .build();
```

C#

```
var template = new cfnInc.CfnInclude(this, "Template", new cfn_inc.CfnIncludeProps
{
    TemplateFile = "my-template.json",
    PreserveLogicalIds = false
});
```

Untuk menempatkan sumber daya yang diimpor di bawah kendali AWS CDK aplikasi Anda, tambahkan tumpukan keApp:

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { MyStack } from '../lib/my-stack';

const app = new cdk.App();
new MyStack(app, 'MyStack');
```

JavaScript

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

const app = new cdk.App();
new MyStack(app, 'MyStack');
```

Python

```
import aws_cdk as cdk
from mystack.my_stack import MyStack

app = cdk.App()
MyStack(app, "MyStack")
```

Java

```
import software.amazon.awscdk.App;

public class MyApp {
    public static void main(final String[] args) {
        App app = new App();

        new MyStack(app, "MyStack");
    }
}
```

C#

```
using Amazon.CDK;

namespace CdkApp
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new MyStack(app, "MyStack");
        }
    }
```

```
}
```

Untuk memverifikasi bahwa tidak akan ada perubahan yang tidak diinginkan pada AWS sumber daya di tumpukan, Anda dapat melakukan diff. Gunakan AWS CDK CLI `cdk diff` perintah dan hilangkan metadata AWS CDK-spesifik apa pun. Berikut ini adalah contohnya:

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

Setelah Anda mengimpor AWS CloudFormation template, AWS CDK aplikasi harus menjadi sumber kebenaran untuk sumber daya impor Anda. Untuk membuat perubahan pada resource Anda, modifikasi di AWS CDK aplikasi Anda dan terapkan dengan AWS CDK CLI cdk deploy perintah.

Mengakses sumber daya yang diimpor

Nama `template` dalam kode contoh mewakili AWS CloudFormation template yang diimpor. Untuk mengakses sumber daya darinya, gunakan `getResource()`metode objek. Untuk mengakses sumber daya yang dikembalikan sebagai jenis sumber daya tertentu, lemparkan hasilnya ke jenis yang diinginkan. Ini tidak diperlukan dalam Python atau. JavaScript Berikut ini adalah contohnya:

TypeScript

```
const cfnBucket = template.getResource('MyBucket') as s3.CfnBucket;
```

JavaScript

```
const cfnBucket = template.getResource('MyBucket');
```

Python

```
cfn_bucket = template.get_resource("MyBucket")
```

Java

```
CfnBucket cfnBucket = (CfnBucket)template.getResource("MyBucket");
```

C#

```
var cfnBucket = (CfnBucket)template.GetResource("MyBucket");
```

Dari contoh ini, sekarang `cfnBucket` merupakan contoh dari `aws-s3.CfnBucket`kelas. Ini adalah konstruksi L1 yang mewakili sumber daya yang sesuai AWS CloudFormation . Anda dapat memperlakukannya seperti sumber daya lain dari jenisnya. Misalnya, Anda bisa mendapatkan nilai ARN dengan properti. `bucket.attrArn`

Untuk membungkus `CfnBucket` sumber daya L1 dalam `aws-s3.Bucket`instance L2 sebagai gantinya, gunakan metode statis `fromBucketArn()`, `fromBucketAttributes()`, atau. `fromBucketName()` Biasanya, `fromBucketName()` metode ini paling nyaman. Berikut ini adalah contohnya:

TypeScript

```
const bucket = s3.Bucket.fromBucketName(this, 'Bucket', cfnBucket.ref);
```

JavaScript

```
const bucket = s3.Bucket.fromBucketName(this, 'Bucket', cfnBucket.ref);
```

Python

```
bucket = s3.Bucket.from_bucket_name(self, "Bucket", cfn_bucket.ref)
```

Java

```
Bucket bucket = (Bucket)Bucket.fromBucketName(this, "Bucket", cfnBucket.getRef());
```

C#

```
var bucket = (Bucket)Bucket.FromBucketName(this, "Bucket", cfnBucket.Ref);
```

Konstruksi L2 lainnya memiliki metode serupa untuk membuat konstruksi dari sumber daya yang ada.

Saat Anda membungkus konstruksi L1 dalam konstruksi L2, itu tidak membuat sumber daya baru. Dari contoh kami, kami tidak membuat ember S3; kedua. Sebaliknya, `Bucket` instance baru merangkum yang sudah ada. `CfnBucket`

Dari contoh, sekarang `bucket` merupakan konstruksi L2 yang berperilaku seperti `Bucket` konstruksi L2 lainnya. Misalnya, Anda dapat memberikan akses tulis AWS Lambda fungsi ke bucket dengan menggunakan `grantWrite()`metode praktis bucket. Anda tidak perlu menentukan kebijakan

yang diperlukan AWS Identity and Access Management (IAM) secara manual. Berikut ini adalah contohnya:

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

Mengganti parameter

Jika AWS CloudFormation template berisi parameter, Anda dapat menggantinya dengan nilai waktu pembuatan saat diimpor menggunakan `parameters` properti. Dalam contoh berikut, kita mengganti `UploadBucket` parameter dengan ARN dari bucket yang didefinisikan di tempat lain dalam kode kita AWS CDK .

TypeScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

JavaScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

Python

```
template = cfn_inc.CfnInclude(self, "Template",
    template_file="my-template.json",
    parameters=dict(UploadBucket=bucket.bucket_arn)
)
```

Java

```
CfnInclude template = CfnInclude.Builder.create(this, "Template")
 .templateFile("my-template.json")
 .parameters(java.util.Map.of(    // Map.of requires Java 9+
   "UploadBucket", bucket.getBucketArn()))
 .build();
```

C#

```
var template = new cfnInc.CfnInclude(this, "Template", new cfnInc.CfnIncludeProps
{
    TemplateFile = "my-template.json",
    Parameters = new Dictionary<string, string>
    {
        { "UploadBucket", bucket.BucketArn }
    }
});
```

Elemen template lainnya

Anda dapat mengimpor elemen AWS CloudFormation template apa pun, bukan hanya sumber daya. Elemen yang diimpor menjadi bagian dari AWS CDK tumpukan. Untuk mengimpor elemen-elemen ini, gunakan metode `CfnInclude` objek berikut:

- `getCondition()`— AWS CloudFormation kondisi.
- `getHook()`— AWS CloudFormation kait untuk penyebaran biru/hijau.
- `getMapping()`— AWS CloudFormation pemetaan.
- `getOutput()`— AWS CloudFormation output.
- `getParameter()`— AWS CloudFormation parameter.
- `getRule()`— AWS CloudFormation aturan untuk AWS Service Catalog template.

Masing-masing metode ini mengembalikan sebuah instance dari kelas yang mewakili jenis tertentu dari AWS CloudFormation elemen. Benda-benda ini bisa berubah. Perubahan yang Anda buat pada mereka akan muncul di template yang dihasilkan dari AWS CDK tumpukan. Berikut ini adalah contoh yang mengimpor parameter dari template dan memodifikasi nilai defaultnya:

TypeScript

```
const param = template.getParameter('MyParameter');
param.default = "AWS CDK"
```

JavaScript

```
const param = template.getParameter('MyParameter');
param.default = "AWS CDK"
```

Python

```
param = template.get_parameter("MyParameter")
param.default = "AWS CDK"
```

Java

```
CfnParameter param = template.getParameter("MyParameter");
param.setDefaultValue("AWS CDK")
```

C#

```
var cfnBucket = (CfnBucket)template.GetResource("MyBucket");
var param = template.GetParameter("MyParameter");
param.Default = "AWS CDK";
```

Tumpukan nested

Anda dapat mengimpor tumpukan bersarang dengan menentukan mereka baik ketika Anda mengimpor template utama mereka, atau di beberapa titik kemudian. Template bersarang harus disimpan dalam file lokal, tetapi direferensikan sebagai `NestedStack` sumber daya di template utama. Juga, nama sumber daya yang digunakan dalam AWS CDK kode harus cocok dengan nama yang digunakan untuk tumpukan bersarang di template utama.

Mengingat definisi sumber daya ini di template utama, kode berikut menunjukkan cara mengimpor tumpukan bersarang yang direferensikan dengan dua arah.

```
"NestedStack": {
  "Type": "AWS::CloudFormation::Stack",
  "Properties": {
    "TemplateURL": "https://my-s3-template-source.s3.amazonaws.com/nested-stack.json"
  }
```

TypeScript

```
// include nested stack when importing main stack
const mainTemplate = new cfninc.CfnInclude(this, 'MainStack', {
  templateFile: 'main-template.json',
  loadNestedStacks: {
    'NestedStack': {
      templateFile: 'nested-template.json',
    },
  },
});

// or add it some time after importing the main stack
const nestedTemplate = mainTemplate.loadNestedStack('NestedTemplate', {
  templateFile: 'nested-template.json',
});
```

JavaScript

```
// include nested stack when importing main stack
const mainTemplate = new cfninc.CfnInclude(this, 'MainStack', {
  templateFile: 'main-template.json',
  loadNestedStacks: {
    'NestedStack': {
```

```
      templateFile: 'nested-template.json',
    },
  },
});

// or add it some time after importing the main stack
const nestedTemplate = mainTemplate.loadNestedStack('NestedStack', {
  templateFile: 'my-nested-template.json',
});
```

Python

```
# include nested stack when importing main stack
main_template = cfn_inc.CfnInclude(self, "MainStack",
    template_file="main-template.json",
    load_nested_stacks=dict(NestedStack=
        cfn_inc.CfnIncludeProps(template_file="nested-template.json")))

# or add it some time after importing the main stack
nested_template = main_template.load_nested_stack("NestedStack",
    template_file="nested-template.json")
```

Java

```
CfnInclude mainTemplate = CfnInclude.Builder.create(this, "MainStack")
 .templateFile("main-template.json")
 .loadNestedStacks(java.util.Map.of(    // Map.of requires Java 9+
    "NestedStack", CfnIncludeProps.builder()
    .templateFile("nested-template.json").build()))
 .build();

// or add it some time after importing the main stack
IncludedNestedStack nestedTemplate = mainTemplate.loadNestedStack("NestedTemplate",
 CfnIncludeProps.builder()
  .templateFile("nested-template.json")
  .build());
```

C#

```
// include nested stack when importing main stack
var mainTemplate = new cfnInc.CfnInclude(this, "MainStack", new
 cfnInc.CfnIncludeProps
```

```
{
    TemplateFile = "main-template.json",
    LoadNestedStacks = new Dictionary<string, cfnInc.ICfnIncludeProps>
    {
        { "NestedStack", new cfnInc.CfnIncludeProps { TemplateFile = "nested-
template.json" } }
    }
});

// or add it some time after importing the main stack
var nestedTemplate = mainTemplate.LoadNestedStack("NestedTemplate", new
 cfnInc.CfnIncludeProps {
    TemplateFile = 'nested-template.json'
});
```

Anda dapat mengimpor beberapa tumpukan bersarang dengan salah satu metode. Saat mengimpor template utama, Anda memberikan pemetaan antara nama sumber daya dari setiap tumpukan bersarang dan file templatnya. Pemetaan ini dapat berisi sejumlah entri. Untuk melakukannya setelah impor awal, panggil `loadNestedStack()` sekali untuk setiap tumpukan bersarang.

Setelah mengimpor tumpukan bersarang, Anda dapat mengaksesnya menggunakan metode template utama. `getNestedStack()`

TypeScript

```
const nestedStack = mainTemplate.getNestedStack('NestedStack').stack;
```

JavaScript

```
const nestedStack = mainTemplate.getNestedStack('NestedStack').stack;
```

Python

```
nested_stack = main_template.get_nested_stack("NestedStack").stack
```

Java

```
NestedStack nestedStack = mainTemplate.getNestedStack("NestedStack").getStack();
```

C#

```
var nestedStack = mainTemplate.GetNestedStack("NestedStack").Stack;
```

`getNestedStack()`Metode mengembalikan sebuah `IncludedNestedStack`instance. Dari contoh ini, Anda dapat mengakses AWS CDK `NestedStack`instance melalui `stack` properti, seperti yang ditunjukkan pada contoh. Anda juga dapat mengakses objek AWS CloudFormation template asli melalui`includedTemplate`, dari mana Anda dapat memuat sumber daya dan AWS CloudFormation elemen lainnya.

Dapatkan nilai dari Systems Manager Parameter Store

AWS Cloud Development Kit (AWS CDK) Dapat mengambil nilai atribut AWS Systems Manager Parameter Store. Selama sintesis, AWS CDK menghasilkan token yang diselesaikan oleh AWS CloudFormation selama penerapan.

AWS CDK Dukungan mengambil nilai polos dan aman. Anda dapat meminta versi tertentu dari kedua jenis nilai tersebut. Untuk nilai biasa, Anda dapat menghilangkan versi dari permintaan Anda untuk mengambil versi terbaru. Untuk nilai aman, Anda harus menentukan versi saat meminta nilai atribut aman.

> **Note**
>
> Topik ini menunjukkan cara membaca atribut dari AWS Systems Manager Parameter Store. Anda juga dapat membaca rahasia dari AWS Secrets Manager (lihatMendapatkan nilai dari AWS Secrets Manager).

Topik

- Baca nilai Systems Manager pada waktu penerapan
- Baca nilai Systems Manager pada waktu sintesis
- Menulis nilai ke Systems Manager

Baca nilai Systems Manager pada waktu penerapan

Untuk membaca nilai dari Systems Manager Parameter Store, gunakan valueForStringParameter dan valueForSecureStringParametermetode. Pilih metode berdasarkan apakah atribut yang Anda inginkan adalah string biasa atau nilai string aman. Metode ini mengembalikan token, bukan nilai sebenarnya. Nilai diselesaikan oleh AWS CloudFormation selama penerapan. Berikut ini adalah contohnya:

TypeScript

```
import * as ssm from 'aws-cdk-lib/aws-ssm';

// Get latest version or specified version of plain string attribute
const latestStringToken = ssm.StringParameter.valueForStringParameter(
    this, 'my-plain-parameter-name');      // latest version
const versionOfStringToken = ssm.StringParameter.valueForStringParameter(
    this, 'my-plain-parameter-name', 1);   // version 1

// Get specified version of secure string attribute
const secureStringToken = ssm.StringParameter.valueForSecureStringParameter(
    this, 'my-secure-parameter-name', 1);   // must specify version
```

JavaScript

```
const ssm = require('aws-cdk-lib/aws-ssm');

// Get latest version or specified version of plain string attribute
const latestStringToken = ssm.StringParameter.valueForStringParameter(
    this, 'my-plain-parameter-name');      // latest version
const versionOfStringToken = ssm.StringParameter.valueForStringParameter(
    this, 'my-plain-parameter-name', 1);   // version 1

// Get specified version of secure string attribute
const secureStringToken = ssm.StringParameter.valueForSecureStringParameter(
    this, 'my-secure-parameter-name', 1);   // must specify version
```

Python

```
import aws_cdk.aws_ssm as ssm

# Get latest version or specified version of plain string attribute
```

```
latest_string_token = ssm.StringParameter.value_for_string_parameter(
    self, "my-plain-parameter-name")
latest_string_token = ssm.StringParameter.value_for_string_parameter(
    self, "my-plain-parameter-name", 1)

# Get specified version of secure string attribute
secure_string_token = ssm.StringParameter.value_for_secure_string_parameter(
    self, "my-secure-parameter-name", 1)   # must specify version
```

Java

```
import software.amazon.awscdk.services.ssm.StringParameter;

//Get latest version or specified version of plain string attribute
String latestStringToken = StringParameter.valueForStringParameter(
            this, "my-plain-parameter-name");       // latest version
String versionOfStringToken = StringParameter.valueForStringParameter(
            this, "my-plain-parameter-name", 1);    // version 1

//Get specified version of secure string attribute
String secureStringToken = StringParameter.valueForSecureStringParameter(
            this, "my-secure-parameter-name", 1);   // must specify version
```

C#

```
using Amazon.CDK.AWS.SSM;

// Get latest version or specified version of plain string attribute
var latestStringToken = StringParameter.ValueForStringParameter(
    this, "my-plain-parameter-name");      // latest version
var versionOfStringToken = StringParameter.ValueForStringParameter(
    this, "my-plain-parameter-name", 1);   // version 1

// Get specified version of secure string attribute
var secureStringToken = StringParameter.ValueForSecureStringParameter(
    this, "my-secure-parameter-name", 1);   // must specify version
```

Sejumlah AWS layanan terbatas saat ini mendukung fitur ini.

Baca nilai Systems Manager pada waktu sintesis

Kadang-kadang, berguna untuk memberikan parameter pada waktu sintesis. Dengan melakukan ini, AWS CloudFormation template akan selalu menggunakan nilai yang sama alih-alih menyelesaikan nilai selama penerapan.

Untuk membaca nilai dari Systems Manager Parameter Store pada waktu sintesis, gunakan valueFromLookupmetode (Python:`value_from_lookup`). Metode ini mengembalikan nilai aktual dari parameter sebagai the section called “Konteks” nilai. Jika nilai belum di-cache `cdk.json` atau diteruskan pada baris perintah, itu diambil dari akun saat ini AWS . Untuk alasan ini, tumpukan harus disintesis dengan informasi AWS lingkungan eksplisit.

Berikut ini adalah contohnya:

TypeScript

```
import * as ssm from 'aws-cdk-lib/aws-ssm';

const stringValue = ssm.StringParameter.valueFromLookup(this, 'my-plain-parameter-
name');
```

JavaScript

```
const ssm = require('aws-cdk-lib/aws-ssm');

const stringValue = ssm.StringParameter.valueFromLookup(this, 'my-plain-parameter-
name');
```

Python

```
import aws_cdk.aws_ssm as ssm

string_value = ssm.StringParameter.value_from_lookup(self, "my-plain-parameter-
name")
```

Java

```
import software.amazon.awscdk.services.ssm.StringParameter;

String stringValue = StringParameter.valueFromLookup(this, "my-plain-parameter-
name");
```

C#

```
using Amazon.CDK.AWS.SSM;

var stringValue = StringParameter.ValueFromLookup(this, "my-plain-parameter-name");
```

Hanya string Systems Manager biasa yang dapat diambil. String aman tidak dapat diambil. Versi terbaru akan selalu dikembalikan. Versi tertentu tidak dapat diminta.

> **Important**
>
> Nilai yang diambil akan berakhir di template yang disintesis AWS CloudFormation Anda. Ini mungkin risiko keamanan, tergantung pada siapa yang memiliki akses ke AWS CloudFormation template Anda dan nilai apa itu. Umumnya, jangan gunakan fitur ini untuk kata sandi, kunci, atau nilai lain yang ingin Anda jaga kerahasiaannya.

Menulis nilai ke Systems Manager

Anda dapat menggunakan AWS CLI, SDK AWS Management Console, atau AWS SDK untuk menetapkan nilai parameter Systems Manager. Contoh berikut menggunakan perintah CLI put-parameter ssm.

```
aws ssm put-parameter --name "parameter-name" --type "String" --value "parameter-value"
aws ssm put-parameter --name "secure-parameter-name" --type "SecureString" --value
 "secure-parameter-value"
```

Saat memperbarui nilai SSM yang sudah ada, sertakan juga `--overwrite` opsi.

```
aws ssm put-parameter --overwrite --name "parameter-name" --type "String" --value
 "parameter-value"
aws ssm put-parameter --overwrite --name "secure-parameter-name" --type "SecureString"
 --value "secure-parameter-value"
```

Mendapatkan nilai dari AWS Secrets Manager

Untuk menggunakan nilai dari AWS Secrets Manager AWS CDK aplikasi Anda, gunakan metode fromSecretAttributes(). Ini merupakan nilai yang diambil dari Secrets Manager dan digunakan pada waktu AWS CloudFormation penyebaran. Berikut ini adalah contohnya:

TypeScript

```
import * as sm from "aws-cdk-lib/aws-secretsmanager";

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

    const secret = sm.Secret.fromSecretAttributes(this, "ImportedSecret", {
      secretCompleteArn:
        "arn:aws:secretsmanager:<region>:<account-id-number>:secret:<secret-name>-
<random-6-characters>"
      // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
      // encryptionKey: ...
    });
```

JavaScript

```
const sm = require("aws-cdk-lib/aws-secretsmanager");

class SecretsManagerStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const secret = sm.Secret.fromSecretAttributes(this, "ImportedSecret", {
      secretCompleteArn:
        "arn:aws:secretsmanager:<region>:<account-id-number>:secret:<secret-name>-
<random-6-characters>"
      // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
      // encryptionKey: ...
    });
  }
}
```

```
module.exports = { SecretsManagerStack }
```

Python

```
import aws_cdk.aws_secretsmanager as sm

class SecretsManagerStack(cdk.Stack):
    def __init__(self, scope: cdk.App, id: str, **kwargs):
      super().__init__(scope, name, **kwargs)

      secret = sm.Secret.from_secret_attributes(self, "ImportedSecret",
          secret_complete_arn="arn:aws:secretsmanager:<region>:<account-id-
number>:secret:<secret-name>-<random-6-characters>",
          # If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
          # encryption_key=....
      )
```

Java

```
import software.amazon.awscdk.services.secretsmanager.Secret;
import software.amazon.awscdk.services.secretsmanager.SecretAttributes;

public class SecretsManagerStack extends Stack {
    public SecretsManagerStack(App scope, String id) {
        this(scope, id, null);
    }

    public SecretsManagerStack(App scope, String id, StackProps props) {
        super(scope, id, props);

        Secret secret = (Secret)Secret.fromSecretAttributes(this, "ImportedSecret",
 SecretAttributes.builder()
            .secretCompleteArn("arn:aws:secretsmanager:<region>:<account-id-
number>:secret:<secret-name>-<random-6-characters>")
             // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
             // .encryptionKey(...)
             .build());
    }
}
```

C#

```
using Amazon.CDK.AWS.SecretsManager;

public class SecretsManagerStack : Stack
{
    public SecretsManagerStack(App scope, string id, StackProps props) : base(scope,
 id, props) {

        var secret = Secret.FromSecretAttributes(this, "ImportedSecret", new
 SecretAttributes {
            SecretCompleteArn = "arn:aws:secretsmanager:<region>:<account-id-
number>:secret:<secret-name>-<random-6-characters>"
            // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
            // encryptionKey = ...,
        });
    }
```

Tip

Gunakan perintah AWS CLI create-secret CLI untuk membuat rahasia dari baris perintah, seperti saat menguji:

```
aws secretsmanager create-secret --name ImportedSecret --secret-string
 mygroovybucket
```

Perintah mengembalikan ARN yang dapat Anda gunakan dengan contoh sebelumnya.

Setelah Anda membuat `Secret` instance, Anda bisa mendapatkan nilai rahasia dari `secretValue` atribut instance. Nilai diwakili oleh sebuah `SecretValue` instance, tipe khusus darithe section called "Token". Karena ini adalah token, itu memiliki makna hanya setelah resolusi. Aplikasi CDK Anda tidak perlu mengakses nilai sebenarnya. Sebagai gantinya, aplikasi dapat meneruskan `SecretValue` instance (atau string atau representasi numeriknya) ke metode CDK apa pun yang membutuhkan nilainya.

Atur CloudWatch alarm

Gunakan paket aws-cloudwatch untuk menyiapkan alarm Amazon CloudWatch pada metrik. CloudWatch Anda dapat menggunakan metrik yang telah ditentukan atau membuat metrik Anda sendiri.

Topik

Menggunakan metrik yang ada

Banyak modul AWS Construct Library memungkinkan Anda menyetel alarm pada metrik yang ada dengan meneruskan nama metrik ke metode kenyamanan pada instance objek yang memiliki metrik. Misalnya, dengan antrean Amazon SQS, Anda bisa mendapatkan metrik ApproximateNumberOfMessagesVisibledari metode metrik () antrian:

TypeScript

```
const metric = queue.metric("ApproximateNumberOfMessagesVisible");
```

JavaScript

```
const metric = queue.metric("ApproximateNumberOfMessagesVisible");
```

Python

```
metric = queue.metric("ApproximateNumberOfMessagesVisible")
```

Java

```
Metric metric = queue.metric("ApproximateNumberOfMessagesVisible");
```

C#

```
var metric = queue.Metric("ApproximateNumberOfMessagesVisible");
```

Membuat metrik Anda sendiri

Buat metrik Anda sendiri sebagai berikut, di mana nilai namespace harus seperti AWS/SQS untuk antrian Amazon SQS. Anda juga perlu menentukan nama dan dimensi metrik Anda:

TypeScript

```
const metric = new cloudwatch.Metric({
  namespace: 'MyNamespace',
  metricName: 'MyMetric',
  dimensionsMap: { MyDimension: 'MyDimensionValue' }
});
```

JavaScript

```
const metric = new cloudwatch.Metric({
  namespace: 'MyNamespace',
  metricName: 'MyMetric',
  dimensionsMap: { MyDimension: 'MyDimensionValue' }
});
```

Python

```
metric = cloudwatch.Metric(
    namespace="MyNamespace",
    metric_name="MyMetric",
    dimensionsMap=dict(MyDimension="MyDimensionValue")
)
```

Java

```
Metric metric = Metric.Builder.create()
        .namespace("MyNamespace")
        .metricName("MyMetric")
        .dimensionsMap(java.util.Map.of(    // Java 9 or later
            "MyDimension", "MyDimensionValue"))
        .build();
```

C#

```
var metric = new Metric(this, "Metric", new MetricProps
```

```
{
    Namespace = "MyNamespace",
    MetricName = "MyMetric",
    Dimensions = new Dictionary<string, object>
    {
        { "MyDimension", "MyDimensionValue" }
    }
});
```

Membuat alarm

Setelah Anda memiliki metrik, baik yang sudah ada atau yang Anda tentukan, Anda dapat membuat alarm. Dalam contoh ini, alarm dinaikkan ketika ada lebih dari 100 metrik Anda dalam dua dari tiga periode evaluasi terakhir. Anda dapat menggunakan perbandingan seperti kurang dari pada alarm Anda melalui properti. `comparisonOperator` G reater-than-or-equal -to adalah AWS CDK default, jadi kita tidak perlu menentukannya.

TypeScript

```
const alarm = new cloudwatch.Alarm(this, 'Alarm', {
  metric: metric,
  threshold: 100,
  evaluationPeriods: 3,
  datapointsToAlarm: 2,
});
```

JavaScript

```
const alarm = new cloudwatch.Alarm(this, 'Alarm', {
  metric: metric,
  threshold: 100,
  evaluationPeriods: 3,
  datapointsToAlarm: 2
});
```

Python

```
alarm = cloudwatch.Alarm(self, "Alarm",
    metric=metric,
    threshold=100,
```

```
    evaluation_periods=3,
    datapoints_to_alarm=2
)
```

Java

```
import software.amazon.awscdk.services.cloudwatch.Alarm;
import software.amazon.awscdk.services.cloudwatch.Metric;

Alarm alarm = Alarm.Builder.create(this, "Alarm")
        .metric(metric)
        .threshold(100)
        .evaluationPeriods(3)
        .datapointsToAlarm(2).build();
```

C#

```
var alarm = new Alarm(this, "Alarm", new AlarmProps
{
    Metric = metric,
    Threshold = 100,
    EvaluationPeriods = 3,
    DatapointsToAlarm = 2
});
```

Cara alternatif untuk membuat alarm menggunakan metode createAlarm () metrik, yang pada dasarnya mengambil properti yang sama dengan konstruktor. `Alarm` Anda tidak perlu memasukkan metrik, karena sudah diketahui.

TypeScript

```
metric.createAlarm(this, 'Alarm', {
  threshold: 100,
  evaluationPeriods: 3,
  datapointsToAlarm: 2,
});
```

JavaScript

```
metric.createAlarm(this, 'Alarm', {
```

```
  threshold: 100,
  evaluationPeriods: 3,
  datapointsToAlarm: 2,
});
```

Python

```
metric.create_alarm(self, "Alarm",
    threshold=100,
    evaluation_periods=3,
    datapoints_to_alarm=2
)
```

Java

```
metric.createAlarm(this, "Alarm", new CreateAlarmOptions.Builder()
        .threshold(100)
        .evaluationPeriods(3)
        .datapointsToAlarm(2)
        .build());
```

C#

```
metric.CreateAlarm(this, "Alarm", new CreateAlarmOptions
{
    Threshold = 100,
    EvaluationPeriods = 3,
    DatapointsToAlarm = 2
});
```

Simpan dan ambil nilai variabel konteks

Anda dapat menentukan variabel konteks dengan AWS Cloud Development Kit (AWS CDK) CLI atau dalam `cdk.json` file. Kemudian, gunakan `TryGetContext` metode untuk mengambil nilai.

Topik

- Tentukan variabel konteks
- Ambil nilai variabel konteks

Tentukan variabel konteks

Anda dapat menentukan variabel konteks baik sebagai bagian dari AWS CDK CLI perintah, atau dalam`cdk.json`.

Untuk membuat variabel konteks baris perintah, gunakan opsi --context (-c), seperti yang ditunjukkan pada contoh berikut.

```
cdk synth -c bucket_name=mygroovybucket
```

Untuk menentukan variabel konteks dan nilai yang sama dalam `cdk.json` file, gunakan kode berikut.

```
{
  "context": {
    "bucket_name": "myotherbucket"
  }
}
```

Jika Anda menentukan variabel konteks menggunakan `cdk.json` file AWS CDK CLI dan, AWS CDK CLI nilainya diutamakan.

Ambil nilai variabel konteks

Untuk mendapatkan nilai variabel konteks di aplikasi Anda, gunakan `TryGetContext` metode dalam konteks konstruksi. (Yaitu, ketika`this`, atau `self` dengan Python, adalah contoh dari beberapa konstruksi.)

Dalam contoh ini, kita mengambil nilai dari variabel `bucket_name` konteks. Jika nilai yang diminta tidak ditentukan, `TryGetContext` mengembalikan `undefined` (`None`dalam Python; `null` di Java dan C #; `nil` di Go) daripada menaikkan pengecualian.

TypeScript

```
const bucket_name = this.node.tryGetContext('bucket_name');
```

JavaScript

```
const bucket_name = this.node.tryGetContext('bucket_name');
```

Python

```
bucket_name = self.node.try_get_context("bucket_name")
```

Java

```
String bucketName = (String)this.getNode().tryGetContext("bucket_name");
```

C#

```
var bucketName = this.Node.TryGetContext("bucket_name");
```

Di luar konteks konstruksi, Anda dapat mengakses variabel konteks dari objek aplikasi, seperti ini.

TypeScript

```
const app = new cdk.App();
const bucket_name = app.node.tryGetContext('bucket_name')
```

JavaScript

```
const app = new cdk.App();
const bucket_name = app.node.tryGetContext('bucket_name');
```

Python

```
app = cdk.App()
bucket_name = app.node.try_get_context("bucket_name")
```

Java

```
App app = App();
String bucketName = (String)app.getNode().tryGetContext("bucket_name");
```

C#

```
app = App();
var bucketName = app.Node.TryGetContext("bucket_name");
```

Untuk detail lebih lanjut tentang bekerja dengan variabel konteks, lihatthe section called “Konteks”.

Menggunakan sumber daya dari AWS CloudFormation Public Registry

AWS CloudFormation Public Registry memungkinkan Anda mengelola ekstensi, baik publik maupun pribadi, seperti sumber daya, modul, dan kait yang tersedia untuk digunakan di situs Anda Akun AWS. Anda dapat menggunakan ekstensi sumber daya publik dalam AWS Cloud Development Kit (AWS CDK) aplikasi Anda dengan `CfnResource`konstruksinya.

Untuk mempelajari lebih lanjut tentang Registri AWS CloudFormation Publik, lihat Menggunakan AWS CloudFormation registri di Panduan AWS CloudFormation Pengguna.

Semua ekstensi publik yang diterbitkan oleh AWS tersedia untuk semua akun di semua Wilayah tanpa tindakan apa pun dari Anda. Namun, Anda harus mengaktifkan setiap ekstensi pihak ketiga yang ingin Anda gunakan, di setiap akun dan Wilayah tempat Anda ingin menggunakannya.

> **Note**
>
> Ketika Anda menggunakan AWS CloudFormation dengan jenis sumber daya pihak ketiga, Anda akan dikenakan biaya. Biaya didasarkan pada jumlah operasi handler yang Anda jalankan per bulan dan durasi operasi handler. Lihat CloudFormation harga untuk detail lengkapnya.

Untuk mempelajari lebih lanjut tentang ekstensi publik, lihat Menggunakan ekstensi publik CloudFormation di Panduan AWS CloudFormation Pengguna

Topik

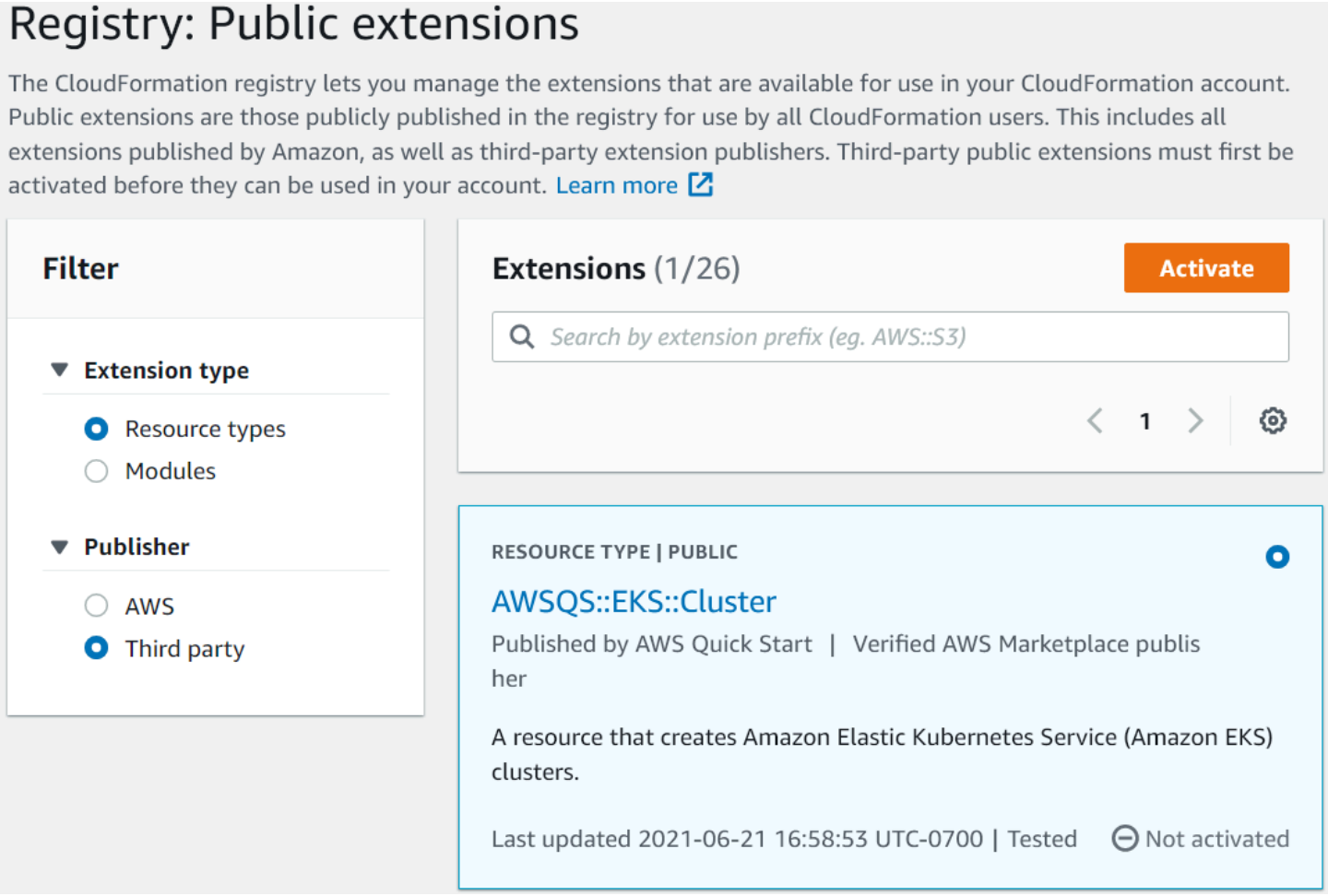
- Mengaktifkan sumber daya pihak ketiga di akun dan Wilayah Anda
- Menambahkan sumber daya dari AWS CloudFormation Public Registry ke aplikasi CDK

Mengaktifkan sumber daya pihak ketiga di akun dan Wilayah Anda

Ekstensi yang diterbitkan oleh AWS tidak memerlukan aktivasi. Mereka selalu tersedia di setiap akun dan Wilayah. Anda dapat mengaktifkan ekstensi pihak ketiga melalui AWS Management Console,

melalui AWS Command Line Interface, atau dengan menggunakan AWS CloudFormation sumber daya khusus.

Untuk mengaktifkan ekstensi pihak ketiga melalui AWS Management Console atau melihat sumber daya apa yang tersedia



1. Masuk ke AWS akun tempat Anda ingin menggunakan ekstensi, lalu beralih ke Wilayah tempat Anda ingin menggunakannya.
2. Arahkan ke CloudFormation konsol melalui menu Layanan.
3. Pilih Ekstensi publik di bilah navigasi, lalu aktifkan tombol radio pihak ketiga di bawah Penerbit. Daftar ekstensi publik pihak ketiga yang tersedia muncul. (Anda juga dapat memilih AWSuntuk melihat daftar ekstensi publik yang diterbitkan oleh AWS, meskipun Anda tidak perlu mengaktifkannya.)
4. Jelajahi daftar dan temukan ekstensi yang ingin Anda aktifkan. Atau, cari, lalu aktifkan tombol radio di sudut kanan atas kartu ekstensi.

5. Pilih tombol Aktifkan di bagian atas daftar untuk mengaktifkan ekstensi yang dipilih. Halaman Aktifkan ekstensi muncul.
6. Di halaman Aktifkan, Anda dapat mengganti nama default ekstensi dan menentukan peran eksekusi dan konfigurasi logging. Anda juga dapat memilih apakah akan memperbarui ekstensi secara otomatis saat versi baru dirilis. Ketika Anda telah mengatur opsi ini sesuka Anda, pilih Aktifkan ekstensi di bagian bawah halaman.

Untuk mengaktifkan ekstensi pihak ketiga menggunakan AWS CLI

- Gunakan perintah `activate-type`. Gantikan ARN dari jenis kustom yang ingin Anda gunakan jika ditunjukkan.

  Berikut ini adalah contohnya:

  ```
  aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
  ```

Untuk mengaktifkan ekstensi pihak ketiga melalui CloudFormation atau CDK

- Menyebarkan sumber daya tipe `AWS::CloudFormation::TypeActivation` dan menentukan properti berikut:

  a. `TypeName`- Nama jenisnya, seperti`AWSQS::EKS::Cluster`.

  b. `MajorVersion`- Nomor versi utama ekstensi yang Anda inginkan. Abaikan jika Anda menginginkan versi terbaru.

  c. `AutoUpdate`- Apakah akan memperbarui ekstensi ini secara otomatis ketika versi minor baru dirilis oleh penerbit. (Pembaruan versi utama memerlukan perubahan `MajorVersion` properti secara eksplisit.)

  d. `ExecutionRoleArn`- ARN dari peran IAM di mana ekstensi ini akan berjalan.

  e. `LoggingConfig`- Konfigurasi logging untuk ekstensi.

  `TypeActivation`Sumber daya dapat digunakan oleh CDK menggunakan konstruksi. `CfnResource` Ini ditampilkan untuk ekstensi yang sebenarnya di bagian berikut.

Menambahkan sumber daya dari AWS CloudFormation Public Registry ke aplikasi CDK

Gunakan `CfnResource`konstruksi untuk menyertakan sumber daya dari AWS CloudFormation Public Registry dalam aplikasi Anda. Konstruksi ini ada di modul CDK. `aws-cdk-lib`

Misalnya, anggaplah ada sumber daya publik bernama `MY::S5::UltimateBucket` yang ingin Anda gunakan dalam AWS CDK aplikasi Anda. Sumber daya ini mengambil satu properti: nama bucket. `CfnResource`Instantiasi yang sesuai terlihat seperti ini.

TypeScript

```
const ubucket = new CfnResource(this, 'MyUltimateBucket', {
    type: 'MY::S5::UltimateBucket::MODULE',
    properties: {
           BucketName: 'UltimateBucket'
    }
});
```

JavaScript

```
const ubucket = new CfnResource(this, 'MyUltimateBucket', {
    type: 'MY::S5::UltimateBucket::MODULE',
    properties: {
           BucketName: 'UltimateBucket'
    }
});
```

Python

```
ubucket = CfnResource(self, "MyUltimateBucket",
            type="MY::S5::UltimateBucket::MODULE",
            properties=dict(
                BucketName="UltimateBucket"))
```

Java

```
CfnResource.Builder.create(this, "MyUltimateBucket")
 .type("MY::S5::UltimateBucket::MODULE")
 .properties(java.util.Map.of(    // Map.of requires Java 9+
```

```
     "BucketName", "UltimateBucket"))
 .build();
```

C#

```
new CfnResource(this, "MyUltimateBucket", new CfnResourceProps
{
    Type = "MY::S5::UltimateBucket::MODULE",
    Properties = new Dictionary<string, object>
    {
        ["BucketName"] = "UltimateBucket"
    }
});
```

Menyebarkan aplikasi AWS CDK

Menyebarkan AWS Cloud Development Kit (AWS CDK) aplikasi.

Topik

AWS CDK validasi kebijakan pada waktu sintesis

Topik

Validasi kebijakan pada waktu sintesis

Jika Anda atau organisasi Anda menggunakan alat validasi kebijakan apa pun, seperti AWS CloudFormation Guardatau OPA, untuk menentukan batasan pada AWS CloudFormation templat Anda, Anda dapat mengintegrasikannya dengan saat sintesis. AWS CDK Dengan menggunakan plugin validasi kebijakan yang sesuai, Anda dapat membuat AWS CDK aplikasi memeriksa AWS CloudFormation template yang dihasilkan terhadap kebijakan Anda segera setelah sintesis. Jika ada pelanggaran, sintesis akan gagal dan laporan akan dicetak ke konsol.

Validasi yang dilakukan oleh kontrol AWS CDK pada waktu sintesis memvalidasi pada satu titik dalam siklus hidup penerapan, tetapi tidak dapat memengaruhi tindakan yang terjadi di luar sintesis. Contohnya termasuk tindakan yang diambil langsung di konsol atau melalui API layanan. Mereka tidak tahan terhadap perubahan AWS CloudFormation template setelah sintesis. Beberapa mekanisme lain untuk memvalidasi aturan yang sama yang ditetapkan secara lebih otoritatif harus diatur secara independen, seperti AWS CloudFormation kait atau. AWS Config Namun demikian, kemampuan AWS CDK untuk mengevaluasi aturan yang ditetapkan selama pengembangan masih berguna karena akan meningkatkan kecepatan deteksi dan produktivitas pengembang.

Tujuan validasi AWS CDK kebijakan adalah untuk meminimalkan jumlah pengaturan yang diperlukan selama pengembangan, dan membuatnya semudah mungkin.

> **Note**
>
> Fitur ini dianggap eksperimental, dan API plugin dan format laporan validasi dapat berubah di masa mendatang.

Topik

Untuk pengembang aplikasi

Untuk menggunakan satu atau beberapa plugin validasi dalam aplikasi Anda, gunakan `policyValidationBeta1` properti: `Stage`

```
import { CfnGuardValidator } from '@cdklabs/cdk-validator-cfnguard';
const app = new App({
  policyValidationBeta1: [
    new CfnGuardValidator()
  ],
});
// only apply to a particular stage
const prodStage = new Stage(app, 'ProdStage', {
  policyValidationBeta1: [...],
});
```

Segera setelah sintesis, semua plugin yang terdaftar dengan cara ini akan dipanggil untuk memvalidasi semua template yang dihasilkan dalam lingkup yang Anda tentukan. Secara khusus, jika Anda mendaftarkan templat di `App` objek, semua templat akan dikenakan validasi.

> **Warning**
>
> Selain memodifikasi perakitan cloud, plugin dapat melakukan apa saja yang dapat dilakukan AWS CDK aplikasi Anda. Mereka dapat membaca data dari sistem file, mengakses jaringan, dll. Ini adalah tanggung jawab Anda sebagai konsumen plugin untuk memverifikasi bahwa itu aman untuk digunakan.

AWS CloudFormation Guard plugin

Menggunakan CfnGuardValidatorplugin memungkinkan Anda untuk menggunakan AWS CloudFormation Guarduntuk melakukan validasi kebijakan. `CfnGuardValidator`Plugin ini dilengkapi dengan satu set kontrol AWS Control Tower proaktif pilihan bawaan. Seperangkat aturan saat ini dapat ditemukan dalam dokumentasi proyek. Seperti disebutkan dalamValidasi kebijakan pada waktu sintesis, kami menyarankan agar organisasi menyiapkan metode validasi yang lebih otoritatif menggunakan AWS CloudFormation kait.

Untuk AWS Control Towerpelanggan, kontrol proaktif yang sama ini dapat diterapkan di seluruh organisasi Anda. Saat Anda mengaktifkan kontrol AWS Control Tower proaktif di AWS Control Tower lingkungan Anda, kontrol dapat menghentikan penyebaran sumber daya yang tidak sesuai yang digunakan melalui. AWS CloudFormation Untuk informasi selengkapnya tentang kontrol proaktif terkelola dan cara kerjanya, lihat AWS Control Tower dokumentasi.

Kontrol yang AWS CDK dibundel dan kontrol AWS Control Tower proaktif terkelola ini paling baik digunakan bersama. Dalam skenario ini Anda dapat mengonfigurasi plugin validasi ini dengan kontrol proaktif yang sama yang aktif di lingkungan AWS Control Tower cloud Anda. Anda kemudian dapat dengan cepat mendapatkan keyakinan bahwa AWS CDK aplikasi Anda akan melewati AWS Control Tower kontrol dengan menjalankan `cdk synth` secara lokal.

Laporan Validasi

Saat Anda mensintesis AWS CDK aplikasi, plugin validator akan dipanggil dan hasilnya akan dicetak. Contoh laporan ditampilkan di bawah ini.

```
Validation Report (CfnGuardValidator)
-------------------------------------
(Summary)
######################################
# Status    # failure               #
######################################
# Plugin    # CfnGuardValidator     #
######################################
(Violations)
Ensure S3 Buckets are encrypted with a KMS CMK (1 occurrences)
Severity: medium
  Occurrences:

    - Construct Path: MyStack/MyCustomL3Construct/Bucket
```

```
    - Stack Template Path: ./cdk.out/MyStack.template.json
    - Creation Stack:
        ###  MyStack (MyStack)
             # Library: aws-cdk-lib.Stack
             # Library Version: 2.50.0
             # Location: Object.<anonymous> (/home/johndoe/tmp/cdk-tmp-app/src/
main.ts:25:20)
             ###  MyCustomL3Construct (MyStack/MyCustomL3Construct)
                  # Library: N/A - (Local Construct)
                  # Library Version: N/A
                  # Location: new MyStack (/home/johndoe/tmp/cdk-tmp-app/src/
main.ts:15:20)
                  ###  Bucket (MyStack/MyCustomL3Construct/Bucket)
                       # Library: aws-cdk-lib/aws-s3.Bucket
                       # Library Version: 2.50.0
                       # Location: new MyCustomL3Construct (/home/johndoe/tmp/cdk-tmp-
app/src/main.ts:9:20)
    - Resource Name: my-bucket
    - Locations:
      > BucketEncryption/ServerSideEncryptionConfiguration/0/
ServerSideEncryptionByDefault/SSEAlgorithm
  Recommendation: Missing value for key `SSEAlgorithm` - must specify `aws:kms`
  How to fix:
    > Add to construct properties for `cdk-app/MyStack/Bucket`
      `encryption: BucketEncryption.KMS`

Validation failed. See above reports for details
```

Secara default, laporan akan dicetak dalam format yang dapat dibaca manusia. Jika Anda menginginkan laporan dalam format JSON, aktifkan menggunakan melalui CLI atau `@aws-cdk/core:validationReportJson` meneruskannya langsung ke aplikasi:

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

Atau, Anda dapat mengatur pasangan kunci-nilai konteks ini menggunakan `cdk.context.json` file `cdk.json` atau di direktori proyek Anda (lihatKonteks runtime).

Jika Anda memilih format JSON, AWS CDK akan mencetak laporan validasi kebijakan ke file yang dipanggil `policy-validation-report.json` di direktori perakitan cloud. Untuk format default yang dapat dibaca manusia, laporan akan dicetak ke output standar.

Untuk penulis plugin

Plugin

Kerangka AWS CDK inti bertanggung jawab untuk mendaftarkan dan memanggil plugin dan kemudian menampilkan laporan validasi yang diformat. Tanggung jawab plugin adalah bertindak sebagai lapisan terjemahan antara AWS CDK kerangka kerja dan alat validasi kebijakan. Plugin dapat dibuat dalam bahasa apa pun yang didukung oleh AWS CDK. Jika Anda membuat plugin yang mungkin dikonsumsi oleh beberapa bahasa maka disarankan agar Anda membuat plugin `TypeScript` sehingga Anda dapat menggunakan JSII untuk mempublikasikan plugin dalam setiap AWS CDK bahasa.

Membuat plugin

Protokol komunikasi antara modul AWS CDK inti dan alat kebijakan Anda ditentukan oleh `IPolicyValidationPluginBeta1` antarmuka. Untuk membuat plugin baru, Anda harus menulis kelas yang mengimplementasikan antarmuka ini. Ada dua hal yang perlu Anda terapkan: nama plugin (dengan mengganti `name` properti), dan `validate()` metode.

Kerangka kerja akan memanggil`validate()`, melewati `IValidationContextBeta1` objek. Lokasi template yang akan divalidasi diberikan oleh`templatePaths`. Plugin harus mengembalikan instance dari`ValidationPluginReportBeta1`. Objek ini mewakili laporan yang akan diterima pengguna pada akhir sintesis.

```
validate(context: IPolicyValidationContextBeta1): PolicyValidationReportBeta1 {
  // First read the templates using context.templatePaths...
  // ...then perform the validation, and then compose and return the report.
  // Using hard-coded values here for better clarity:
  return {
    success: false,
    violations: [{
      ruleName: 'CKV_AWS_117',
      description: 'Ensure that AWS Lambda function is configured inside a VPC',
      fix: 'https://docs.bridgecrew.io/docs/ensure-that-aws-lambda-function-is-
configured-inside-a-vpc-1',
      violatingResources: [{
        resourceName: 'MyFunction3BAA72D1',
        templatePath: '/home/johndoe/myapp/cdk.out/MyService.template.json',
        locations: 'Properties/VpcConfig',
      }],
```

```
    }],
  };
}
```

Perhatikan bahwa plugin tidak diizinkan untuk memodifikasi apa pun di perakitan cloud. Setiap upaya untuk melakukannya akan mengakibatkan kegagalan sintesis.

Jika plugin Anda bergantung pada alat eksternal, perlu diingat bahwa beberapa pengembang mungkin belum menginstal alat itu di workstation mereka. Untuk meminimalkan gesekan, kami sangat menyarankan Anda menyediakan beberapa skrip instalasi bersama dengan paket plugin Anda, untuk mengotomatiskan seluruh proses. Lebih baik lagi, jalankan skrip itu sebagai bagian dari instalasi paket Anda. Dengan`npm`, misalnya, Anda dapat menambahkannya ke `postinstall` skrip dalam `package.json` file.

Penanganan Pengecualian

Jika organisasi Anda memiliki mekanisme untuk menangani pengecualian, itu dapat diimplementasikan sebagai bagian dari plugin validator.

Contoh skenario untuk mengilustrasikan mekanisme pengecualian yang mungkin:

- Organisasi memiliki aturan bahwa bucket Amazon S3 publik tidak diizinkan, kecuali untuk skenario tertentu.
- Pengembang membuat bucket Amazon S3 yang termasuk dalam salah satu skenario tersebut dan meminta pengecualian (buat tiket misalnya).
- Perkakas keamanan tahu cara membaca dari sistem internal yang mendaftarkan pengecualian

Dalam skenario ini pengembang akan meminta pengecualian dalam sistem internal dan kemudian akan memerlukan beberapa cara untuk “mendaftarkan” pengecualian itu. Menambahkan ke contoh plugin penjaga, Anda dapat membuat plugin yang menangani pengecualian dengan memfilter pelanggaran yang memiliki pengecualian yang cocok dalam sistem tiket internal.

Lihat plugin yang ada misalnya implementasi.

- @cdklabs/cdk-validator-cfnguard

Integrasi dan pengiriman berkelanjutan (CI/CD) menggunakan CDK Pipelines

Gunakan modul CDK Pipelines dari AWS Construct Library untuk mengonfigurasi pengiriman aplikasi yang berkelanjutan. AWS CDK Saat Anda memasukkan kode sumber aplikasi CDK ke AWS CodeCommit,, atau GitHub AWS CodeStar, CDK Pipelines dapat secara otomatis membuat, menguji, dan menerapkan versi baru Anda.

CDK Pipelines diperbarui sendiri. Jika Anda menambahkan tahapan atau tumpukan aplikasi, pipeline secara otomatis mengkonfigurasi ulang dirinya sendiri untuk menerapkan tahapan atau tumpukan baru tersebut.

> **Note**
>
> CDK Pipelines mendukung dua API. Salah satunya adalah API asli yang tersedia di Pratinjau Pengembang CDK Pipelines. Yang lainnya adalah API modern yang menggabungkan umpan balik dari pelanggan CDK yang diterima selama fase pratinjau. Contoh dalam topik ini menggunakan API modern. Untuk detail tentang perbedaan antara dua API yang didukung, lihat API asli CDK Pipelines di repositori aws-cdk. GitHub

Topik

Bootstrap AWS lingkungan Anda

Sebelum Anda dapat menggunakan CDK Pipelines, Anda harus mem-bootstrap lingkungan AWS tempat Anda akan menyebarkan tumpukan Anda.

Pipeline CDK melibatkan setidaknya dua lingkungan. Lingkungan pertama adalah tempat pipa disediakan. Lingkungan kedua adalah tempat Anda ingin menyebarkan tumpukan atau tahapan aplikasi (tahapan adalah grup tumpukan terkait). Lingkungan ini bisa sama, tetapi rekomendasi praktik terbaik adalah mengisolasi tahapan satu sama lain di lingkungan yang berbeda.

> **Note**
>
> Lihat the section called “Bootstrapping” untuk informasi lebih lanjut tentang jenis sumber daya yang dibuat oleh bootstrap dan cara menyesuaikan tumpukan bootstrap.

Penerapan berkelanjutan dengan CDK Pipelines memerlukan hal-hal berikut untuk disertakan dalam tumpukan CDK Toolkit:

- Bucket Amazon Simple Storage Service (Amazon S3).
- Repositori Amazon ECR.
- IAM berperan untuk memberikan berbagai bagian pipa izin yang mereka butuhkan.

CDK Toolkit akan memutakhirkan tumpukan bootstrap yang ada atau membuat yang baru jika perlu.

Untuk mem-bootstrap lingkungan yang dapat menyediakan AWS CDK pipeline, panggil `cdk bootstrap` seperti yang ditunjukkan pada contoh berikut. Memanggil AWS CDK Toolkit melalui `npx` perintah untuk sementara menginstalnya jika perlu. Ini juga akan menggunakan versi Toolkit yang diinstal dalam proyek saat ini, jika ada.

--cloudformation-execution-policiesmenentukan ARN dari kebijakan di mana penerapan CDK Pipelines masa depan akan dijalankan. `AdministratorAccess`Kebijakan default memastikan bahwa pipeline Anda dapat menerapkan setiap jenis AWS sumber daya. Jika Anda menggunakan kebijakan ini, pastikan Anda mempercayai semua kode dan dependensi yang membentuk aplikasi Anda AWS CDK .

Sebagian besar organisasi mengamanatkan kontrol yang lebih ketat pada jenis sumber daya apa yang dapat digunakan oleh otomatisasi. Periksa dengan departemen yang sesuai dalam organisasi Anda untuk menentukan kebijakan yang harus digunakan pipeline Anda.

Anda dapat menghilangkan --profile opsi jika AWS profil default Anda berisi konfigurasi otentikasi yang diperlukan dan. Wilayah AWS

macOS/Linux

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE \
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
```

Windows

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE ^
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
```

Untuk mem-bootstrap lingkungan tambahan di mana AWS CDK aplikasi akan digunakan oleh pipeline, gunakan perintah berikut sebagai gantinya. --trustOpsi ini menunjukkan akun lain mana yang harus memiliki izin untuk menyebarkan AWS CDK aplikasi ke lingkungan ini. Untuk opsi ini, tentukan ID AWS akun pipeline.

Sekali lagi, Anda dapat menghilangkan --profile opsi jika AWS profil default Anda berisi konfigurasi otentikasi yang diperlukan dan. Wilayah AWS

macOS/Linux

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE \
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
 \
    --trust PIPELINE-ACCOUNT-NUMBER
```

Windows

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE ^
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
 ^
    --trust PIPELINE-ACCOUNT-NUMBER
```

> **Tip**
>
> Gunakan kredensi administratif hanya untuk bootstrap dan untuk menyediakan pipeline awal. Setelah itu, gunakan pipeline itu sendiri, bukan mesin lokal Anda, untuk menyebarkan perubahan.

Jika Anda memutakhirkan lingkungan bootstrapped lama, bucket Amazon S3 sebelumnya menjadi yatim piatu saat bucket baru dibuat. Hapus secara manual dengan menggunakan konsol Amazon S3.

Inisialisasi proyek

Buat GitHub proyek baru yang kosong dan kloning ke workstation Anda di direktori. `my-pipeline` (Contoh kode kami dalam topik ini digunakan GitHub. Anda juga dapat menggunakan AWS CodeStar atau AWS CodeCommit.)

```
git clone GITHUB-CLONE-URL my-pipeline
cd my-pipeline
```

> **Note**
>
> Anda dapat menggunakan nama selain `my-pipeline` untuk direktori utama aplikasi Anda. Namun, jika Anda melakukannya, Anda harus mengubah file dan nama kelas nanti dalam topik ini. Ini karena AWS CDK Toolkit mendasarkan beberapa nama file dan kelas pada nama direktori utama.

Setelah kloning, inisialisasi proyek seperti biasa.

TypeScript

```
cdk init app --language typescript
```

JavaScript

```
cdk init app --language javascript
```

Python

```
cdk init app --language python
```

Setelah aplikasi dibuat, masukkan juga dua perintah berikut. Ini mengaktifkan lingkungan virtual Python aplikasi dan menginstal dependensi AWS CDK inti.

```
source .venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

Java

```
cdk init app --language java
```

Jika Anda menggunakan IDE, Anda sekarang dapat membuka atau mengimpor proyek. Di Eclipse, misalnya, pilih File > Import > Maven > Existing Maven Projects. Pastikan bahwa pengaturan proyek diatur untuk menggunakan Java 8 (1.8).

C#

```
cdk init app --language csharp
```

Jika Anda menggunakan Visual Studio, buka file solusi di `src` direktori.

Go

```
cdk init app --language go
```

Setelah aplikasi dibuat, masukkan juga perintah berikut untuk menginstal modul AWS Construct Library yang dibutuhkan aplikasi.

```
go get
```

> **Important**
>
> Pastikan untuk memasukkan file `cdk.json` dan `cdk.context.json` file Anda ke kontrol sumber. Informasi konteks (seperti flag fitur dan nilai cache yang diambil dari AWS akun Anda) adalah bagian dari status proyek Anda. Nilainya mungkin berbeda di lingkungan lain, yang dapat menyebabkan perubahan tak terduga pada hasil Anda. Untuk informasi selengkapnya, lihat the section called “Konteks”.

Tentukan pipa

Aplikasi CDK Pipelines Anda akan menyertakan setidaknya dua tumpukan: satu yang mewakili pipeline itu sendiri, dan satu atau lebih tumpukan yang mewakili aplikasi yang digunakan melaluinya.

Tumpukan juga dapat dikelompokkan ke dalam beberapa tahap, yang dapat Anda gunakan untuk menyebarkan salinan tumpukan infrastruktur ke lingkungan yang berbeda. Untuk saat ini, kami akan mempertimbangkan pipeline, dan kemudian menyelidiki aplikasi yang akan disebarkan.

Konstruk `CodePipeline`adalah konstruksi yang mewakili Pipeline CDK yang digunakan AWS CodePipeline sebagai mesin penyebarannya. Saat membuat instance `CodePipeline` dalam tumpukan, Anda menentukan lokasi sumber untuk pipeline (seperti GitHub repositori). Anda juga menentukan perintah untuk membangun aplikasi.

Misalnya, berikut ini mendefinisikan pipa yang sumbernya disimpan dalam GitHub repositori. Ini juga mencakup langkah membangun untuk aplikasi TypeScript CDK. Isi informasi tentang GitHub repo Anda di mana ditunjukkan.

> **Note**
>
> Secara default, pipeline mengautentikasi GitHub menggunakan token akses pribadi yang disimpan di Secrets Manager dengan nama `github-token` tersebut.

Anda juga perlu memperbarui instantiasi tumpukan pipeline untuk menentukan AWS akun dan Wilayah.

TypeScript

Dalam `lib/my-pipeline-stack.ts` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { CodePipeline, CodePipelineSource, ShellStep } from 'aws-cdk-lib/pipelines';

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

    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
```

```
    });
  }
}
```

Dalam `bin/my-pipeline.ts` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
#!/usr/bin/env node
import * as cdk from 'aws-cdk-lib';
import { MyPipelineStack } from '../lib/my-pipeline-stack';

const app = new cdk.App();
new MyPipelineStack(app, 'MyPipelineStack', {
  env: {
    account: '111111111111',
    region: 'eu-west-1',
  }
});

app.synth();
```

JavaScript

Dalam `lib/my-pipeline-stack.js` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
const cdk = require('aws-cdk-lib');
const { CodePipeline, CodePipelineSource, ShellStep } = require('aws-cdk-lib/
pipelines');

 class MyPipelineStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });
  }
}
```

```
module.exports = { MyPipelineStack }
```

Dalam `bin/my-pipeline.js` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
#!/usr/bin/env node

const cdk = require('aws-cdk-lib');
const { MyPipelineStack } = require('../lib/my-pipeline-stack');

const app = new cdk.App();
new MyPipelineStack(app, 'MyPipelineStack', {
  env: {
    account: '111111111111',
    region: 'eu-west-1',
  }
});

app.synth();
```

Python

Dalam `my-pipeline/my-pipeline-stack.py` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
import aws_cdk as cdk
from constructs import Construct
from aws_cdk.pipelines import CodePipeline, CodePipelineSource, ShellStep

class MyPipelineStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        pipeline =  CodePipeline(self, "Pipeline",
                        pipeline_name="MyPipeline",
                        synth=ShellStep("Synth",
                            input=CodePipelineSource.git_hub("OWNER/REPO", "main"),
                            commands=["npm install -g aws-cdk",
                                "python -m pip install -r requirements.txt",
                                "cdk synth"]
                        )
```

```
                    )
```

Dalam `app.py`:

```
#!/usr/bin/env python3
import aws_cdk as cdk
from my_pipeline.my_pipeline_stack import MyPipelineStack

app = cdk.App()
MyPipelineStack(app, "MyPipelineStack",
    env=cdk.Environment(account="111111111111", region="eu-west-1")
)

app.synth()
```

Java

Dalam `src/main/java/com/myorg/MyPipelineStack.java` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
package com.myorg;

import java.util.Arrays;
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.pipelines.CodePipeline;
import software.amazon.awscdk.pipelines.CodePipelineSource;
import software.amazon.awscdk.pipelines.ShellStep;

public class MyPipelineStack extends Stack {
    public MyPipelineStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyPipelineStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
             .pipelineName("MyPipeline")
             .synth(ShellStep.Builder.create("Synth")
                .input(CodePipelineSource.gitHub("OWNER/REPO", "main"))
```

```
                .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))
                .build())
            .build();
    }
}
```

Dalam `src/main/java/com/myorg/MyPipelineApp.java` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.Environment;
import software.amazon.awscdk.StackProps;

public class MyPipelineApp {
    public static void main(final String[] args) {
        App app = new App();

        new MyPipelineStack(app, "PipelineStack", StackProps.builder()
            .env(Environment.builder()
                .account("111111111111")
                .region("eu-west-1")
                .build())
            .build());

        app.synth();
    }
}
```

C#

Dalam `src/MyPipeline/MyPipelineStack.cs` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
using Amazon.CDK;
using Amazon.CDK.Pipelines;

namespace MyPipeline
{
    public class MyPipelineStack : Stack
    {
```

```
        internal MyPipelineStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
            {
                PipelineName = "MyPipeline",
                Synth = new ShellStep("Synth", new ShellStepProps
                {
                    Input = CodePipelineSource.GitHub("OWNER/REPO", "main"),
                    Commands = new string[] { "npm install -g aws-cdk", "cdk
 synth" }
                })
            });
        }
    }
}
```

Dalam `src/MyPipeline/Program.cs` (dapat bervariasi jika folder proyek Anda tidak diberi nama`my-pipeline`):

```
using Amazon.CDK;

namespace MyPipeline
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new MyPipelineStack(app, "MyPipelineStack", new StackProps
            {
                Env = new Amazon.CDK.Environment {
                    Account = "111111111111", Region = "eu-west-1" }
            });

            app.Synth();
        }
    }
}
```

Anda harus menggunakan pipeline secara manual sekali. Setelah itu, pipeline tetap up to date dari repositori kode sumber. Jadi pastikan bahwa kode dalam repo adalah kode yang ingin Anda gunakan. Periksa perubahan Anda dan dorong ke GitHub, lalu terapkan:

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

> **Tip**
>
> Sekarang setelah Anda melakukan penerapan awal, AWS akun lokal Anda tidak lagi memerlukan akses administratif. Ini karena semua perubahan pada aplikasi Anda akan diterapkan melalui pipeline. Yang perlu Anda lakukan hanyalah mendorong GitHub.

Tahapan aplikasi

Untuk menentukan AWS aplikasi multi-stack yang dapat ditambahkan ke pipeline sekaligus, tentukan subclass dari. `Stage` (Ini berbeda dengan modul `CdkStage` CDK Pipelines.)

Panggung berisi tumpukan yang membentuk aplikasi Anda. Jika ada dependensi antara tumpukan, tumpukan secara otomatis ditambahkan ke pipa dalam urutan yang benar. Tumpukan yang tidak bergantung satu sama lain digunakan secara paralel. Anda dapat menambahkan hubungan ketergantungan antar tumpukan dengan menelepon. `stack1.addDependency(stack2)`

Tahapan menerima `env` argumen default, yang menjadi lingkungan default untuk tumpukan di dalamnya. (Tumpukan masih dapat memiliki lingkungannya sendiri yang ditentukan.).

Aplikasi ditambahkan ke pipeline dengan menelepon `addStage()` dengan instance. `Stage` Sebuah tahap dapat dipakai dan ditambahkan ke pipeline beberapa kali untuk menentukan tahapan yang berbeda dari pipeline aplikasi DTAP atau Multi-region Anda.

Kami akan membuat tumpukan yang berisi fungsi Lambda sederhana dan menempatkan tumpukan itu di panggung. Kemudian kita akan menambahkan stage ke pipeline sehingga bisa dikerahkan.

TypeScript

Buat file baru `lib/my-pipeline-lambda-stack.ts` untuk menampung tumpukan aplikasi kami yang berisi fungsi Lambda.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { Function, InlineCode, Runtime } from 'aws-cdk-lib/aws-lambda';

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

      new Function(this, 'LambdaFunction', {
        runtime: Runtime.NODEJS_18_X,
        handler: 'index.handler',
        code: new InlineCode('exports.handler = _ => "Hello, CDK";')
      });
    }
}
```

Buat file baru `lib/my-pipeline-app-stage.ts` untuk menahan panggung kami.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from "constructs";
import { MyLambdaStack } from './my-pipeline-lambda-stack';

export class MyPipelineAppStage extends cdk.Stage {

    constructor(scope: Construct, id: string, props?: cdk.StageProps) {
      super(scope, id, props);

      const lambdaStack = new MyLambdaStack(this, 'LambdaStack');
    }
}
```

Edit `lib/my-pipeline-stack.ts` untuk menambahkan panggung ke pipeline kami.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { CodePipeline, CodePipelineSource, ShellStep } from 'aws-cdk-lib/pipelines';
import { MyPipelineAppStage } from './my-pipeline-app-stage';

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

```
    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });

    pipeline.addStage(new MyPipelineAppStage(this, "test", {
      env: { account: "111111111111", region: "eu-west-1" }
    }));
  }
}
```

JavaScript

Buat file baru `lib/my-pipeline-lambda-stack.js` untuk menampung tumpukan aplikasi kami yang berisi fungsi Lambda.

```
const cdk = require('aws-cdk-lib');
const { Function, InlineCode, Runtime } = require('aws-cdk-lib/aws-lambda');

class MyLambdaStack extends cdk.Stack {
    constructor(scope, id, props) {
      super(scope, id, props);

      new Function(this, 'LambdaFunction', {
        runtime: Runtime.NODEJS_18_X,
        handler: 'index.handler',
        code: new InlineCode('exports.handler = _ => "Hello, CDK";')
      });
    }
}

module.exports = { MyLambdaStack }
```

Buat file baru `lib/my-pipeline-app-stage.js` untuk menahan panggung kami.

```
const cdk = require('aws-cdk-lib');
const { MyLambdaStack } = require('./my-pipeline-lambda-stack');

class MyPipelineAppStage extends cdk.Stage {
```

```
    constructor(scope, id, props) {
      super(scope, id, props);

      const lambdaStack = new MyLambdaStack(this, 'LambdaStack');
    }
}

module.exports = { MyPipelineAppStage };
```

Edit `lib/my-pipeline-stack.ts` untuk menambahkan panggung ke pipeline kami.

```
const cdk = require('aws-cdk-lib');
const { CodePipeline, CodePipelineSource, ShellStep } = require('aws-cdk-lib/
pipelines');
const { MyPipelineAppStage } = require('./my-pipeline-app-stage');

 class MyPipelineStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });

    pipeline.addStage(new MyPipelineAppStage(this, "test", {
      env: { account: "111111111111", region: "eu-west-1" }
    }));

  }
}

module.exports = { MyPipelineStack }
```

Python

Buat file baru `my_pipeline/my_pipeline_lambda_stack.py` untuk menampung tumpukan aplikasi kami yang berisi fungsi Lambda.

```
import aws_cdk as cdk
```

```
from constructs import Construct
from aws_cdk.aws_lambda import Function, InlineCode, Runtime

class MyLambdaStack(cdk.Stack):
    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        Function(self, "LambdaFunction",
            runtime=Runtime.NODEJS_18_X,
            handler="index.handler",
            code=InlineCode("exports.handler = _ => 'Hello, CDK';")
        )
```

Buat file baru `my_pipeline/my_pipeline_app_stage.py` untuk menahan panggung kami.

```
import aws_cdk as cdk
from constructs import Construct
from my_pipeline.my_pipeline_lambda_stack import MyLambdaStack

class MyPipelineAppStage(cdk.Stage):
    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        lambdaStack = MyLambdaStack(self, "LambdaStack")
```

Edit `my_pipeline/my-pipeline-stack.py` untuk menambahkan panggung ke pipeline kami.

```
import aws_cdk as cdk
from constructs import Construct
from aws_cdk.pipelines import CodePipeline, CodePipelineSource, ShellStep
from my_pipeline.my_pipeline_app_stage import MyPipelineAppStage

class MyPipelineStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        pipeline =  CodePipeline(self, "Pipeline",
                        pipeline_name="MyPipeline",
                        synth=ShellStep("Synth",
                            input=CodePipelineSource.git_hub("OWNER/REPO", "main"),
                            commands=["npm install -g aws-cdk",
                                "python -m pip install -r requirements.txt",
```

```
                                "cdk synth"]))

        pipeline.add_stage(MyPipelineAppStage(self, "test",
            env=cdk.Environment(account="111111111111", region="eu-west-1")))
```

Java

Buat file baru `src/main/java/com.myorg/MyPipelineLambdaStack.java` untuk menampung tumpukan aplikasi kami yang berisi fungsi Lambda.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.lambda.InlineCode;

public class MyPipelineLambdaStack extends Stack {
    public MyPipelineLambdaStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyPipelineLambdaStack(final Construct scope, final String id, final
 StackProps props) {
        super(scope, id, props);

        Function.Builder.create(this, "LambdaFunction")
          .runtime(Runtime.NODEJS_18_X)
          .handler("index.handler")
          .code(new InlineCode("exports.handler = _ => 'Hello, CDK';"))
          .build();

    }

}
```

Buat file baru `src/main/java/com.myorg/MyPipelineAppStage.java` untuk menahan panggung kami.

```
package com.myorg;
```

```
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.Stage;
import software.amazon.awscdk.StageProps;

public class MyPipelineAppStage extends Stage {
    public MyPipelineAppStage(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyPipelineAppStage(final Construct scope, final String id, final
 StageProps props) {
        super(scope, id, props);

        Stack lambdaStack = new MyPipelineLambdaStack(this, "LambdaStack");
    }

}
```

Edit `src/main/java/com.myorg/MyPipelineStack.java` untuk menambahkan panggung ke pipeline kami.

```
package com.myorg;

import java.util.Arrays;
import software.constructs.Construct;
import software.amazon.awscdk.Environment;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.StageProps;
import software.amazon.awscdk.pipelines.CodePipeline;
import software.amazon.awscdk.pipelines.CodePipelineSource;
import software.amazon.awscdk.pipelines.ShellStep;

public class MyPipelineStack extends Stack {
    public MyPipelineStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public MyPipelineStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);
```

```
        final CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
            .pipelineName("MyPipeline")
            .synth(ShellStep.Builder.create("Synth")
                .input(CodePipelineSource.gitHub("OWNER/REPO", "main"))
                .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))
                .build())
            .build();

        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
            .env(Environment.builder()
                .account("111111111111")
                .region("eu-west-1")
                .build())
            .build()));
    }
}
```

C#

Buat file baru `src/MyPipeline/MyPipelineLambdaStack.cs` untuk menampung tumpukan aplikasi kami yang berisi fungsi Lambda.

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.AWS.Lambda;

namespace MyPipeline
{
    class MyPipelineLambdaStack : Stack
    {
        public MyPipelineLambdaStack(Construct scope, string id, StackProps
 props=null) : base(scope, id, props)
        {
            new Function(this, "LambdaFunction", new FunctionProps
            {
                Runtime = Runtime.NODEJS_18_X,
                Handler = "index.handler",
                Code = new InlineCode("exports.handler = _ => 'Hello, CDK';")
            });
        }
    }
}
```

Buat file baru `src/MyPipeline/MyPipelineAppStage.cs` untuk menahan panggung kami.

```
using Amazon.CDK;
using Constructs;

namespace MyPipeline
{
    class MyPipelineAppStage : Stage
    {
        public MyPipelineAppStage(Construct scope, string id, StageProps
 props=null) : base(scope, id, props)
        {
            Stack lambdaStack = new MyPipelineLambdaStack(this, "LambdaStack");
        }
    }
}
```

Edit `src/MyPipeline/MyPipelineStack.cs` untuk menambahkan panggung ke pipeline kami.

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.Pipelines;

namespace MyPipeline
{
    public class MyPipelineStack : Stack
    {
        internal MyPipelineStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
            {
                PipelineName = "MyPipeline",
                Synth = new ShellStep("Synth", new ShellStepProps
                {
                    Input = CodePipelineSource.GitHub("OWNER/REPO", "main"),
                    Commands = new string[] { "npm install -g aws-cdk", "cdk
 synth" }
                })
            });

            pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps
```

```
            {
                Env = new Environment
                {
                    Account = "111111111111", Region = "eu-west-1"
                }
            }));
        }
    }
}
```

Setiap tahap aplikasi yang ditambahkan oleh `addStage()` hasil penambahan tahap pipeline yang sesuai, diwakili oleh StageDeploymentinstance yang dikembalikan oleh `addStage()` panggilan. Anda dapat menambahkan tindakan pra-penerapan atau pasca-penerapan ke panggung dengan memanggil metodenya atau. `addPre()` `addPost()`

TypeScript

```
// import { ManualApprovalStep } from 'aws-cdk-lib/pipelines';

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, 'testing', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

    testingStage.addPost(new ManualApprovalStep('approval'));
```

JavaScript

```
// const { ManualApprovalStep } = require('aws-cdk-lib/pipelines');

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, 'testing', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

testingStage.addPost(new ManualApprovalStep('approval'));
```

Python

```
# from aws_cdk.pipelines import ManualApprovalStep

testing_stage = pipeline.add_stage(MyPipelineAppStage(self, "testing",
    env=cdk.Environment(account="111111111111", region="eu-west-1")))
```

```
testing_stage.add_post(ManualApprovalStep('approval'))
```

Java

```
// import software.amazon.awscdk.pipelines.StageDeployment;
// import software.amazon.awscdk.pipelines.ManualApprovalStep;

StageDeployment testingStage =
        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
                .env(Environment.builder()
                        .account("111111111111")
                        .region("eu-west-1")
                        .build())
                .build()));

testingStage.addPost(new ManualApprovalStep("approval"));
```

C#

```
var testingStage = pipeline.AddStage(new MyPipelineAppStage(this, "test", new
 StageProps
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));

testingStage.AddPost(new ManualApprovalStep("approval"));
```

Anda dapat menambahkan tahapan ke Wave untuk menerapkannya secara paralel, misalnya saat menerapkan tahapan ke beberapa akun atau Wilayah.

TypeScript

```
const wave = pipeline.addWave('wave');
wave.addStage(new MyApplicationStage(this, 'MyAppEU', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));
wave.addStage(new MyApplicationStage(this, 'MyAppUS', {
```

```
  env: { account: '111111111111', region: 'us-west-1' }
}));
```

JavaScript

```
const wave = pipeline.addWave('wave');
wave.addStage(new MyApplicationStage(this, 'MyAppEU', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));
wave.addStage(new MyApplicationStage(this, 'MyAppUS', {
  env: { account: '111111111111', region: 'us-west-1' }
}));
```

Python

```
wave = pipeline.add_wave("wave")
wave.add_stage(MyApplicationStage(self, "MyAppEU",
    env=cdk.Environment(account="111111111111", region="eu-west-1")))
wave.add_stage(MyApplicationStage(self, "MyAppUS",
    env=cdk.Environment(account="111111111111", region="us-west-1")))
```

Java

```
// import software.amazon.awscdk.pipelines.Wave;
final Wave wave = pipeline.addWave("wave");
wave.addStage(new MyPipelineAppStage(this, "MyAppEU", StageProps.builder()
        .env(Environment.builder()
                .account("111111111111")
                .region("eu-west-1")
                .build())
        .build()));
wave.addStage(new MyPipelineAppStage(this, "MyAppUS", StageProps.builder()
        .env(Environment.builder()
                .account("111111111111")
                .region("us-west-1")
                .build())
        .build()));
```

C#

```
var wave = pipeline.AddWave("wave");
wave.AddStage(new MyPipelineAppStage(this, "MyAppEU", new StageProps
```

```
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));
wave.AddStage(new MyPipelineAppStage(this, "MyAppUS", new StageProps
{
    Env = new Environment
    {
        Account = "111111111111", Region = "us-west-1"
    }
}));
```

Pengujian penerapan

Anda dapat menambahkan langkah-langkah ke Pipeline CDK untuk memvalidasi penerapan yang sedang Anda lakukan. Misalnya, Anda dapat menggunakan perpustakaan CDK Pipeline `ShellStep` untuk melakukan tugas-tugas seperti berikut:

- Mencoba mengakses Amazon API Gateway yang baru diterapkan yang didukung oleh fungsi Lambda
- Memeriksa pengaturan sumber daya yang digunakan dengan mengeluarkan perintah AWS CLI

Dalam bentuknya yang paling sederhana, menambahkan tindakan validasi terlihat seperti ini:

TypeScript

```
// stage was returned by pipeline.addStage

stage.addPost(new ShellStep("validate", {
  commands: ['../tests/validate.sh'],
}));
```

JavaScript

```
// stage was returned by pipeline.addStage

stage.addPost(new ShellStep("validate", {
  commands: ['../tests/validate.sh'],
```

```
}));
```

Python

```
# stage was returned by pipeline.add_stage

stage.add_post(ShellStep("validate",
  commands=[''../tests/validate.sh'']
))
```

Java

```
// stage was returned by pipeline.addStage

stage.addPost(ShellStep.Builder.create("validate")
        .commands(Arrays.asList("'../tests/validate.sh'"))
        .build());
```

C#

```
// stage was returned by pipeline.addStage

stage.AddPost(new ShellStep("validate", new ShellStepProps
{
    Commands = new string[] { "'../tests/validate.sh'" }
}));
```

Banyak AWS CloudFormation penerapan menghasilkan generasi sumber daya dengan nama yang tidak dapat diprediksi. Karena itu, CDK Pipelines menyediakan cara untuk AWS CloudFormation membaca output setelah penerapan. Ini memungkinkan untuk meneruskan (misalnya) URL yang dihasilkan dari penyeimbang beban ke tindakan pengujian.

Untuk menggunakan output, paparkan `CfnOutput` objek yang Anda minati. Kemudian, berikan dalam `envFromCfnOutputs` properti langkah untuk membuatnya tersedia sebagai variabel lingkungan dalam langkah itu.

TypeScript

```
// given a stack lbStack that exposes a load balancer construct as loadBalancer
this.loadBalancerAddress = new cdk.CfnOutput(lbStack, 'LbAddress', {
```

```
  value: `https://${lbStack.loadBalancer.loadBalancerDnsName}/`
});

// pass the load balancer address to a shell step
stage.addPost(new ShellStep("lbaddr", {
  envFromCfnOutputs: {lb_addr: lbStack.loadBalancerAddress},
  commands: ['echo $lb_addr']
}));
```

JavaScript

```
// given a stack lbStack that exposes a load balancer construct as loadBalancer
this.loadBalancerAddress = new cdk.CfnOutput(lbStack, 'LbAddress', {
  value: `https://${lbStack.loadBalancer.loadBalancerDnsName}/`
});

// pass the load balancer address to a shell step
stage.addPost(new ShellStep("lbaddr", {
  envFromCfnOutputs: {lb_addr: lbStack.loadBalancerAddress},
  commands: ['echo $lb_addr']
}));
```

Python

```
# given a stack lb_stack that exposes a load balancer construct as load_balancer
self.load_balancer_address = cdk.CfnOutput(lb_stack, "LbAddress",
    value=f"https://{lb_stack.load_balancer.load_balancer_dns_name}/")

# pass the load balancer address to a shell step
stage.add_post(ShellStep("lbaddr",
    env_from_cfn_outputs={"lb_addr": lb_stack.load_balancer_address}
    commands=["echo $lb_addr"]))
```

Java

```
// given a stack lbStack that exposes a load balancer construct as loadBalancer
loadBalancerAddress = CfnOutput.Builder.create(lbStack, "LbAddress")
                            .value(String.format("https://%s/",
                                    lbStack.loadBalancer.loadBalancerDnsName))
                            .build();

stage.addPost(ShellStep.Builder.create("lbaddr")
    .envFromCfnOutputs(     // Map.of requires Java 9 or later
```

```
        java.util.Map.of("lbAddr", loadBalancerAddress))
    .commands(Arrays.asList("echo $lbAddr"))
    .build());
```

C#

```
// given a stack lbStack that exposes a load balancer construct as loadBalancer
loadBalancerAddress = new CfnOutput(lbStack, "LbAddress", new CfnOutputProps
{
    Value = string.Format("https://{0}/", lbStack.loadBalancer.LoadBalancerDnsName)
});

stage.AddPost(new ShellStep("lbaddr", new ShellStepProps
{
    EnvFromCfnOutputs = new Dictionary<string, CfnOutput>
    {
        {  "lbAddr", loadBalancerAddress }
    },
    Commands = new string[] { "echo $lbAddr" }
}));
```

Anda dapat menulis tes validasi sederhana tepat di`ShellStep`, tetapi pendekatan ini menjadi berat ketika tes lebih dari beberapa baris. Untuk pengujian yang lebih kompleks, Anda dapat membawa file tambahan (seperti skrip shell lengkap, atau program dalam bahasa lain) ke dalam `inputs` properti `ShellStep` via. Input dapat berupa langkah apa pun yang memiliki output, termasuk sumber (seperti GitHub repo) atau lainnya. `ShellStep`

Membawa file dari repositori sumber sesuai jika file langsung dapat digunakan dalam pengujian (misalnya, jika file itu sendiri dapat dieksekusi). Dalam contoh ini, kami mendeklarasikan GitHub repo kami sebagai `source` (daripada membuat instance sebaris sebagai bagian dari). `CodePipeline` Kemudian, kami meneruskan fileset ini ke pipeline dan uji validasi.

TypeScript

```
const source = CodePipelineSource.gitHub('OWNER/REPO', 'main');

const pipeline = new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyPipeline',
  synth: new ShellStep('Synth', {
    input: source,
    commands: ['npm ci', 'npm run build', 'npx cdk synth']
```

```
  })
});

const stage = pipeline.addStage(new MyPipelineAppStage(this, 'test', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

stage.addPost(new ShellStep('validate', {
  input: source,
  commands: ['sh ../tests/validate.sh']
}));
```

JavaScript

```
const source = CodePipelineSource.gitHub('OWNER/REPO', 'main');

const pipeline = new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyPipeline',
  synth: new ShellStep('Synth', {
    input: source,
    commands: ['npm ci', 'npm run build', 'npx cdk synth']
  })
});

const stage = pipeline.addStage(new MyPipelineAppStage(this, 'test', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

stage.addPost(new ShellStep('validate', {
  input: source,
  commands: ['sh ../tests/validate.sh']
}));
```

Python

```
source   = CodePipelineSource.git_hub("OWNER/REPO", "main")

pipeline =  CodePipeline(self, "Pipeline",
                pipeline_name="MyPipeline",
                synth=ShellStep("Synth",
                    input=source,
                    commands=["npm install -g aws-cdk",
                        "python -m pip install -r requirements.txt",
```

```
                        "cdk synth"]))

stage = pipeline.add_stage(MyApplicationStage(self, "test",
            env=cdk.Environment(account="111111111111", region="eu-west-1")))

stage.add_post(ShellStep("validate", input=source,
    commands=["sh ../tests/validate.sh"],
))
```

Java

```
final CodePipelineSource source = CodePipelineSource.gitHub("OWNER/REPO", "main");

final CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
        .pipelineName("MyPipeline")
        .synth(ShellStep.Builder.create("Synth")
                .input(source)
                .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))
                .build())
        .build();

final StageDeployment stage =
        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
                .env(Environment.builder()
                        .account("111111111111")
                        .region("eu-west-1")
                        .build())
                .build()));

stage.addPost(ShellStep.Builder.create("validate")
        .input(source)
        .commands(Arrays.asList("sh ../tests/validate.sh"))
        .build());
```

C#

```
var source = CodePipelineSource.GitHub("OWNER/REPO", "main");

var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
{
    PipelineName = "MyPipeline",
    Synth = new ShellStep("Synth", new ShellStepProps
    {
```

```
        Input = source,
        Commands = new string[] { "npm install -g aws-cdk", "cdk synth" }
    })
});

var stage = pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));

stage.AddPost(new ShellStep("validate", new ShellStepProps
{
    Input = source,
    Commands = new string[] { "sh ../tests/validate.sh" }
}));
```

Mendapatkan file tambahan dari langkah synth sesuai jika pengujian Anda perlu dikompilasi, yang dilakukan sebagai bagian dari sintesis.

TypeScript

```
const synthStep = new ShellStep('Synth', {
  input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
  commands: ['npm ci', 'npm run build', 'npx cdk synth'],
});

const pipeline = new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyPipeline',
  synth: synthStep
});

const stage = pipeline.addStage(new MyPipelineAppStage(this, 'test', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

// run a script that was transpiled from TypeScript during synthesis
stage.addPost(new ShellStep('validate', {
  input: synthStep,
  commands: ['node tests/validate.js']
```

```
}));
```

JavaScript

```
const synthStep = new ShellStep('Synth', {
  input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
  commands: ['npm ci', 'npm run build', 'npx cdk synth'],
});

const pipeline = new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyPipeline',
  synth: synthStep
});

const stage = pipeline.addStage(new MyPipelineAppStage(this, "test", {
  env: { account: "111111111111", region: "eu-west-1" }
}));

// run a script that was transpiled from TypeScript during synthesis
stage.addPost(new ShellStep('validate', {
  input: synthStep,
  commands: ['node tests/validate.js']
}));
```

Python

```
synth_step = ShellStep("Synth",
                input=CodePipelineSource.git_hub("OWNER/REPO", "main"),
                commands=["npm install -g aws-cdk",
                  "python -m pip install -r requirements.txt",
                  "cdk synth"])

pipeline   = CodePipeline(self, "Pipeline",
                pipeline_name="MyPipeline",
                synth=synth_step)

stage = pipeline.add_stage(MyApplicationStage(self, "test",
            env=cdk.Environment(account="111111111111", region="eu-west-1")))

# run a script that was compiled during synthesis
stage.add_post(ShellStep("validate",
    input=synth_step,
    commands=["node test/validate.js"],
```

```
))
```

Java

```
final ShellStep synth = ShellStep.Builder.create("Synth")
                            .input(CodePipelineSource.gitHub("OWNER/REPO", "main"))
                            .commands(Arrays.asList("npm install -g aws-cdk", "cdk
 synth"))
                            .build();

final CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
        .pipelineName("MyPipeline")
        .synth(synth)
        .build();

final StageDeployment stage =
        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
                .env(Environment.builder()
                        .account("111111111111")
                        .region("eu-west-1")
                        .build())
                .build()));

stage.addPost(ShellStep.Builder.create("validate")
        .input(synth)
        .commands(Arrays.asList("node ./tests/validate.js"))
        .build());
```

C#

```
var synth = new ShellStep("Synth", new ShellStepProps
{
    Input = CodePipelineSource.GitHub("OWNER/REPO", "main"),
    Commands = new string[] { "npm install -g aws-cdk", "cdk synth" }
});

var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
{
    PipelineName = "MyPipeline",
    Synth = synth
});

var stage = pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps
```

```
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));

stage.AddPost(new ShellStep("validate", new ShellStepProps
{
    Input = synth,
    Commands = new string[] { "node ./tests/validate.js" }
}));
```

Catatan keamanan

Segala bentuk pengiriman berkelanjutan memiliki risiko keamanan yang melekat. Di bawah Model Tanggung Jawab AWS Bersama, Anda bertanggung jawab atas keamanan informasi Anda di AWS Cloud. Pustaka CDK Pipelines memberi Anda awal yang baik dengan menggabungkan default aman dan praktik terbaik pemodelan.

Namun, pada dasarnya, perpustakaan yang membutuhkan akses tingkat tinggi untuk memenuhi tujuan yang dimaksudkan tidak dapat menjamin keamanan yang lengkap. Ada banyak vektor serangan di luar AWS dan organisasi Anda.

Secara khusus, perlu diingat hal-hal berikut:

- Berhati-hatilah dengan perangkat lunak yang Anda andalkan. Periksa semua perangkat lunak pihak ketiga yang Anda jalankan di pipeline Anda, karena dapat mengubah infrastruktur yang akan digunakan.
- Gunakan penguncian ketergantungan untuk mencegah peningkatan yang tidak disengaja. CDK Pipelines `package-lock.json` menghormati `yarn.lock` dan memastikan bahwa dependensi Anda adalah yang Anda harapkan.
- CDK Pipelines berjalan pada sumber daya yang dibuat di akun Anda sendiri, dan konfigurasi sumber daya tersebut dikendalikan oleh pengembang yang mengirimkan kode melalui pipeline. Oleh karena itu, CDK Pipelines dengan sendirinya tidak dapat melindungi dari pengembang jahat yang mencoba melewati pemeriksaan kepatuhan. Jika model ancaman Anda menyertakan pengembang yang menulis kode CDK, Anda harus memiliki mekanisme kepatuhan eksternal seperti AWS CloudFormation Hooks (preventif) atau AWS Config (reaktif) yang Peran AWS CloudFormation Eksekusi tidak memiliki izin untuk dinonaktifkan.

- Kredensi untuk lingkungan produksi harus berumur pendek. Setelah bootstrap dan penyediaan awal, pengembang tidak perlu memiliki kredensi akun sama sekali. Perubahan dapat diterapkan melalui pipa. Kurangi kemungkinan kredensil bocor dengan tidak membutuhkannya sejak awal.

Pemecahan Masalah

Masalah berikut biasanya ditemui saat memulai dengan CDK Pipelines.

Pipa: Kegagalan Internal

```
CREATE_FAILED  | AWS::CodePipeline::Pipeline | Pipeline/Pipeline
Internal Failure
```

Periksa token GitHub akses Anda. Mungkin hilang, atau mungkin tidak memiliki izin untuk mengakses repositori.

Kunci: Kebijakan berisi pernyataan dengan satu atau beberapa prinsip yang tidak valid

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey
Policy contains a statement with one or more invalid principals.
```

Salah satu lingkungan target belum di-bootstrap dengan tumpukan bootstrap baru. Pastikan semua lingkungan target Anda di-bootstrap.

Stack dalam status ROLLBACK_COMPLETE dan tidak dapat diperbarui.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
ID: ...)
```

Tumpukan gagal penerapan sebelumnya dan dalam keadaan tidak dapat dicoba ulang. Hapus tumpukan dari AWS CloudFormation konsol dan coba lagi penerapan.

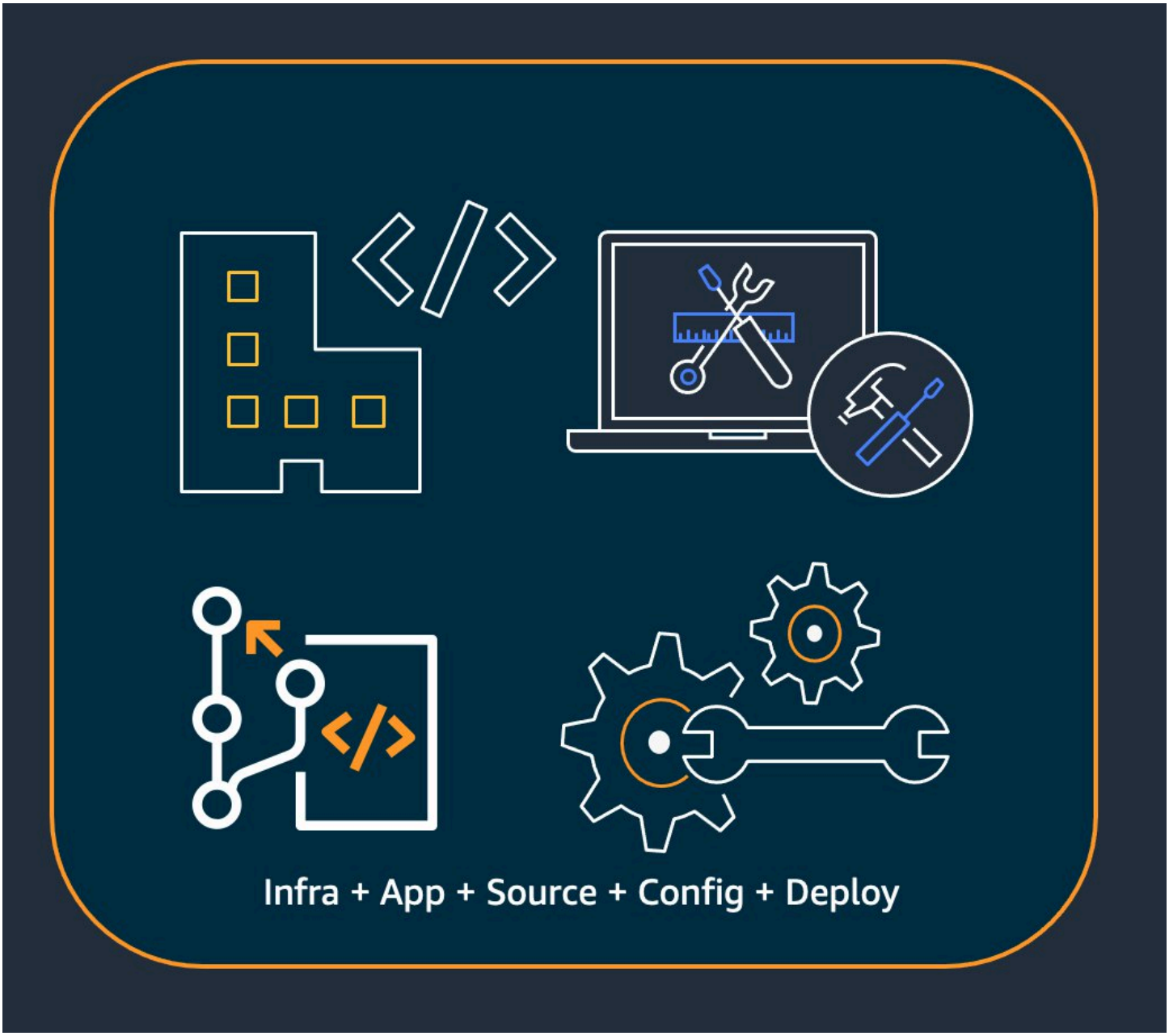
Praktik terbaik untuk mengembangkan dan menerapkan infrastruktur cloud dengan AWS CDK

Dengan AWS CDK, pengembang atau administrator dapat menentukan infrastruktur cloud mereka dengan menggunakan bahasa pemrograman yang didukung. Aplikasi CDK harus diatur ke dalam unit logis, seperti API, database, dan sumber daya pemantauan, dan secara opsional memiliki pipeline untuk penerapan otomatis. Unit logis harus diimplementasikan sebagai konstruksi termasuk yang berikut:

- Infrastruktur (seperti bucket Amazon S3, database Amazon RDS, atau jaringan VPC Amazon)
- Kode runtime (seperti AWS Lambda fungsi)
- Kode konfigurasi

Tumpukan mendefinisikan model penyebaran unit logis ini. Untuk pengantar yang lebih rinci tentang konsep di balik CDK, lihatMemulai.

Ini AWS CDK mencerminkan pertimbangan yang cermat terhadap kebutuhan pelanggan dan tim internal kami dan pola kegagalan yang sering muncul selama penyebaran dan pemeliharaan berkelanjutan aplikasi cloud yang kompleks. Kami menemukan bahwa kegagalan sering terkait dengan perubahan out-of-band "" pada aplikasi yang tidak sepenuhnya diuji, seperti perubahan konfigurasi. Oleh karena itu, kami mengembangkan model di mana seluruh aplikasi Anda didefinisikan dalam kode, tidak hanya logika bisnis tetapi juga infrastruktur dan konfigurasi. AWS CDK Dengan begitu, perubahan yang diusulkan dapat ditinjau dengan cermat, diuji secara komprehensif di lingkungan yang menyerupai produksi dengan tingkat yang berbeda-beda, dan sepenuhnya diputar kembali jika terjadi kesalahan.



Pada waktu penerapan, AWS CDK mensintesis rakitan cloud yang berisi hal-hal berikut:

- AWS CloudFormation template yang menggambarkan infrastruktur Anda di semua lingkungan target
- Aset file yang berisi kode runtime Anda dan file pendukungnya

Dengan CDK, setiap komit di cabang kontrol versi utama aplikasi Anda dapat mewakili versi aplikasi Anda yang lengkap, konsisten, dan dapat diterapkan. Aplikasi Anda kemudian dapat digunakan secara otomatis setiap kali perubahan dilakukan.

Filosofi di balik AWS CDK mengarah pada praktik terbaik yang kami rekomendasikan, yang telah kami bagi menjadi empat kategori besar.

- the section called “Praktik terbaik organisasi”
- the section called “Praktik terbaik pengkodean”
- the section called “Membangun praktik terbaik”
- the section called “Praktik terbaik aplikasi”

> **Tip**
>
> Pertimbangkan juga praktik terbaik untuk AWS CloudFormation dan AWS layanan individual yang Anda gunakan, jika berlaku untuk infrastruktur yang ditentukan CDK.

Praktik terbaik organisasi

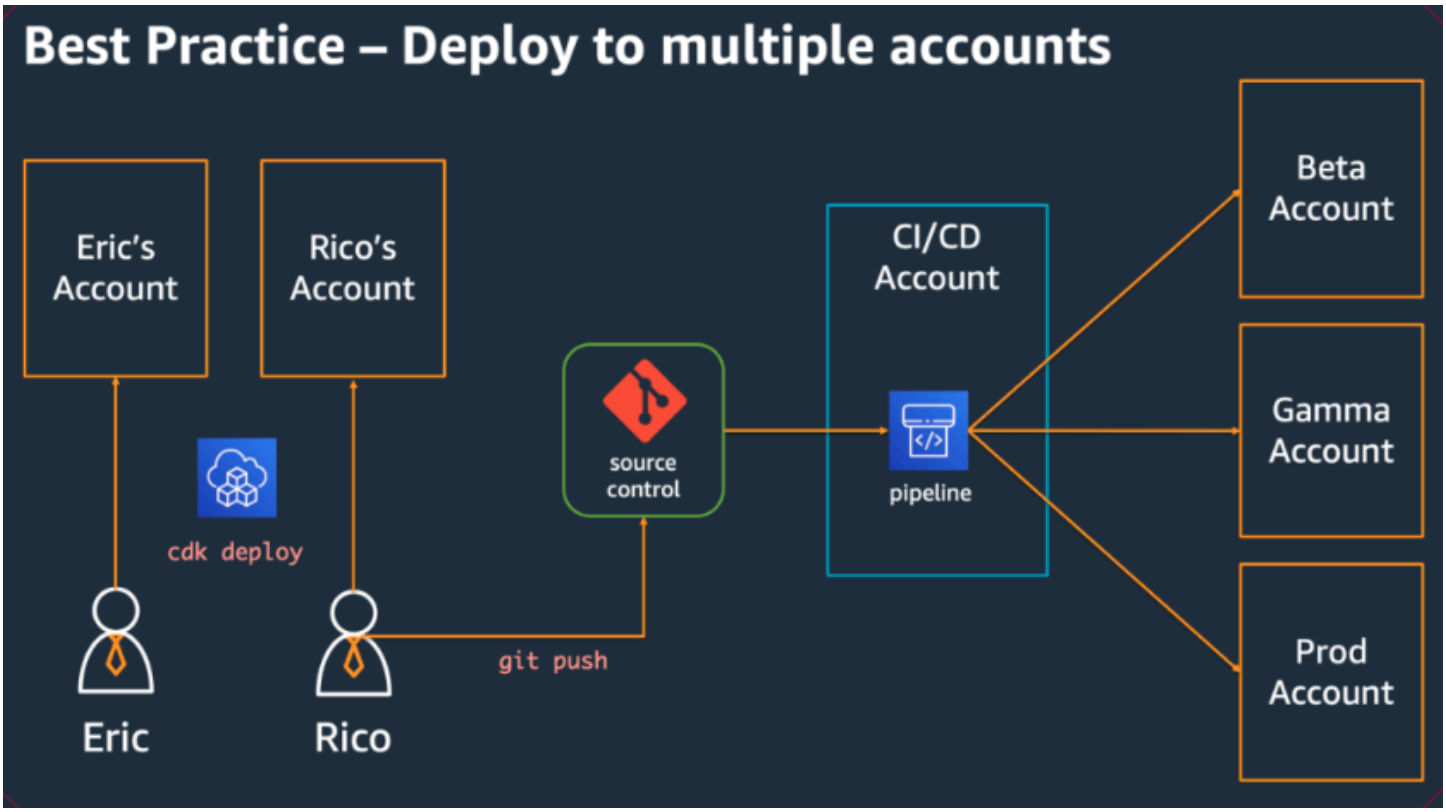
Pada tahap awal AWS CDK adopsi, penting untuk mempertimbangkan bagaimana mengatur organisasi Anda untuk sukses. Ini adalah praktik terbaik untuk memiliki tim ahli yang bertanggung jawab untuk melatih dan membimbing seluruh perusahaan saat mereka mengadopsi CDK. Ukuran tim ini mungkin bervariasi, dari satu atau dua orang di perusahaan kecil hingga Cloud Center of Excellence (CCoE) yang lengkap di perusahaan yang lebih besar. Tim ini bertanggung jawab untuk menetapkan standar dan kebijakan untuk infrastruktur cloud di perusahaan Anda, dan juga untuk pelatihan dan pendampingan pengembang.

CCoE mungkin memberikan panduan tentang bahasa pemrograman apa yang harus digunakan untuk infrastruktur cloud. Detail akan bervariasi dari satu organisasi ke organisasi berikutnya, tetapi kebijakan yang baik membantu memastikan bahwa pengembang dapat memahami dan memelihara infrastruktur cloud perusahaan.

CCoE juga menciptakan “landing zone” yang mendefinisikan unit organisasi Anda di dalamnya. AWS Landing zone adalah AWS lingkungan multi-akun yang telah dikonfigurasi sebelumnya, aman, dapat diskalakan, berdasarkan cetak biru praktik terbaik. Untuk menyatukan layanan yang membentuk landing zone Anda, Anda dapat menggunakan AWS Control Tower, yang mengonfigurasi dan mengelola seluruh sistem multi-akun Anda dari satu antarmuka pengguna.

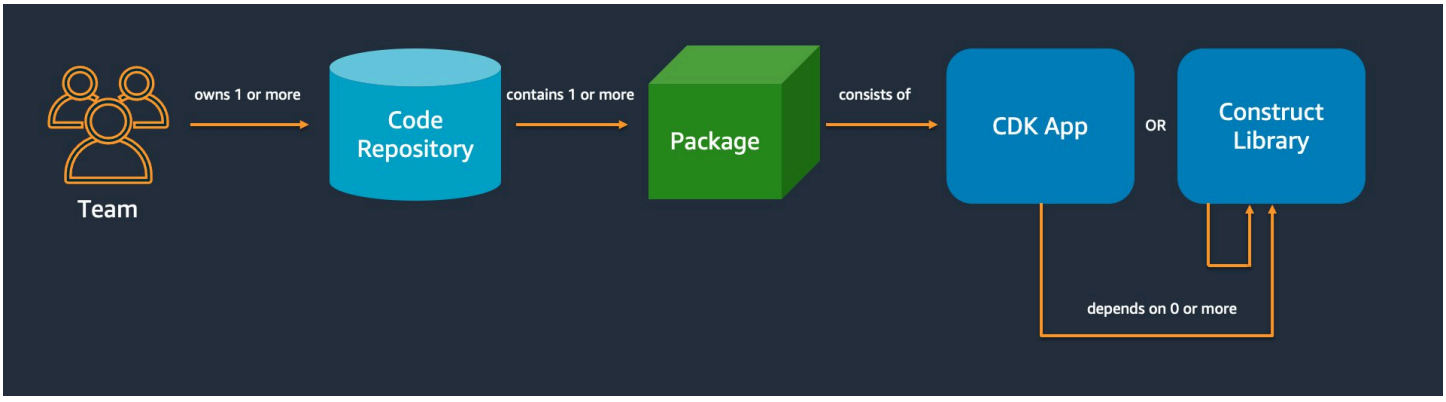
Tim pengembangan harus dapat menggunakan akun mereka sendiri untuk menguji dan menyebarkan sumber daya baru di akun ini sesuai kebutuhan. Pengembang individu dapat

memperlakukan sumber daya ini sebagai ekstensi dari workstation pengembangan mereka sendiri. Menggunakan CDK Pipelines, AWS CDK aplikasi kemudian dapat digunakan melalui akun CI/CD untuk pengujian, integrasi, dan lingkungan produksi (masing-masing diisolasi di Wilayah atau akunnya sendiri). AWS Ini dilakukan dengan menggabungkan kode pengembang ke dalam repositori kanonik organisasi Anda.



Praktik terbaik pengkodean

Bagian ini menyajikan praktik terbaik untuk mengatur AWS CDK kode Anda. Diagram berikut menunjukkan hubungan antara tim dan repositori kode tim, paket, aplikasi, dan pustaka konstruksi.



Mulai sederhana dan tambahkan kompleksitas hanya ketika Anda membutuhkannya

Prinsip panduan untuk sebagian besar praktik terbaik kami adalah menjaga hal-hal sesederhana mungkin—tetapi tidak lebih sederhana. Tambahkan kompleksitas hanya ketika kebutuhan Anda menentukan solusi yang lebih rumit. Dengan AWS CDK, Anda dapat memfaktorkan ulang kode Anda seperlunya untuk mendukung persyaratan baru. Anda tidak perlu merancang untuk semua skenario yang mungkin di muka.

Sejajarkan dengan Kerangka AWS Well-Architected

AWS Well-Architected Framework mendefinisikan komponen sebagai kode, konfigurasi, AWS dan sumber daya yang bersama-sama memenuhi persyaratan. Komponen sering merupakan unit kepemilikan teknis, dan dipisahkan dari komponen lain. Istilah beban kerja digunakan untuk mengidentifikasi serangkaian komponen yang bersama-sama memberikan nilai bisnis. Beban kerja biasanya merupakan tingkat detail yang dikomunikasikan oleh para pemimpin bisnis dan teknologi.

AWS CDK Aplikasi memetakan ke komponen seperti yang didefinisikan oleh AWS Well-Architected Framework. AWS CDK aplikasi adalah mekanisme untuk mengkodifikasi dan memberikan praktik terbaik aplikasi cloud yang Dirancang dengan Baik. Anda juga dapat membuat dan berbagi komponen sebagai pustaka kode yang dapat digunakan kembali melalui repositori artefak, seperti. AWS CodeArtifact

Setiap aplikasi dimulai dengan satu paket dalam satu repositori

Satu paket adalah titik masuk AWS CDK aplikasi Anda. Di sini, Anda menentukan bagaimana dan di mana untuk menyebarkan unit logis yang berbeda dari aplikasi Anda. Anda juga menentukan pipeline CI/CD untuk menyebarkan aplikasi. Konstruksi aplikasi menentukan unit logis dari solusi Anda.

Gunakan paket tambahan untuk konstruksi yang Anda gunakan di lebih dari satu aplikasi. (Konstruksi bersama juga harus memiliki siklus hidup dan strategi pengujian sendiri.) Dependensi antar paket dalam repositori yang sama dikelola oleh perkakas build repo Anda.

Meskipun mungkin, kami tidak menyarankan untuk menempatkan beberapa aplikasi di repositori yang sama, terutama saat menggunakan pipeline penerapan otomatis. Melakukan hal ini meningkatkan “radius ledakan” perubahan selama penerapan. Ketika ada beberapa aplikasi dalam repositori, perubahan pada satu aplikasi memicu penerapan yang lain (bahkan jika yang lain tidak berubah). Selain itu, jeda dalam satu aplikasi mencegah aplikasi lain dikerahkan.

Pindahkan kode ke repositori berdasarkan siklus hidup kode atau kepemilikan tim

Ketika paket mulai digunakan dalam beberapa aplikasi, pindahkan ke repositori mereka sendiri. Dengan cara ini, paket dapat direferensikan oleh sistem build aplikasi yang menggunakannya, dan mereka juga dapat diperbarui pada irama independen dari siklus hidup aplikasi. Namun, pada awalnya mungkin masuk akal untuk meletakkan semua konstruksi bersama dalam satu repositori.

Juga, pindahkan paket ke repositori mereka sendiri ketika tim yang berbeda sedang mengerjakannya. Ini membantu menegakkan kontrol akses.

Untuk menggunakan paket di seluruh batas repositori, Anda memerlukan repositori paket pribadi —mirip dengan NPM,, atau Maven Central PyPi, tetapi internal organisasi Anda. Anda juga memerlukan proses rilis yang membangun, menguji, dan menerbitkan paket ke repositori paket pribadi. CodeArtifactdapat meng-host paket untuk sebagian besar bahasa pemrograman populer.

Dependensi pada paket dalam repositori paket dikelola oleh manajer paket bahasa Anda, seperti NPM untuk atau aplikasi. TypeScript JavaScript Manajer paket Anda membantu memastikan bahwa build dapat diulang. Ini dilakukan dengan merekam versi spesifik dari setiap paket yang bergantung pada aplikasi Anda. Ini juga memungkinkan Anda memutakhirkan dependensi tersebut secara terkontrol.

Paket bersama membutuhkan strategi pengujian yang berbeda. Untuk satu aplikasi, mungkin cukup baik untuk menyebarkan aplikasi ke lingkungan pengujian dan mengonfirmasi bahwa itu masih berfungsi. Tetapi paket bersama harus diuji secara independen dari aplikasi yang dikonsumsi, seolah-olah mereka dirilis ke publik. (Organisasi Anda mungkin memilih untuk benar-benar merilis beberapa paket bersama ke publik.)

Perlu diingat bahwa konstruksi bisa sewenang-wenang sederhana atau kompleks. A `Bucket` adalah konstruksi, tetapi `CameraShopWebsite` bisa menjadi konstruksi juga.

Infrastruktur dan kode runtime hidup dalam paket yang sama

Selain menghasilkan AWS CloudFormation template untuk menerapkan infrastruktur, mereka AWS CDK juga menggabungkan aset runtime seperti fungsi Lambda dan gambar Docker dan menerapkannya di samping infrastruktur Anda. Ini memungkinkan untuk menggabungkan kode yang mendefinisikan infrastruktur Anda dan kode yang mengimplementasikan logika runtime Anda ke dalam satu konstruksi. Ini adalah praktik terbaik untuk melakukan ini. Kedua jenis kode ini tidak perlu tinggal di repositori terpisah atau bahkan dalam paket terpisah.

Untuk mengembangkan dua jenis kode bersama-sama, Anda dapat menggunakan konstruksi mandiri yang sepenuhnya menggambarkan sepotong fungsionalitas, termasuk infrastruktur dan logikanya. Dengan konstruksi mandiri, Anda dapat menguji dua jenis kode secara terpisah, berbagi dan menggunakan kembali kode di seluruh proyek, dan versi semua kode yang disinkronkan.

Membangun praktik terbaik

Bagian ini berisi praktik terbaik untuk mengembangkan konstruksi. Konstruksi adalah modul yang dapat digunakan kembali dan dapat dikomposisi yang merangkum sumber daya. Mereka adalah blok bangunan AWS CDK aplikasi.

Model dengan konstruksi, gunakan dengan tumpukan

Tumpukan adalah unit penerapan: semua yang ada di tumpukan dikerahkan bersama. Jadi saat membangun unit logis tingkat tinggi aplikasi Anda dari beberapa AWS sumber daya, wakili setiap unit logis sebagai `Construct`, bukan sebagai. `Stack` Gunakan tumpukan hanya untuk menjelaskan bagaimana konstruksi Anda harus disusun dan dihubungkan untuk berbagai skenario penerapan Anda.

Misalnya, jika salah satu unit logis Anda adalah situs web, konstruksi yang menyusunnya (seperti bucket Amazon S3, API Gateway, fungsi Lambda, atau tabel Amazon RDS) harus disusun menjadi satu konstruksi tingkat tinggi. Kemudian konstruksi itu harus dipakai dalam satu atau lebih tumpukan untuk penerapan.

Dengan menggunakan konstruksi untuk bangunan dan tumpukan untuk digunakan, Anda meningkatkan potensi penggunaan kembali infrastruktur Anda dan memberi diri Anda lebih banyak fleksibilitas dalam cara penerapannya.

Konfigurasikan dengan properti dan metode, bukan variabel lingkungan

Pencarian variabel lingkungan di dalam konstruksi dan tumpukan adalah anti-pola yang umum. Baik konstruksi dan tumpukan harus menerima objek properti untuk memungkinkan konfigurasi penuh sepenuhnya dalam kode. Melakukan sebaliknya memperkenalkan ketergantungan pada mesin tempat kode akan berjalan, yang menciptakan lebih banyak informasi konfigurasi yang harus Anda lacak dan kelola.

Secara umum, pencarian variabel lingkungan harus dibatasi pada tingkat atas aplikasi. AWS CDK Mereka juga harus digunakan untuk menyampaikan informasi yang diperlukan untuk berjalan di lingkungan pengembangan. Untuk informasi selengkapnya, lihat the section called “Lingkungan”.

Unit menguji infrastruktur Anda

Untuk menjalankan rangkaian lengkap pengujian unit secara konsisten pada waktu pembuatan di semua lingkungan, hindari pencarian jaringan selama sintesis dan modelkan semua tahapan produksi Anda dalam kode. (Praktik terbaik ini dibahas nanti.) Jika ada komit tunggal yang selalu menghasilkan template yang dihasilkan sama, Anda dapat mempercayai pengujian unit yang Anda tulis untuk mengonfirmasi bahwa templat yang dihasilkan terlihat seperti yang Anda harapkan. Untuk informasi selengkapnya, lihat Menguji konstruksi.

Jangan ubah ID logis sumber daya stateful

Mengubah ID logis sumber daya menghasilkan sumber daya diganti dengan yang baru pada penerapan berikutnya. Untuk sumber daya stateful seperti database dan bucket S3, atau infrastruktur persisten seperti VPC Amazon, ini jarang yang Anda inginkan. Hati-hati tentang refactoring AWS CDK kode Anda yang dapat menyebabkan ID berubah. Tulis pengujian unit yang menyatakan bahwa ID logis dari sumber daya stateful Anda tetap statis. ID logis berasal dari yang `id` Anda tentukan saat Anda membuat instance konstruksi, dan posisi konstruksi di pohon konstruksi. Untuk informasi selengkapnya, lihat the section called “ID Logis”.

Konstruksi tidak cukup untuk kepatuhan

Banyak pelanggan perusahaan menulis pembungkus mereka sendiri untuk konstruksi L2 (konstruksi “dikurasi” yang mewakili AWS sumber daya individu dengan default waras bawaan dan praktik terbaik). Pembungkus ini menerapkan praktik terbaik keamanan seperti enkripsi statis dan kebijakan IAM tertentu. Misalnya, Anda dapat membuat `MyCompanyBucket` yang kemudian Anda gunakan dalam aplikasi Anda sebagai pengganti konstruksi Amazon S3 `Bucket` biasa. Pola ini berguna untuk memunculkan panduan keamanan di awal siklus hidup pengembangan perangkat lunak, tetapi jangan mengandalkannya sebagai satu-satunya sarana penegakan hukum.

Sebagai gantinya, gunakan AWS fitur seperti kebijakan kontrol layanan dan batasan izin untuk menegakkan pagar keamanan Anda di tingkat organisasi. Gunakan the section called “Aspek” atau alat seperti CloudFormation Guard untuk membuat pernyataan tentang properti keamanan elemen infrastruktur sebelum penerapan. Gunakan AWS CDK untuk apa yang terbaik.

Terakhir, perlu diingat bahwa menulis konstruksi “L2+” Anda sendiri dapat mencegah pengembang Anda memanfaatkan AWS CDK paket seperti Konstruksi AWS Solusi atau konstruksi pihak ketiga dari Construct Hub. Paket-paket ini biasanya dibangun di atas AWS CDK konstruksi standar dan tidak akan dapat menggunakan konstruksi pembungkus Anda.

Praktik terbaik aplikasi

Pada bagian ini kita membahas cara menulis AWS CDK aplikasi Anda, menggabungkan konstruksi untuk menentukan bagaimana AWS sumber daya Anda terhubung.

Buat keputusan pada waktu sintesis

Meskipun AWS CloudFormation memungkinkan Anda membuat keputusan pada waktu penerapan (menggunakan`Conditions`,`{ Fn::If }`, dan`Parameters`), dan AWS CDK memberi Anda beberapa akses ke mekanisme ini, sebaiknya jangan menggunakannya. Jenis nilai yang dapat Anda gunakan dan jenis operasi yang dapat Anda lakukan pada mereka terbatas dibandingkan dengan apa yang tersedia dalam bahasa pemrograman tujuan umum.

Sebagai gantinya, cobalah untuk membuat semua keputusan, seperti konstruksi mana yang akan dibuat, dalam AWS CDK aplikasi Anda dengan menggunakan `if` pernyataan bahasa pemrograman Anda dan fitur lainnya. Misalnya, idiom CDK umum, mengulangi daftar dan membuat instance konstruksi dengan nilai dari setiap item dalam daftar, tidak mungkin menggunakan ekspresi. AWS CloudFormation

Perlakukan AWS CloudFormation sebagai detail implementasi yang AWS CDK digunakan untuk penerapan cloud yang kuat, bukan sebagai target bahasa. Anda tidak menulis AWS CloudFormation template dalam TypeScript atau Python, Anda menulis kode CDK yang kebetulan digunakan CloudFormation untuk penerapan.

Gunakan nama sumber daya yang dihasilkan, bukan nama fisik

Nama adalah sumber daya yang berharga. Setiap nama hanya dapat digunakan satu kali. Oleh karena itu, jika Anda membuat hardcode nama tabel atau nama bucket ke dalam infrastruktur dan aplikasi Anda, Anda tidak dapat menerapkan infrastruktur itu dua kali di akun yang sama. (Nama yang kita bicarakan di sini adalah nama yang ditentukan oleh, misalnya, `bucketName` properti pada konstruksi bucket Amazon S3.)

Yang lebih buruk, Anda tidak dapat membuat perubahan pada sumber daya yang mengharuskannya untuk diganti. Jika properti hanya dapat disetel pada pembuatan sumber daya, seperti tabel Amazon DynamoDB, maka properti tersebut tidak dapat diubah. `KeySchema` Mengubah properti ini membutuhkan sumber daya baru. Namun, sumber daya baru harus memiliki nama yang sama untuk menjadi pengganti yang benar. Tetapi tidak dapat memiliki nama yang sama sementara sumber daya yang ada masih menggunakan nama itu.

Pendekatan yang lebih baik adalah menentukan sesedikit mungkin nama. Jika Anda menghilangkan nama sumber daya, AWS CDK akan menghasilkannya untuk Anda dengan cara yang tidak akan menimbulkan masalah. Misalkan Anda memiliki tabel sebagai sumber daya. Anda kemudian dapat meneruskan nama tabel yang dihasilkan sebagai variabel lingkungan ke dalam AWS Lambda fungsi Anda. Dalam AWS CDK aplikasi Anda, Anda dapat mereferensikan nama tabel sebagai`table.tableName`. Atau, Anda dapat membuat file konfigurasi pada instans Amazon EC2 Anda saat startup, atau menulis nama tabel yang sebenarnya ke AWS Systems Manager Parameter Store sehingga aplikasi Anda dapat membacanya dari sana.

Jika tempat yang Anda butuhkan adalah AWS CDK tumpukan lain, itu bahkan lebih mudah. Misalkan satu tumpukan mendefinisikan sumber daya dan tumpukan lain perlu menggunakannya, hal berikut ini berlaku:

- Jika kedua tumpukan berada di AWS CDK aplikasi yang sama, berikan referensi di antara dua tumpukan. Misalnya, simpan referensi ke konstruksi sumber daya sebagai atribut dari stack yang menentukan ()`this.stack.uploadBucket = myBucket`. Kemudian, berikan atribut itu ke konstruktor tumpukan yang membutuhkan sumber daya.
- Ketika dua tumpukan berada di AWS CDK aplikasi yang berbeda, gunakan `from` metode statis untuk menggunakan sumber daya yang ditentukan secara eksternal berdasarkan ARN, nama, atau atribut lainnya. (Misalnya, gunakan `Table.fromArn()` untuk tabel DynamoDB). Gunakan `CfnOutput` konstruksi untuk mencetak ARN atau nilai lain yang diperlukan dalam output`cdk deploy`, atau lihat di. AWS Management Console Atau, aplikasi kedua dapat membaca CloudFormation template yang dihasilkan oleh aplikasi pertama dan mengambil nilai itu dari `Outputs` bagian tersebut.

Menentukan kebijakan penghapusan dan penyimpanan log

AWS CDK Upaya untuk mencegah Anda kehilangan data dengan default ke kebijakan yang mempertahankan semua yang Anda buat. Misalnya, kebijakan penghapusan default pada sumber daya yang berisi data (seperti bucket Amazon S3 dan tabel database) tidak menghapus sumber daya saat dihapus dari tumpukan. Sebaliknya, sumber daya menjadi yatim piatu dari tumpukan. Demikian pula, default CDK adalah menyimpan semua log selamanya. Dalam lingkungan produksi, default ini dapat dengan cepat menghasilkan penyimpanan sejumlah besar data yang sebenarnya tidak Anda butuhkan, dan tagihan yang sesuai. AWS

Pertimbangkan dengan cermat apa yang Anda inginkan kebijakan ini untuk setiap sumber daya produksi dan tentukan sesuai dengan itu. Gunakan the section called “Aspek” untuk memvalidasi kebijakan penghapusan dan pencatatan di tumpukan Anda.

Pisahkan aplikasi Anda menjadi beberapa tumpukan seperti yang ditentukan oleh persyaratan penerapan

Tidak ada aturan keras dan cepat untuk berapa banyak tumpukan yang dibutuhkan aplikasi Anda. Anda biasanya akan mendasarkan keputusan pada pola penerapan Anda. Ingatlah pedoman berikut:

- Biasanya lebih mudah untuk menyimpan sebanyak mungkin sumber daya dalam tumpukan yang sama, jadi simpan bersama kecuali Anda tahu Anda ingin mereka dipisahkan.
- Pertimbangkan untuk menyimpan sumber daya stateful (seperti database) dalam tumpukan terpisah dari sumber daya stateless. Anda kemudian dapat mengaktifkan perlindungan terminasi pada tumpukan stateful. Dengan cara ini, Anda dapat dengan bebas menghancurkan atau membuat banyak salinan tumpukan stateless tanpa risiko kehilangan data.
- Sumber daya stateful lebih sensitif terhadap penggantian nama konstruksi—mengganti nama mengarah ke penggantian sumber daya. Oleh karena itu, jangan sarang sumber daya stateful di dalam konstruksi yang kemungkinan akan dipindahkan atau diganti namanya (kecuali status dapat dibangun kembali jika hilang, seperti cache). Ini adalah alasan bagus lainnya untuk menempatkan sumber daya stateful di tumpukan mereka sendiri.

Berkomitmen **`cdk.context.json`** untuk menghindari perilaku non-deterministik

Determinisme adalah kunci keberhasilan AWS CDK penerapan. AWS CDK Aplikasi pada dasarnya harus memiliki hasil yang sama setiap kali diterapkan ke lingkungan tertentu.

Karena AWS CDK aplikasi Anda ditulis dalam bahasa pemrograman tujuan umum, aplikasi dapat mengeksekusi kode arbitrer, menggunakan pustaka arbitrer, dan membuat panggilan jaringan arbitrer. Misalnya, Anda dapat menggunakan AWS SDK untuk mengambil beberapa informasi dari AWS akun Anda saat mensintesis aplikasi Anda. Ketahuilah bahwa melakukan hal itu akan menghasilkan persyaratan pengaturan kredensyal tambahan, peningkatan latensi, dan kemungkinan, betapapun kecilnya, kegagalan setiap kali Anda menjalankan. `cdk synth`

Jangan pernah memodifikasi AWS akun atau sumber daya Anda selama sintesis. Mensintesis aplikasi seharusnya tidak memiliki efek samping. Perubahan pada infrastruktur Anda harus terjadi

hanya dalam fase penerapan, setelah AWS CloudFormation template dibuat. Dengan cara ini, jika ada masalah, secara otomatis AWS CloudFormation dapat memutar kembali perubahan. Untuk membuat perubahan yang tidak dapat dengan mudah dibuat dalam AWS CDK kerangka kerja, gunakan sumber daya khusus untuk mengeksekusi kode arbitrer pada waktu penerapan.

Bahkan panggilan hanya-baca yang ketat belum tentu aman. Pertimbangkan apa yang terjadi jika nilai yang dikembalikan oleh panggilan jaringan berubah. Bagian apa dari infrastruktur Anda yang akan berdampak? Apa yang akan terjadi pada sumber daya yang sudah digunakan? Berikut adalah dua contoh situasi di mana perubahan mendadak dalam nilai dapat menyebabkan masalah.

- Jika Anda menyediakan VPC Amazon ke semua Availability Zone yang tersedia di Wilayah tertentu, dan jumlah AZ adalah dua pada hari penerapan, maka ruang IP Anda akan dibagi dua. Jika AWS meluncurkan Availability Zone baru pada hari berikutnya, penerapan berikutnya setelah itu mencoba membagi ruang IP Anda menjadi tiga, mengharuskan semua subnet dibuat ulang. Ini mungkin tidak akan mungkin karena instans Amazon EC2 Anda masih berjalan, dan Anda harus membersihkannya secara manual.
- Jika Anda menanyakan image mesin Amazon Linux terbaru dan menerapkan instans Amazon EC2, dan hari berikutnya gambar baru dirilis, penerapan berikutnya akan mengambil AMI baru dan menggantikan semua instans Anda. Ini mungkin bukan apa yang Anda harapkan terjadi.

Situasi ini dapat merusak karena perubahan AWS sisi mungkin terjadi setelah berbulan-bulan atau bertahun-tahun penerapan yang berhasil. Tiba-tiba penerapan Anda gagal “tanpa alasan” dan Anda sudah lama lupa apa yang Anda lakukan dan mengapa.

Untungnya, AWS CDK termasuk mekanisme yang disebut penyedia konteks untuk merekam snapshot nilai non-deterministik. Hal ini memungkinkan operasi sintesis future untuk menghasilkan template yang persis sama seperti yang mereka lakukan saat pertama kali digunakan. Satu-satunya perubahan dalam template baru adalah perubahan yang Anda buat dalam kode Anda. Saat Anda menggunakan `.fromLookup()` metode konstruksi, hasil panggilan di-cache. `cdk.context.json` Anda harus melakukan ini ke kontrol versi bersama dengan sisa kode Anda untuk memastikan bahwa eksekusi aplikasi CDK Anda di masa mendatang menggunakan nilai yang sama. CDK Toolkit menyertakan perintah untuk mengelola cache konteks, sehingga Anda dapat menyegarkan entri tertentu saat diperlukan. Untuk informasi selengkapnya, lihat the section called “Konteks”.

Jika Anda memerlukan beberapa nilai (dari AWS atau di tempat lain) yang tidak ada penyedia konteks CDK asli, kami sarankan untuk menulis skrip terpisah. Skrip harus mengambil nilai dan menulisnya ke file, lalu membaca file itu di aplikasi CDK Anda. Jalankan skrip hanya jika Anda ingin menyegarkan nilai yang disimpan, bukan sebagai bagian dari proses pembuatan reguler Anda.

Biarkan AWS CDK mengelola peran dan kelompok keamanan

Dengan metode `grant()` kenyamanan pustaka konstruksi AWS CDK, Anda dapat membuat AWS Identity and Access Management peran yang memberikan akses ke satu sumber daya oleh sumber daya lainnya menggunakan izin cakupan minimal. Misalnya, pertimbangkan baris seperti berikut ini:

```
myBucket.grantRead(myLambda)
```

Baris tunggal ini menambahkan kebijakan ke peran fungsi Lambda (yang juga dibuat untuk Anda). Peran itu dan kebijakannya lebih dari selusin baris CloudFormation yang tidak perlu Anda tulis. Hanya AWS CDK memberikan izin minimal yang diperlukan agar fungsi dapat dibaca dari bucket.

Jika Anda mengharuskan pengembang untuk selalu menggunakan peran yang telah ditentukan sebelumnya yang dibuat oleh tim keamanan, AWS CDK pengkodean menjadi jauh lebih rumit. Tim Anda bisa kehilangan banyak fleksibilitas dalam cara mereka merancang aplikasi mereka. Alternatif yang lebih baik adalah menggunakan kebijakan kontrol layanan dan batas izin untuk memastikan bahwa pengembang tetap berada di dalam pagar pembatas.

Model semua tahapan produksi dalam kode

Dalam AWS CloudFormation skenario tradisional, tujuan Anda adalah menghasilkan artefak tunggal yang diparameterisasi sehingga dapat digunakan ke berbagai lingkungan target setelah menerapkan nilai konfigurasi khusus untuk lingkungan tersebut. Di CDK, Anda dapat, dan harus, membangun konfigurasi itu ke dalam kode sumber Anda. Buat tumpukan untuk lingkungan produksi Anda, dan buat tumpukan terpisah untuk setiap tahapan Anda yang lain. Kemudian, letakkan nilai konfigurasi untuk setiap tumpukan dalam kode. Gunakan layanan seperti Secrets Manager dan Systems Manager Parameter Store untuk nilai sensitif yang tidak ingin Anda periksa ke kontrol sumber, menggunakan nama atau ARN sumber daya tersebut.

Saat Anda mensintesis aplikasi Anda, rakitan cloud yang dibuat di `cdk.out` folder berisi templat terpisah untuk setiap lingkungan. Seluruh bangunan Anda bersifat deterministik. Tidak ada out-of-band perubahan pada aplikasi Anda, dan setiap komit yang diberikan selalu menghasilkan AWS CloudFormation template yang sama persis dan aset yang menyertainya. Ini membuat pengujian unit jauh lebih andal.

Ukur semuanya

Mencapai tujuan penyebaran berkelanjutan penuh, tanpa campur tangan manusia, membutuhkan otomatisasi tingkat tinggi. Otomatisasi itu hanya dimungkinkan dengan pemantauan dalam jumlah

ekstensif. Untuk mengukur semua aspek sumber daya yang Anda gunakan, buat metrik, alarm, dan dasbor. Jangan berhenti mengukur hal-hal seperti penggunaan CPU dan ruang disk. Juga catat metrik bisnis Anda, dan gunakan pengukuran tersebut untuk mengotomatiskan keputusan penerapan seperti rollback. Sebagian besar konstruksi L2 AWS CDK memiliki metode kenyamanan untuk membantu Anda membuat metrik, seperti `metricUserErrors()` metode pada kelas DynamoDB.table.

AWS CDK referensi

Bagian ini berisi informasi referensi untuk AWS Cloud Development Kit (AWS CDK).

Topik

- Referensi API
- AWS CDK pembuatan versi

Referensi API

Referensi API berisi informasi tentang AWS Construct Library dan API lain yang disediakan oleh. AWS Cloud Development Kit (AWS CDK) Sebagian besar AWS Construct Library terkandung dalam satu paket yang disebut dengan TypeScript namanya:`aws-cdk-lib`. Nama paket sebenarnya bervariasi menurut bahasa. Versi terpisah dari referensi API disediakan untuk setiap bahasa pemrograman yang didukung.

Referensi CDK API diatur ke dalam sub-modul. Ada satu atau lebih sub-modul untuk masing-masing Layanan AWS.

Setiap sub-modul memiliki ikhtisar yang mencakup informasi tentang cara menggunakan API-nya. Misalnya, ikhtisar S3 menunjukkan cara menyetel enkripsi default pada bucket Amazon Simple Storage Service (Amazon S3).

AWS CDK pembuatan versi

Topik ini memberikan informasi referensi tentang cara AWS Cloud Development Kit (AWS CDK) menangani pembuatan versi.

Nomor versi terdiri dari tiga bagian versi numerik: mayor. kecil. patch, dan benar-benar mematuhi model versi semantik. Ini berarti bahwa melanggar perubahan pada API stabil terbatas pada rilis utama.

Rilis minor dan patch kompatibel ke belakang. Kode yang ditulis dalam versi sebelumnya dengan versi utama yang sama dapat ditingkatkan ke versi yang lebih baru dalam versi utama yang sama. Ini juga akan terus membangun dan menjalankan, menghasilkan output yang sama.

Topik

- AWS CDKCLIkompatibilitas
- AWS Membangun versi Perpustakaan
- Stabilitas pengikatan bahasa

AWS CDKCLIkompatibilitas

AWS CDK CLIItu selalu kompatibel dengan pustaka konstruksi dengan nomor versi yang lebih rendah atau sama secara semantik. Oleh karena itu, selalu aman untuk meningkatkan AWS CDK CLI versi utama yang sama.

AWS CDK CLIIni tidak selalu kompatibel dengan pustaka konstruksi dari versi yang lebih tinggi secara semantik. Kompatibilitas tergantung pada apakah versi skema perakitan cloud yang sama digunakan oleh dua komponen. AWS CDK Kerangka kerja menghasilkan perakitan cloud selama sintesis dan AWS CDK CLI mengkonsumsinya untuk penerapan. Skema yang mendefinisikan format perakitan cloud ditentukan dan berversi secara ketat.

AWS pustaka konstruksi menggunakan versi skema perakitan cloud tertentu kompatibel dengan versi yang menggunakan AWS CDK CLI versi skema tersebut atau yang lebih baru. Ini mungkin termasuk rilis AWS CDK CLI yang lebih awal dari rilis pustaka konstruksi tertentu.

Jika versi perakitan cloud yang diperlukan oleh pustaka konstruksi tidak kompatibel dengan versi yang didukung oleh AWS CDK CLI, Anda menerima pesan kesalahan seperti berikut:

```
Cloud assembly schema version mismatch: Maximum schema version supported is 3.0.0, but
 found 4.0.0.
    Please upgrade your CLI in order to interact with this app.
```

Untuk mengatasi kesalahan ini, perbarui AWS CDK CLI ke versi yang kompatibel dengan versi perakitan cloud yang diperlukan, atau ke versi terbaru yang tersedia. Alternatifnya (menurunkan modul library build yang digunakan aplikasi Anda) umumnya tidak disarankan.

> **Note**
>
> Untuk detail selengkapnya tentang skema perakitan cloud, lihat Pembuatan Versi Cloud Assembly.

AWS Membangun versi Perpustakaan

Modul-modul di AWS Construct Library bergerak melalui berbagai tahap saat dikembangkan dari konsep ke API yang matang. Tahapan yang berbeda menawarkan berbagai tingkat stabilitas API di versi berikutnya AWS CDK.

API di AWS CDK pustaka utama,`aws-cdk-lib`, stabil, dan pustaka sepenuhnya berversi semantik. Paket ini mencakup konstruksi AWS CloudFormation (L1) untuk semua AWS layanan dan semua modul tingkat tinggi (L2 dan L3) yang stabil. (Ini juga termasuk kelas CDK inti seperti `App` dan`Stack`). API tidak akan dihapus dari paket ini (meskipun mungkin tidak digunakan lagi) hingga rilis utama CDK berikutnya. Tidak ada API individual yang akan mengalami perubahan yang melanggar. Ketika perubahan yang melanggar diperlukan, API yang sama sekali baru akan ditambahkan.

API baru yang sedang dikembangkan untuk layanan yang sudah `aws-cdk-lib` dimasukkan diidentifikasi menggunakan `Beta`*`N`* akhiran, di mana `N` dimulai dari 1 dan ditambah dengan setiap perubahan yang melanggar ke API baru. `Beta`*`N`* API tidak pernah dihapus, hanya tidak digunakan lagi, sehingga aplikasi Anda yang ada terus bekerja dengan versi yang lebih baru. `aws-cdk-lib` Saat API dianggap stabil, API baru tanpa `Beta`*`N`* akhiran ditambahkan.

Ketika API tingkat tinggi (L2 atau L3) mulai dikembangkan untuk AWS layanan yang sebelumnya hanya memiliki API L1, API tersebut awalnya didistribusikan dalam paket terpisah. Nama paket semacam itu memiliki akhiran “Alpha”, dan versinya cocok dengan versi pertama yang `aws-cdk-lib` kompatibel dengan, dengan `alpha` sub-versi. Ketika modul mendukung kasus penggunaan yang dimaksudkan, API-nya ditambahkan ke`aws-cdk-lib`.

Stabilitas pengikatan bahasa

Seiring waktu, kami mungkin menambahkan dukungan untuk bahasa pemrograman tambahan. AWS CDK Meskipun API yang dijelaskan dalam semua bahasa adalah sama, cara API diekspresikan bervariasi menurut bahasa dan mungkin berubah seiring berkembangnya dukungan bahasa. Untuk alasan ini, binding bahasa dianggap eksperimental untuk sementara waktu sampai dianggap siap untuk digunakan produksi.

Language	Stability
TypeScript	Stable
JavaScript	Stable

Language	Stability
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK tutorial

Bagian ini berisi tutorial untuk AWS Cloud Development Kit (AWS CDK).

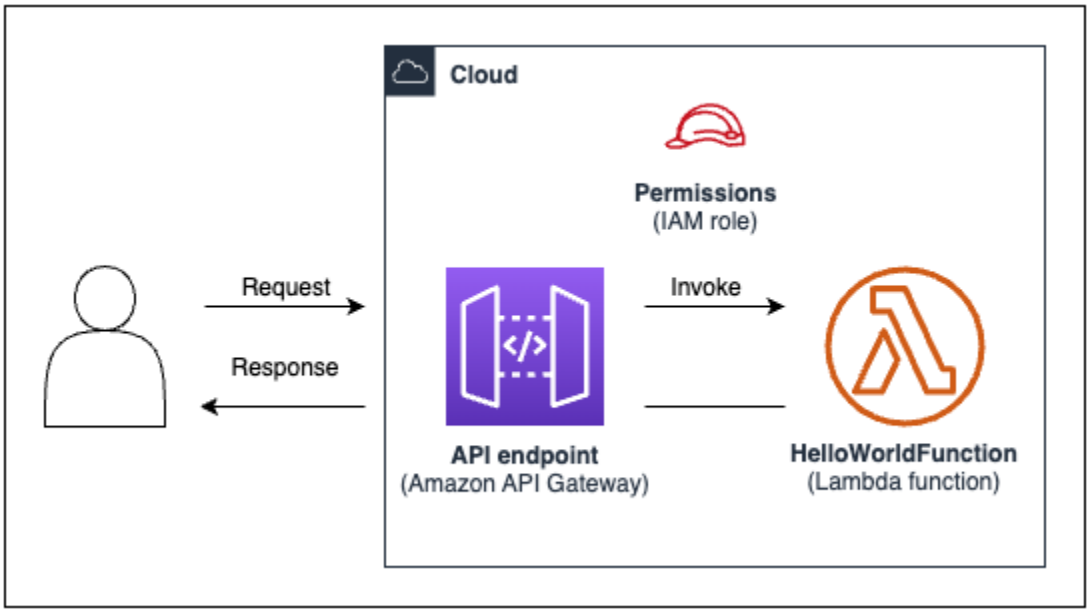
Topik

Buat aplikasi Hello World tanpa server

Dalam tutorial ini, Anda menggunakan AWS Cloud Development Kit (AWS CDK) untuk membuat Hello World aplikasi tanpa server sederhana yang mengimplementasikan backend API dasar dengan membuat yang berikut:

- Amazon API Gateway REST API - Menyediakan titik akhir HTTP yang digunakan untuk memanggil fungsi Anda melalui permintaan. HTTP GET
- AWS Lambda fungsi - Fungsi yang mengembalikan `Hello World!` pesan ketika dipanggil dengan titik HTTP akhir.
- Integrasi dan izin — Detail konfigurasi dan izin untuk sumber daya Anda untuk berinteraksi satu sama lain dan melakukan tindakan, seperti menulis log ke Amazon. CloudWatch

Diagram berikut menunjukkan komponen dari aplikasi ini:



Untuk tutorial ini, Anda akan menyelesaikan yang berikut:

1. Buat AWS CDK proyek.
2. Tentukan fungsi Lambda dan API REST API Gateway API menggunakan konstruksi L2 dari Construct Library. AWS
3. Terapkan aplikasi Anda ke file. AWS Cloud
4. Berinteraksi dengan aplikasi Anda di AWS Cloud.
5. Hapus aplikasi sampel dari file AWS Cloud.

Prasyarat

Sebelum memulai tutorial ini, selesaikan yang berikut ini:

- Membuat Akun AWS dan memiliki AWS Command Line Interface (AWS CLI) diinstal dan dikonfigurasi.
- Instal Node.js dannpm.
- Instal CDK Toolkit secara global, menggunakan. `npm install -g aws-cdk`

Untuk informasi selengkapnya, lihat Memulai dengan AWS CDK.

Kami juga merekomendasikan pemahaman dasar tentang hal-hal berikut:

- Apa itu AWS CDK?untuk pengantar dasar untuk AWS CDK.
- AWS CDK konsepuntuk ikhtisar konsep inti dari AWS CDK.

Langkah 1: Buat proyek CDK

Pada langkah ini, Anda membuat proyek CDK baru menggunakan AWS CDK CLI `cdk init` perintah.

Untuk membuat proyek CDK

1. Dari direktori awal pilihan Anda, buat dan navigasikan ke direktori proyek yang diberi nama `cdk-hello-world` di mesin Anda:

```
$ mkdir cdk-hello-world && cd cdk-hello-world
```

2. Gunakan `cdk init` perintah untuk membuat proyek baru dalam bahasa pemrograman pilihan Anda:

   TypeScript

   ```
   $ cdk init --language typescript
   ```

   Instal AWS CDK pustaka:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   JavaScript

   ```
   $ cdk init --language javascript
   ```

   Instal AWS CDK pustaka:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   Python

   ```
   $ cdk init --language python
   ```

   Aktifkan lingkungan virtual:

   ```
   $ source .venv/bin/activate
   ```

   Instal AWS CDK pustaka dan dependensi proyek:

   ```
   (.venv)$ python3 -m pip install -r requirements.txt
   ```

   Java

   ```
   $ cdk init --language java
   ```

   Instal AWS CDK pustaka dan dependensi proyek:

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

Instal AWS CDK pustaka dan dependensi proyek:

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

Instal dependensi proyek:

```
$ go mod tidy
```

CDK CLI membuat proyek dengan struktur berikut:

TypeScript

```
cdk-hello-world
### .git
### .gitignore
### .npmignore
### README.md
### bin
#   ### cdk-hello-world.ts
### cdk.json
### jest.config.js
### lib
#   ### cdk-hello-world-stack.ts
### node_modules
### package-lock.json
### package.json
### test
#   ### cdk-hello-world.test.ts
```

```
### tsconfig.json
```

JavaScript

```
cdk-hello-world
### .git
### .gitignore
### .npmignore
### README.md
### bin
#   ### cdk-hello-world.js
### cdk.json
### jest.config.js
### lib
#   ### cdk-hello-world-stack.js
### node_modules
### package-lock.json
### package.json
### test
    ### cdk-hello-world.test.js
```

Python

```
cdk-hello-world
### .git
### .gitignore
### .venv
### README.md
### app.py
### cdk.json
### cdk_hello_world
#   ### __init__.py
#   ### cdk_hello_world_stack.py
### requirements-dev.txt
### requirements.txt
### source.bat
### tests
```

Java

```
cdk-hello-world
### .git
```

```
### .gitignore
### README.md
### cdk.json
### pom.xml
### src
#   ### main
#   #   ### java
#   #       ### com
#   #           ### myorg
#   #               ### CdkHelloWorldApp.java
#   #               ### CdkHelloWorldStack.java
### target
```

C#

```
cdk-hello-world
### .git
### .gitignore
### README.md
### cdk.json
### src
    ### CdkHelloWorld
    #   ### CdkHelloWorld.csproj
    #   ### CdkHelloWorldStack.cs
    #   ### GlobalSuppressions.cs
    #   ### Program.cs
    ### CdkHelloWorld.sln
```

Go

```
cdk-hello-world
### .git
### .gitignore
### README.md
### cdk-hello-world.go
### cdk-hello-world_test.go
### cdk.json
### go.mod
```

CDK CLI secara otomatis membuat aplikasi CDK yang berisi satu tumpukan. Instance aplikasi CDK dibuat dari App kelas. Berikut ini adalah bagian dari file aplikasi CDK Anda:

TypeScript

Terletak di`bin/cdk-hello-world.ts`:

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { CdkHelloWorldStack } from '../lib/cdk-hello-world-stack';

const app = new cdk.App();
new CdkHelloWorldStack(app, 'CdkHelloWorldStack', {
});
```

JavaScript

Terletak di`bin/cdk-hello-world.js`:

```
#!/usr/bin/env node
const cdk = require('aws-cdk-lib');
const { CdkHelloWorldStack } = require('../lib/cdk-hello-world-stack');
const app = new cdk.App();
new CdkHelloWorldStack(app, 'CdkHelloWorldStack', {
});
```

Python

Terletak di`app.py`:

```
#!/usr/bin/env python3
import os
import aws_cdk as cdk
from cdk_hello_world.cdk_hello_world_stack import CdkHelloWorldStack

app = cdk.App()
CdkHelloWorldStack(app, "CdkHelloWorldStack",)
app.synth()
```

Java

Terletak di`src/main/java/.../CdkHelloWorldApp.java`:

```
package com.myorg;
```

```
import software.amazon.awscdk.App;
import software.amazon.awscdk.Environment;
import software.amazon.awscdk.StackProps;

import java.util.Arrays;

public class JavaApp {
    public static void main(final String[] args) {
        App app = new App();

        new JavaStack(app, "JavaStack", StackProps.builder()
                .build());

        app.synth();
    }
}
```

C#

Terletak di`src/CdkHelloWorld/Program.cs`:

```
using Amazon.CDK;
using System;
using System.Collections.Generic;
using System.Linq;

namespace CdkHelloWorld
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new CdkHelloWorldStack(app, "CdkHelloWorldStack", new StackProps
            {

            });
            app.Synth();
        }
    }
}
```

Go

Terletak di`cdk-hello-world.go`:

```
package main
import (
    "github.com/aws/aws-cdk-go/awscdk/v2"
    "github.com/aws/constructs-go/constructs/v10"
    "github.com/aws/jsii-runtime-go"
)

// ...

func main() {
    defer jsii.Close()
    app := awscdk.NewApp(nil)
    NewCdkHelloWorldStack(app, "CdkHelloWorldStack", &CdkHelloWorldStackProps{
        awscdk.StackProps{
            Env: env(),
        },
    })
    app.Synth(nil)
}

func env() *awscdk.Environment {
    return nil
}
```

Langkah 2: Buat fungsi Lambda Anda

Dalam proyek CDK Anda, buat `lambda` direktori yang menyertakan `hello.js` file baru. Berikut ini adalah contohnya:

TypeScript

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Berikut ini sekarang harus ditambahkan ke proyek CDK Anda:

```
cdk-hello-world
### lambda
    ### hello.js
```

JavaScript

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Berikut ini sekarang harus ditambahkan ke proyek CDK Anda:

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Berikut ini sekarang harus ditambahkan ke proyek CDK Anda:

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

Berikut ini sekarang harus ditambahkan ke proyek CDK Anda:

```
cdk-hello-world
```

```
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Berikut ini sekarang harus ditambahkan ke proyek CDK Anda:

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Berikut ini sekarang harus ditambahkan ke proyek CDK Anda:

```
cdk-hello-world
### lambda
    ### hello.js
```

> **Note**
>
> Untuk menjaga tutorial ini sederhana, kita menggunakan fungsi JavaScript Lambda untuk semua bahasa pemrograman CDK.

Tentukan fungsi Lambda Anda dengan menambahkan yang berikut ini ke file yang baru dibuat:

```
exports.handler = async (event) => {
    return {
        statusCode: 200,
        headers: { "Content-Type": "text/plain" },
        body: JSON.stringify({ message: "Hello, World!" }),
    };
};
```

Langkah 3: Tentukan konstruksi Anda

Pada langkah ini, Anda akan menentukan sumber daya Lambda dan API Gateway Anda menggunakan konstruksi AWS CDK L2.

Buka file proyek yang mendefinisikan tumpukan CDK Anda. Anda akan memodifikasi file ini untuk menentukan konstruksi Anda. Berikut ini adalah contoh file stack awal Anda:

TypeScript

Terletak di`lib/cdk-hello-world-stack.ts`:

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';

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

    // Your constructs will go here

  }
}
```

JavaScript

Terletak di`lib/cdk-hello-world-stack.js`:

```
const { Stack, Duration } = require('aws-cdk-lib');
const lambda = require('aws-cdk-lib/aws-lambda');
const apigateway = require('aws-cdk-lib/aws-apigateway');

class CdkHelloWorldStack extends Stack {
```

```
  constructor(scope, id, props) {
    super(scope, id, props);

    // Your constructs will go here

  }
}

module.exports = { CdkHelloWorldStack }
```

Python

Terletak dicdk_hello_world/cdk_hello_world_stack.py:

```
from aws_cdk import Stack
from constructs import Construct

class CdkHelloWorldStack(Stack):
    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

          // Your constructs will go here
```

Java

Terletak disrc/main/java/.../CdkHelloWorldStack.java:

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

public class CdkHelloWorldStack extends Stack {
    public CdkHelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public CdkHelloWorldStack(final Construct scope, final String id, final
 StackProps props) {
        super(scope, id, props);

        // Your constructs will go here
```

```
    }
}
```

C#

Terletak di`src/CdkHelloWorld/CdkHelloWorldStack.cs`:

```
using Amazon.CDK;
using Constructs;

namespace CdkHelloWorld
{
    public class CdkHelloWorldStack : Stack
    {
        internal CdkHelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            // Your constructs will go here
        }
    }
}
```

Go

Terletak di`cdk-hello-world.go`:

```
package main
import (
    "github.com/aws/aws-cdk-go/awscdk/v2"
    "github.com/aws/constructs-go/constructs/v10"
    "github.com/aws/jsii-runtime-go"
)
type CdkHelloWorldStackProps struct {
    awscdk.StackProps
}
func NewCdkHelloWorldStack(scope constructs.Construct, id string, props
 *CdkHelloWorldStackProps) awscdk.Stack {
    var sprops awscdk.StackProps
    if props != nil {
        sprops = props.StackProps
    }
    stack := awscdk.NewStack(scope, &id, &sprops)
    // Your constructs will go here
```

```
    return stack
}
func main() {
    // ...
}

func env() *awscdk.Environment {
    return nil
}
```

Dalam file ini, melakukan hal berikut: AWS CDK

- Instans tumpukan CDK Anda dipakai dari kelas. `Stack`
- Kelas `Constructs` dasar diimpor dan disediakan sebagai lingkup atau induk dari instance tumpukan Anda.

Tentukan sumber daya fungsi Lambda Anda

Untuk menentukan sumber daya fungsi Lambda, Anda mengimpor dan menggunakan konstruksi `aws-lambda` L2 dari Construct Library. AWS

Ubah file tumpukan Anda sebagai berikut:

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
// Import Lambda L2 construct
import * as lambda from 'aws-cdk-lib/aws-lambda';

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

    // Define the Lambda function resource
    const helloWorldFunction = new lambda.Function(this, 'HelloWorldFunction', {
      runtime: lambda.Runtime.NODEJS_20_X, // Choose any supported Node.js runtime
      code: lambda.Code.fromAsset('lambda'), // Points to the lambda directory
      handler: 'hello.handler', // Points to the 'hello' file in the lambda
 directory
    });
```

```
  }
}
```

JavaScript

```
const { Stack, Duration } = require('aws-cdk-lib');
// Import Lambda L2 construct
const lambda = require('aws-cdk-lib/aws-lambda');


class CdkHelloWorldStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // Define the Lambda function resource
    const helloWorldFunction = new lambda.Function(this, 'HelloWorldFunction', {
      runtime: lambda.Runtime.NODEJS_20_X, // Choose any supported Node.js runtime
      code: lambda.Code.fromAsset('lambda'), // Points to the lambda directory
      handler: 'hello.handler', // Points to the 'hello' file in the lambda
 directory
    });
  }
}

module.exports = { CdkHelloWorldStack }
```

Python

```
from aws_cdk import (
    Stack,
    # Import Lambda L2 construct
    aws_lambda as _lambda,
)
# ...

class CdkHelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Define the Lambda function resource
        hello_world_function = _lambda.Function(
            self,
```

```
            "HelloWorldFunction",
            runtime = _lambda.Runtime.NODEJS_20_X, # Choose any supported Node.js
 runtime
            code = _lambda.Code.from_asset("lambda"), # Points to the lambda
 directory
            handler = "hello.handler", # Points to the 'hello' file in the lambda
 directory
        )
```

> **Note**
>
> Kami mengimpor `aws_lambda` modul sebagai `_lambda` because `lambda` adalah pengenal bawaan di. Python

Java

```
// ...
// Import Lambda L2 construct
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;

public class CdkHelloWorldStack extends Stack {
    public CdkHelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public CdkHelloWorldStack(final Construct scope, final String id, final
 StackProps props) {
        super(scope, id, props);

        // Define the Lambda function resource
        Function helloWorldFunction = Function.Builder.create(this,
 "HelloWorldFunction")
                .runtime(Runtime.NODEJS_20_X)  // Choose any supported Node.js
 runtime
                .code(Code.fromAsset("src/main/resources/lambda")) // Points to the
 lambda directory
                .handler("hello.handler")  // Points to the 'hello' file in the
 lambda directory
                .build();
```

```
    }
}
```

C#

```
// ...
// Import Lambda L2 construct
using Amazon.CDK.AWS.Lambda;

namespace CdkHelloWorld
{
    public class CdkHelloWorldStack : Stack
    {
        internal CdkHelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            // Define the Lambda function resource
            var helloWorldFunction = new Function(this, "HelloWorldFunction", new
 FunctionProps
            {
                Runtime = Runtime.NODEJS_20_X, // Choose any supported Node.js
 runtime
                Code = Code.FromAsset("lambda"), // Points to the lambda directory
                Handler = "hello.handler" // Points to the 'hello' file in the
 lambda directory
            });
        }
    }
}
```

Go

```
package main

import (
    // ...
    // Import Lambda L2 construct
    "github.com/aws/aws-cdk-go/awscdk/v2/awslambda"
    // ...
)

// ...
```

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

    // Define the Lambda function resource
    helloWorldFunction := awslambda.NewFunction(stack,
 jsii.String("HelloWorldFunction"), &awslambda.FunctionProps{
        Runtime: awslambda.Runtime_NODEJS_20_X(), // Choose any supported Node.js
 runtime
        Code:    awslambda.Code_FromAsset(jsii.String("lambda")), // Points to the
 lambda directory
        Handler: jsii.String("hello"), // Points to the 'hello' file in the lambda
 directory
    })

    return stack
}

// ...
```

Di sini, Anda membuat sumber daya fungsi Lambda dan menentukan properti berikut:

- `runtime`— Lingkungan tempat fungsi berjalan. Di sini, kami menggunakan Node.js versi20.x.
- `code`— Jalur ke kode fungsi pada mesin lokal Anda.
- `handler`— Nama file tertentu yang berisi kode fungsi Anda.

Tentukan REST API sumber daya API Gateway

Untuk menentukan REST API resource API Gateway, Anda mengimpor dan menggunakan konstruksi `aws-apigateway` L2 dari Construct Library AWS .

Ubah file tumpukan Anda sebagai berikut:

TypeScript

```
// ...
//Import API Gateway L2 construct
```

```
import * as apigateway from 'aws-cdk-lib/aws-apigateway';


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

    // ...

    // Define the API Gateway resource
    const api = new apigateway.LambdaRestApi(this, 'HelloWorldApi', {
      handler: helloWorldFunction,
      proxy: false,
    });

    // Define the '/hello' resource with a GET method
    const helloResource = api.root.addResource('hello');
    helloResource.addMethod('GET');
  }
}
```

JavaScript

```
// ...
// Import API Gateway L2 construct
const apigateway = require('aws-cdk-lib/aws-apigateway');


class CdkHelloWorldStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // ...

    // Define the API Gateway resource
    const api = new apigateway.LambdaRestApi(this, 'HelloWorldApi', {
      handler: helloWorldFunction,
      proxy: false,
    });

    // Define the '/hello' resource with a GET method
    const helloResource = api.root.addResource('hello');
    helloResource.addMethod('GET');
```

```
  };
};

// ...
```

Python

```
from aws_cdk import (
    # ...
    # Import API Gateway L2 construct
    aws_apigateway as apigateway,
)
from constructs import Construct

class CdkHelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # ...

        # Define the API Gateway resource
        api = apigateway.LambdaRestApi(
            self,
            "HelloWorldApi",
            handler = hello_world_function,
            proxy = False,
        )

        # Define the '/hello' resource with a GET method
        hello_resource = api.root.add_resource("hello")
        hello_resource.add_method("GET")
```

Java

```
// ...
// Import API Gateway L2 construct
import software.amazon.awscdk.services.apigateway.LambdaRestApi;
import software.amazon.awscdk.services.apigateway.Resource;

public class CdkHelloWorldStack extends Stack {
    public CdkHelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
```

```
    }

    public CdkHelloWorldStack(final Construct scope, final String id, final
 StackProps props) {
        super(scope, id, props);

        // ...

        // Define the API Gateway resource
        LambdaRestApi api = LambdaRestApi.Builder.create(this, "HelloWorldApi")
                .handler(helloWorldFunction)
                .proxy(false) // Turn off default proxy integration
                .build();

        // Define the '/hello' resource and its GET method
        Resource helloResource = api.getRoot().addResource("hello");
        helloResource.addMethod("GET");
    }
}
```

C#

```
// ...
// Import API Gateway L2 construct
using Amazon.CDK.AWS.APIGateway;

namespace CdkHelloWorld
{
    public class CdkHelloWorldStack : Stack
    {
        internal CdkHelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
           // ...

            // Define the API Gateway resource
            var api = new LambdaRestApi(this, "HelloWorldApi", new
 LambdaRestApiProps
            {
                Handler = helloWorldFunction,
                Proxy = false
            });
```

```
            // Add a '/hello' resource with a GET method
            var helloResource = api.Root.AddResource("hello");
            helloResource.AddMethod("GET");
        }
    }
}
```

Go

```
// ...

import (
    // ...
    // Import Api Gateway L2 construct
    "github.com/aws/aws-cdk-go/awscdk/v2/awsapigateway"
    // ...
)

// ...

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

    // Define the Lambda function resource
    // ...

    // Define the API Gatweay resource
    api := awsapigateway.NewLambdaRestApi(stack, jsii.String("HelloWorldApi"),
 &awsapigateway.LambdaRestApiProps{
        Handler: helloWorldFunction,
        Proxy: jsii.Bool(false),
    })

    // Add a '/hello' resource with a GET method
    helloResource := api.Root().AddResource(jsii.String("hello"))
    helloResource.AddMethod(jsii.String("GET"))

    return stack
```

```
}

// ...
```

Di sini, Anda membuat REST API resource API Gateway, bersama dengan yang berikut ini:

- Integrasi antara fungsi REST API dan Lambda Anda, memungkinkan API untuk menjalankan fungsi Anda. Ini termasuk pembuatan sumber daya izin Lambda.
- Sumber daya atau jalur baru bernama `hello` yang ditambahkan ke root titik akhir API. Ini menciptakan titik akhir baru yang `/hello` menambah basis URL Anda.
- Metode GET untuk `hello` sumber daya. Ketika permintaan GET dikirim ke `/hello` titik akhir, fungsi Lambda dipanggil dan responsnya dikembalikan.

Langkah 4: Siapkan aplikasi Anda untuk penyebaran

Pada langkah ini Anda mempersiapkan aplikasi Anda untuk penyebaran dengan membangun, jika perlu, dan melakukan validasi dasar dengan perintah. AWS CDK CLI `cdk synth`

Jika perlu, buat aplikasi Anda:

TypeScript

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ npm run build
```

JavaScript

Bangunan tidak diperlukan.

Python

Bangunan tidak diperlukan.

Java

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ mvn package
```

C#

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ dotnet build src
```

Go

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ go build
```

Jalankan `cdk synth` untuk mensintesis AWS CloudFormation template dari kode CDK Anda. Dengan menggunakan konstruksi L2, banyak detail konfigurasi yang diperlukan oleh AWS CloudFormation untuk memfasilitasi interaksi antara fungsi Lambda Anda dan REST API disediakan untuk Anda oleh. AWS CDK

Dari akar proyek Anda, jalankan yang berikut ini:

```
$ cdk synth
```

> **Note**
>
> Jika Anda menerima kesalahan seperti berikut ini, verifikasi bahwa Anda berada di `cdk-hello-world` direktori dan coba lagi:
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

Jika berhasil, AWS CDK CLI akan menampilkan AWS CloudFormation template dalam YAML format pada prompt perintah. Template JSON yang diformat juga disimpan di `cdk.out` direktori.

Berikut ini adalah contoh output dari AWS CloudFormation template:

AWS CloudFormation Template

```
Resources:
  HelloWorldFunctionServiceRoleunique-identifier:
    Type: AWS::IAM::Role
```

```
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
        Version: "2012-10-17"
      ManagedPolicyArns:
        - Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/ServiceRole/Resource
  HelloWorldFunctionunique-identifier:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        S3Bucket:
          Fn::Sub: cdk-unique-identifier-assets-${AWS::AccountId}-${AWS::Region}
        S3Key: unique-identifier.zip
      Handler: hello.handler
      Role:
        Fn::GetAtt:
          - HelloWorldFunctionServiceRoleunique-identifier
          - Arn
      Runtime: nodejs20.x
    DependsOn:
      - HelloWorldFunctionServiceRoleunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/Resource
      aws:asset:path: asset.unique-identifier
      aws:asset:is-bundled: false
      aws:asset:property: Code
  HelloWorldApiunique-identifier:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: HelloWorldApi
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Resource
  HelloWorldApiDeploymentunique-identifier:
    Type: AWS::ApiGateway::Deployment
```

```
    Properties:
      Description: Automatically created by the RestApi construct
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    DependsOn:
      - HelloWorldApihelloGETunique-identifier
      - HelloWorldApihellounique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Deployment/Resource
  HelloWorldApiDeploymentStageprod012345ABC:
    Type: AWS::ApiGateway::Stage
    Properties:
      DeploymentId:
        Ref: HelloWorldApiDeploymentunique-identifier
      RestApiId:
        Ref: HelloWorldApiunique-identifier
      StageName: prod
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/DeploymentStage.prod/Resource
  HelloWorldApihellounique-identifier:
    Type: AWS::ApiGateway::Resource
    Properties:
      ParentId:
        Fn::GetAtt:
          - HelloWorldApiunique-identifier
          - RootResourceId
      PathPart: hello
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/Resource
  HelloWorldApihelloGETApiPermissionCdkHelloWorldStackHelloWorldApiunique-identifier:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunctionunique-identifier
          - Arn
      Principal: apigateway.amazonaws.com
      SourceArn:
        Fn::Join:
          - ""
          - - "arn:"
```

```
            - Ref: AWS::Partition
            - ":execute-api:"
            - Ref: AWS::Region
            - ":"
            - Ref: AWS::AccountId
            - ":"
            - Ref: HelloWorldApi9E278160
            - /
            - Ref: HelloWorldApiDeploymentStageprodunique-identifier
            - /GET/hello
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/
ApiPermission.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello
  HelloWorldApihelloGETApiPermissionTestCdkHelloWorldStackHelloWorldApiunique-
identifier:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunctionunique-identifier
          - Arn
      Principal: apigateway.amazonaws.com
      SourceArn:
        Fn::Join:
          - ""
          - - "arn:"
            - Ref: AWS::Partition
            - ":execute-api:"
            - Ref: AWS::Region
            - ":"
            - Ref: AWS::AccountId
            - ":"
            - Ref: HelloWorldApiunique-identifier
            - /test-invoke-stage/GET/hello
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/
ApiPermission.Test.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello
  HelloWorldApihelloGETunique-identifier:
    Type: AWS::ApiGateway::Method
    Properties:
      AuthorizationType: NONE
      HttpMethod: GET
      Integration:
```

```
        IntegrationHttpMethod: POST
        Type: AWS_PROXY
        Uri:
          Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - ":apigateway:"
              - Ref: AWS::Region
              - :lambda:path/2015-03-31/functions/
              - Fn::GetAtt:
                  - HelloWorldFunctionunique-identifier
                  - Arn
              - /invocations
      ResourceId:
        Ref: HelloWorldApihellounique-identifier
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/Resource
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Analytics: v2:deflate64:unique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/CDKMetadata/Default
    Condition: CDKMetadataAvailable
Outputs:
  HelloWorldApiEndpointunique-identifier:
    Value:
      Fn::Join:
        - ""
        - - https://
          - Ref: HelloWorldApiunique-identifier
          - .execute-api.
          - Ref: AWS::Region
          - "."
          - Ref: AWS::URLSuffix
          - /
          - Ref: HelloWorldApiDeploymentStageprodunique-identifier
          - /
Conditions:
  CDKMetadataAvailable:
    Fn::Or:
```

```
      - Fn::Or:
          - Fn::Equals:
              - Ref: AWS::Region
              - af-south-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-east-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-northeast-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-northeast-2
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-south-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-southeast-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-southeast-2
          - Fn::Equals:
              - Ref: AWS::Region
              - ca-central-1
          - Fn::Equals:
              - Ref: AWS::Region
              - cn-north-1
          - Fn::Equals:
              - Ref: AWS::Region
              - cn-northwest-1
      - Fn::Or:
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-central-1
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-north-1
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-south-1
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-west-1
```

```
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-west-2
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-west-3
          - Fn::Equals:
              - Ref: AWS::Region
              - il-central-1
          - Fn::Equals:
              - Ref: AWS::Region
              - me-central-1
          - Fn::Equals:
              - Ref: AWS::Region
              - me-south-1
          - Fn::Equals:
              - Ref: AWS::Region
              - sa-east-1
      - Fn::Or:
          - Fn::Equals:
              - Ref: AWS::Region
              - us-east-1
          - Fn::Equals:
              - Ref: AWS::Region
              - us-east-2
          - Fn::Equals:
              - Ref: AWS::Region
              - us-west-1
          - Fn::Equals:
              - Ref: AWS::Region
              - us-west-2
Parameters:
  BootstrapVersion:
    Type: AWS::SSM::Parameter::Value<String>
    Default: /cdk-bootstrap/hnb659fds/version
    Description: Version of the CDK Bootstrap resources in this environment,
 automatically retrieved from SSM Parameter Store. [cdk:skip]
Rules:
  CheckBootstrapVersion:
    Assertions:
      - Assert:
          Fn::Not:
            - Fn::Contains:
                - - "1"
```

```
                  - "2"
                  - "3"
                  - "4"
                  - "5"
                - Ref: BootstrapVersion
        AssertDescription: CDK bootstrap stack version 6 required. Please run 'cdk
 bootstrap' with a recent version of the CDK CLI.
```

Dengan menggunakan konstruksi L2, Anda mendefinisikan beberapa properti untuk mengonfigurasi sumber daya Anda dan menggunakan metode pembantu untuk mengintegrasikannya bersama-sama. AWS CDK Konfigurasi sebagian besar AWS CloudFormation sumber daya dan properti Anda yang diperlukan untuk menyediakan aplikasi Anda.

Langkah 5: Menerapkan aplikasi Anda

Pada langkah ini, Anda menggunakan AWS CDK CLI `cdk deploy` perintah untuk menyebarkan aplikasi Anda. AWS CDK Bekerja dengan AWS CloudFormation layanan untuk menyediakan sumber daya Anda.

> **Important**
>
> Anda harus melakukan bootstrap satu kali dari lingkungan Anda AWS sebelum penerapan. Untuk petunjuk, lihat Bootstrap lingkungan Anda.

Dari akar proyek Anda, jalankan yang berikut ini. Konfirmasikan perubahan jika diminta:

```
$ cdk deploy

#  Synthesis time: 2.44s

...

Do you wish to deploy these changes (y/n)? y
```

Saat penerapan selesai, AWS CDK CLI akan menampilkan URL titik akhir Anda. Salin URL ini untuk langkah selanjutnya. Berikut ini adalah contohnya:

```
...
```

```
#  HelloWorldStack

#  Deployment time: 45.37s

Outputs:
HelloWorldStack.HelloWorldApiEndpointunique-identifier = https://<api-id>.execute-
api.<region>.amazonaws.com/prod/
Stack ARN:
arn:aws:cloudformation:region:account-id:stack/HelloWorldStack/unique-identifier
...
```

Langkah 6: Berinteraksi dengan aplikasi Anda

Pada langkah ini, Anda memulai permintaan GET ke titik akhir API Anda dan menerima respons fungsi Lambda Anda.

Temukan URL titik akhir Anda dari langkah sebelumnya dan tambahkan `/hello` jalurnya. Kemudian, menggunakan browser atau command prompt Anda, kirim permintaan GET ke titik akhir Anda. Berikut ini adalah contohnya:

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello
{"message":"Hello World!"}%
```

Selamat, Anda telah berhasil membuat, menyebarkan, dan berinteraksi dengan aplikasi Anda menggunakan! AWS CDK

Langkah 7: Hapus aplikasi Anda

Pada langkah ini, Anda menggunakan AWS CDK CLI untuk menghapus aplikasi Anda dari AWS Cloud.

Untuk menghapus aplikasi Anda, jalankan`cdk destroy`. Saat diminta, konfirmasikan permintaan Anda untuk menghapus aplikasi:

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
 #  CdkHelloWorldStack: destroyed
```

Pemecahan Masalah

Kesalahan: {“message”: “Kesalahan server internal”}%

Saat menjalankan fungsi Lambda yang diterapkan, Anda menerima kesalahan ini. Kesalahan ini dapat terjadi karena berbagai alasan.

Untuk memecahkan masalah lebih lanjut

Gunakan tombol AWS CLI untuk menjalankan fungsi Lambda Anda.

1. Ubah file tumpukan Anda untuk menangkap nilai output dari nama fungsi Lambda yang Anda gunakan. Berikut ini adalah contohnya:

```
...

class CdkHelloWorldStack extends Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // Define the Lambda function resource
    // ...

    new CfnOutput(this, 'HelloWorldFunctionName', {
      value: helloWorldFunction.functionName,
      description: 'JavaScript Lambda function'
    });

    // Define the API Gateway resource
    // ...
```

2. Terapkan aplikasi Anda lagi. AWS CDK CLIAkan menampilkan nilai nama fungsi Lambda yang Anda gunakan:

```
$ cdk deploy

#  Synthesis time: 0.29s
...
 #  CdkHelloWorldStack

#  Deployment time: 20.36s

Outputs:
```

```
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. Gunakan tombol AWS CLI untuk menjalankan fungsi Lambda Anda AWS Cloud di dan menampilkan respons ke file teks:

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-
identifier output.txt
```

4. Periksa `output.txt` untuk melihat hasil Anda.

Kemungkinan penyebabnya: Sumber daya API Gateway didefinisikan secara tidak benar di file tumpukan Anda.

Jika `output.txt` menunjukkan respons fungsi Lambda yang berhasil, masalahnya mungkin terkait dengan cara Anda mendefinisikan API REST API Gateway API Anda. Itu AWS CLI memanggil Lambda Anda secara langsung, bukan melalui titik akhir Anda. Periksa kode Anda untuk memastikannya cocok dengan tutorial ini. Kemudian, gunakan lagi.

Kemungkinan penyebabnya: Sumber daya Lambda didefinisikan secara tidak benar di file tumpukan Anda.

Jika `output.txt` mengembalikan kesalahan, masalahnya mungkin dengan cara Anda mendefinisikan fungsi Lambda Anda. Periksa kode Anda untuk memastikannya cocok dengan tutorial ini. Kemudian gunakan lagi.

Buat aplikasi dengan banyak tumpukan

Anda dapat membuat AWS Cloud Development Kit (AWS CDK) aplikasi yang berisi beberapa tumpukan. Saat Anda menerapkan AWS CDK aplikasi, setiap tumpukan menjadi AWS CloudFormation templatnya sendiri. Anda juga dapat mensintesis dan menyebarkan setiap tumpukan satu per satu menggunakan perintah. AWS CDK CLI `cdk deploy`

Tutorial ini mencakup hal-hal berikut:

- Cara memperluas `Stack` kelas untuk menerima properti atau argumen baru.
- Cara menggunakan properti untuk menentukan sumber daya mana yang berisi tumpukan dan konfigurasinya.

- Cara membuat instance beberapa tumpukan dari kelas ini.

Contoh dalam topik ini menggunakan properti Boolean, bernama `encryptBucket` (Python`encrypt_bucket`:). Ini menunjukkan apakah bucket Amazon S3 harus dienkripsi. Jika demikian, tumpukan memungkinkan enkripsi menggunakan kunci yang dikelola oleh AWS Key Management Service (AWS KMS). Aplikasi ini membuat dua instance tumpukan ini, satu dengan enkripsi dan satu tanpa.

Topik

Sebelum Anda mulai

Pertama, instal Node.js dan alat baris AWS CDK perintah, jika Anda belum melakukannya. Lihat Memulai dengan AWS CDK untuk detail.

Selanjutnya, buat AWS CDK proyek dengan memasukkan perintah berikut di baris perintah.

TypeScript

```
mkdir multistack
cd multistack
cdk init --language=typescript
```

JavaScript

```
mkdir multistack
cd multistack
cdk init --language=javascript
```

Python

```
mkdir multistack
```

```
cd multistack
cdk init --language=python
source .venv/bin/activate
pip install -r requirements.txt
```

Java

```
mkdir multistack
cd multistack
cdk init --language=java
```

Anda dapat mengimpor proyek Maven yang dihasilkan ke IDE Java Anda.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

Anda dapat membuka file `src/Pipeline.sln` di Visual Studio.

Tambahkan parameter opsional

`props`Argumen `Stack` konstruktor memenuhi antarmuka. `StackProps` Dalam contoh ini, kami ingin tumpukan menerima properti tambahan untuk memberi tahu kami apakah akan mengenkripsi bucket Amazon S3. Kita harus membuat antarmuka atau kelas yang mencakup properti. Hal ini memungkinkan compiler untuk memastikan bahwa properti memiliki nilai Boolean dan memungkinkan pelengkapan otomatis untuk itu di IDE Anda.

Jadi buka file sumber yang ditunjukkan di IDE atau editor Anda dan tambahkan antarmuka, kelas, atau argumen baru. Kode akan terlihat seperti ini setelah perubahan. Garis yang kami tambahkan ditampilkan dalam huruf tebal.

TypeScript

Berkas: `lib/multistack-stack.ts`

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';

```

```
interface MultiStackProps extends cdk.StackProps {
  encryptBucket?: boolean;
}

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

    // The code that defines your stack goes here
  }
}
```

JavaScript

Berkas: `lib/multistack-stack.js`

JavaScript tidak memiliki fitur antarmuka; kita tidak perlu menambahkan kode apa pun.

```
const cdk = require('aws-cdk-stack');

class MultistackStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // The code that defines your stack goes here
  }
}

module.exports = { MultistackStack }
```

Python

Berkas: `multistack/multistack_stack.py`

Python tidak memiliki fitur antarmuka, jadi kami akan memperluas tumpukan kami untuk menerima properti baru dengan menambahkan argumen kata kunci.

```
import aws_cdk as cdk
from constructs import Construct

class MultistackStack(cdk.Stack):

    # The Stack class doesn't know about our encrypt_bucket parameter,
```

```
    # so accept it separately and pass along any other keyword arguments.
    def __init__(self, scope: Construct, id: str, *, encrypt_bucket=False,
                 **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # The code that defines your stack goes here
```

Java

Berkas: `src/main/java/com/myorg/MultistackStack.java`

Ini lebih rumit daripada yang benar-benar ingin kita masuki untuk memperluas jenis alat peraga di Java. Sebagai gantinya, tulis konstruktor tumpukan untuk menerima parameter Boolean opsional. Karena `props` merupakan argumen opsional, kami akan menulis konstruktor tambahan yang memungkinkan Anda melewatinya. Ini akan default ke`false`.

```
package com.myorg;

import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.constructs.Construct;

import software.amazon.awscdk.services.s3.Bucket;

public class MultistackStack extends Stack {
    // additional constructors to allow props and/or encryptBucket to be omitted
    public MultistackStack(final Construct scope, final String id, boolean
 encryptBucket) {
        this(scope, id, null, encryptBucket);
    }

    public MultistackStack(final Construct scope, final String id) {
        this(scope, id, null, false);
    }

    public MultistackStack(final Construct scope, final String id, final StackProps
 props,
            final boolean encryptBucket) {
        super(scope, id, props);

        // The code that defines your stack goes here
    }
}
```

C#

Berkas: `src/Multistack/MultistackStack.cs`

```
using Amazon.CDK;
using constructs;

namespace Multistack
{


    public class MultiStackProps : StackProps
    {
        public bool? EncryptBucket { get; set; }
    }


    public class MultistackStack : Stack
    {
        public MultistackStack(Construct scope, string id, MultiStackProps props) :
 base(scope, id, props)
        {
            // The code that defines your stack goes here
        }
    }
}
```

Properti baru adalah opsional. Jika `encryptBucket` (Python:`encrypt_bucket`) tidak ada, nilainya adalah`undefined`, atau setara lokal. Bucket tidak akan dienkripsi secara default.

Tentukan kelas tumpukan

Sekarang mari kita mendefinisikan kelas stack kita, menggunakan properti baru kita. Buat kode terlihat seperti berikut ini. Kode yang perlu Anda tambahkan atau ubah ditampilkan dalam huruf tebal.

TypeScript

Berkas: `lib/multistack-stack.ts`

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from constructs;
```

```
import * as s3 from 'aws-cdk-lib/aws-s3';

interface MultistackProps extends cdk.StackProps {
  encryptBucket?: boolean;
}

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

    // Add a Boolean property "encryptBucket" to the stack constructor.
    // If true, creates an encrypted bucket. Otherwise, the bucket is unencrypted.
    // Encrypted bucket uses KMS-managed keys (SSE-KMS).
    if (props && props.encryptBucket) {
      new s3.Bucket(this, "MyGroovyBucket", {
        encryption: s3.BucketEncryption.KMS_MANAGED,
        removalPolicy: cdk.RemovalPolicy.DESTROY
      });
    } else {
      new s3.Bucket(this, "MyGroovyBucket", {
        removalPolicy: cdk.RemovalPolicy.DESTROY});
    }
  }
}
```

JavaScript

Berkas: `lib/multistack-stack.js`

```
const cdk = require('aws-cdk-lib');
const s3 = require('aws-cdk-lib/aws-s3');

class MultistackStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // Add a Boolean property "encryptBucket" to the stack constructor.
    // If true, creates an encrypted bucket. Otherwise, the bucket is unencrypted.
    // Encrypted bucket uses KMS-managed keys (SSE-KMS).
    if ( props && props.encryptBucket) {
      new s3.Bucket(this, "MyGroovyBucket", {
        encryption: s3.BucketEncryption.KMS_MANAGED,
        removalPolicy: cdk.RemovalPolicy.DESTROY
      });
```

```
    } else {
      new s3.Bucket(this, "MyGroovyBucket", {
        removalPolicy: cdk.RemovalPolicy.DESTROY});
    }
  }
}

module.exports = { MultistackStack }
```

Python

Berkas: `multistack/multistack_stack.py`

```
import aws_cdk as cdk
from constructs import Construct
from aws_cdk import aws_s3 as s3

class MultistackStack(cdk.Stack):

    # The Stack class doesn't know about our encrypt_bucket parameter,
    # so accept it separately and pass along any other keyword arguments.
    def __init__(self, scope: Construct, id: str, *, encrypt_bucket=False,
                 **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # Add a Boolean property "encryptBucket" to the stack constructor.
        # If true, creates an encrypted bucket. Otherwise, the bucket is
 unencrypted.
        # Encrypted bucket uses KMS-managed keys (SSE-KMS).
        if encrypt_bucket:
            s3.Bucket(self, "MyGroovyBucket",
                      encryption=s3.BucketEncryption.KMS_MANAGED,
                      removal_policy=cdk.RemovalPolicy.DESTROY)
        else:
            s3.Bucket(self, "MyGroovyBucket",
                     removal_policy=cdk.RemovalPolicy.DESTROY)
```

Java

Berkas: `src/main/java/com/myorg/MultistackStack.java`

```
package com.myorg;
```

```
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.constructs.Construct;
import software.amazon.awscdk.RemovalPolicy;

import software.amazon.awscdk.services.s3.Bucket;
import software.amazon.awscdk.services.s3.BucketEncryption;

public class MultistackStack extends Stack {
    // additional constructors to allow props and/or encryptBucket to be omitted
    public MultistackStack(final Construct scope, final String id,
            boolean encryptBucket) {
        this(scope, id, null, encryptBucket);
    }

    public MultistackStack(final Construct scope, final String id) {
        this(scope, id, null, false);
    }

    // main constructor
    public MultistackStack(final Construct scope, final String id,
            final StackProps props, final boolean encryptBucket) {
        super(scope, id, props);

        // Add a Boolean property "encryptBucket" to the stack constructor.
        // If true, creates an encrypted bucket. Otherwise, the bucket is
        // unencrypted. Encrypted bucket uses KMS-managed keys (SSE-KMS).
        if (encryptBucket) {
            Bucket.Builder.create(this, "MyGroovyBucket")
                    .encryption(BucketEncryption.KMS_MANAGED)
                    .removalPolicy(RemovalPolicy.DESTROY).build();
        } else {
            Bucket.Builder.create(this, "MyGroovyBucket")
                    .removalPolicy(RemovalPolicy.DESTROY).build();
        }
    }
}
```

C#

Berkas: `src/Multistack/MultistackStack.cs`

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;
```

```
namespace Multistack
{

    public class MultiStackProps : StackProps
    {
        public bool? EncryptBucket { get; set; }
    }

    public class MultistackStack : Stack
    {
        public MultistackStack(Construct scope, string id, IMultiStackProps props =
 null) : base(scope, id, props)
        {
            // Add a Boolean property "EncryptBucket" to the stack constructor.
            // If true, creates an encrypted bucket. Otherwise, the bucket is
 unencrypted.
            // Encrypted bucket uses KMS-managed keys (SSE-KMS).
            if (props?.EncryptBucket ?? false)
            {
                new Bucket(this, "MyGroovyBucket", new BucketProps
                {
                    Encryption = BucketEncryption.KMS_MANAGED,
                    RemovalPolicy = RemovalPolicy.DESTROY
                });
            }
            else
            {
                new Bucket(this, "MyGroovyBucket", new BucketProps
                {
                    RemovalPolicy = RemovalPolicy.DESTROY
                });
            }
        }
    }
}
```

Buat dua instance tumpukan

Sekarang kita akan menambahkan kode untuk membuat instance dua tumpukan terpisah. Seperti sebelumnya, baris kode yang ditampilkan dalam huruf tebal adalah yang perlu Anda tambahkan. Hapus `MultistackStack` definisi yang ada.

TypeScript

Berkas: `bin/multistack.ts`

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { MultistackStack } from '../lib/multistack-stack';

const app = new cdk.App();

new MultistackStack(app, "MyWestCdkStack", {
    env: {region: "us-west-1"},
    encryptBucket: false
});

new MultistackStack(app, "MyEastCdkStack", {
    env: {region: "us-east-1"},
    encryptBucket: true
});

app.synth();
```

JavaScript

Berkas: `bin/multistack.js`

```
#!/usr/bin/env node
const cdk = require('aws-cdk-lib');
const { MultistackStack } = require('../lib/multistack-stack');

const app = new cdk.App();

new MultistackStack(app, "MyWestCdkStack", {
    env: {region: "us-west-1"},
    encryptBucket: false
});

new MultistackStack(app, "MyEastCdkStack", {
    env: {region: "us-east-1"},
    encryptBucket: true
});
```

```
app.synth();
```

Python

Berkas: `./app.py`

```
#!/usr/bin/env python3

import aws_cdk as cdk

from multistack.multistack_stack import MultistackStack

app = cdk.App()
MultistackStack(app, "MyWestCdkStack",
                env=cdk.Environment(region="us-west-1"),
                encrypt_bucket=False)

MultistackStack(app, "MyEastCdkStack",
                env=cdk.Environment(region="us-east-1"),
                encrypt_bucket=True)

app.synth()
```

Java

Berkas: `src/main/java/com/myorg/MultistackApp.java`

```
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.Environment;
import software.amazon.awscdk.StackProps;

public class MultistackApp {
    public static void main(final String argv[]) {
        App app = new App();

        new MultistackStack(app, "MyWestCdkStack", StackProps.builder()
                .env(Environment.builder()
                        .region("us-west-1")
                        .build())
                .build(), false);
```

```
        new MultistackStack(app, "MyEastCdkStack", StackProps.builder()
                .env(Environment.builder()
                        .region("us-east-1")
                        .build())
                .build(), true);

        app.synth();
    }
}
```

C#

Berkas: src/Multistack/Program.cs

```
using Amazon.CDK;

namespace Multistack
{
    class Program
    {
        static void Main(string[] args)
        {
            var app = new App();

            new MultistackStack(app, "MyWestCdkStack", new MultiStackProps
            {
                Env = new Environment { Region = "us-west-1" },
                EncryptBucket = false
            });

            new MultistackStack(app, "MyEastCdkStack", new MultiStackProps
            {
                Env = new Environment { Region = "us-east-1" },
                EncryptBucket = true
            });

            app.Synth();
        }
    }
}
```

Kode ini menggunakan properti baru `encryptBucket` (Python:`encrypt_bucket`) di `MultistackStack` kelas untuk membuat contoh berikut:

- Satu tumpukan dengan bucket Amazon S3 terenkripsi di Wilayah. `us-east-1` AWS
- Satu tumpukan dengan bucket Amazon S3 yang tidak terenkripsi di Wilayah. `us-west-1` AWS

Sintesis dan gunakan tumpukan

Sekarang Anda dapat menerapkan tumpukan dari aplikasi. Pertama, sintesis AWS CloudFormation template untuk `MyEastCdkStack` —tumpukan di. `us-east-1` Ini adalah tumpukan dengan bucket S3 terenkripsi.

```
$ cdk synth MyEastCdkStack
```

Untuk menyebarkan tumpukan ini ke AWS akun Anda, keluarkan salah satu perintah berikut. Perintah pertama menggunakan AWS profil default Anda untuk mendapatkan kredensyal untuk menyebarkan tumpukan. Yang kedua menggunakan profil yang Anda tentukan. Untuk *`PROFILE_NAME`*, ganti nama AWS CLI profil yang berisi kredensi yang sesuai untuk disebarkan ke Wilayah. `us-east-1` AWS

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

Hapus

Untuk menghindari biaya untuk sumber daya yang Anda gunakan, hancurkan tumpukan menggunakan perintah berikut.

```
cdk destroy MyEastCdkStack
```

Operasi penghancuran gagal jika ada sesuatu yang disimpan dalam ember tumpukan. Seharusnya tidak ada jika Anda hanya mengikuti instruksi dalam topik ini. Tetapi jika Anda memasukkan sesuatu ke dalam ember, Anda harus menghapus isi ember sebelum menghancurkan tumpukan. (Jangan hapus ember itu sendiri.) Gunakan AWS Management Console atau AWS CLI untuk menghapus isi ember.

Contoh-contoh

Topik ini berisi contoh-contoh berikut:

- Buat aplikasi Hello World tanpa serverMembuat aplikasi tanpa server menggunakan Lambda, API Gateway, dan Amazon S3.
- Membuat layanan AWS Fargate menggunakan AWS CDKMembuat layanan Amazon ECS Fargate dari gambar aktif. DockerHub

Membuat layanan AWS Fargate menggunakan AWS CDK

Contoh ini memandu Anda melalui cara membuat layanan AWS Fargate yang berjalan di cluster Amazon Elastic Container Service (Amazon ECS) Container Service (Amazon ECS) yang diawali oleh Application Load Balancer yang menghadap ke internet dari gambar di Amazon ECR.

Amazon ECS adalah layanan manajemen kontainer yang sangat skalabel, cepat, yang memudahkan untuk menjalankan, menghentikan, dan mengelola kontainer Docker di cluster. Anda dapat meng-host klaster Anda pada infrastruktur tanpa server yang dikelola oleh Amazon ECS dengan meluncurkan layanan atau tugas Anda menggunakan jenis peluncuran Fargate. Untuk kontrol lebih lanjut, Anda dapat meng-host tugas Anda di klaster instans Amazon Elastic Compute Cloud (Amazon EC2) yang dikelola dengan menggunakan jenis peluncuran Amazon EC2.

Tutorial ini menunjukkan cara meluncurkan beberapa layanan menggunakan jenis peluncuran Fargate. Jika Anda telah menggunakan AWS Management Console untuk membuat layanan Fargate, Anda tahu bahwa ada banyak langkah yang harus diikuti untuk menyelesaikan tugas itu. AWS memiliki beberapa tutorial dan topik dokumentasi yang memandu Anda melalui pembuatan layanan Fargate, termasuk:

- Cara Menerapkan Kontainer Docker - AWS
- Menyiapkan dengan Amazon ECS
- Memulai Amazon ECS Menggunakan Fargate

Contoh ini menciptakan layanan Fargate serupa dalam AWS CDK kode.

Konstruksi Amazon ECS yang digunakan dalam tutorial ini membantu Anda menggunakan AWS layanan dengan memberikan manfaat berikut:

- Secara otomatis mengkonfigurasi penyeimbang beban.
- Secara otomatis membuka grup keamanan untuk penyeimbang beban. Hal ini memungkinkan penyeimbang beban untuk berkomunikasi dengan instans tanpa Anda secara eksplisit membuat grup keamanan.
- Secara otomatis memesan ketergantungan antara layanan dan penyeimbang beban yang dilampirkan ke grup target, di mana AWS CDK memberlakukan urutan pembuatan listener yang benar sebelum instance dibuat.
- Secara otomatis mengonfigurasi data pengguna pada grup penskalaan otomatis. Ini membuat konfigurasi yang benar untuk mengaitkan cluster ke AMI.
- Memvalidasi kombinasi parameter lebih awal. Ini memperlihatkan AWS CloudFormation masalah sebelumnya, sehingga menghemat waktu penerapan Anda. Misalnya, tergantung pada tugasnya, mudah untuk salah mengkonfigurasi pengaturan memori. Sebelumnya, Anda tidak akan menemukan kesalahan sampai Anda menerapkan aplikasi Anda. Tetapi sekarang AWS CDK dapat mendeteksi kesalahan konfigurasi dan mengeluarkan kesalahan saat Anda mensintesis aplikasi Anda.
- Secara otomatis menambahkan izin untuk Amazon Elastic Container Registry (Amazon ECR) Registry ECR) jika Anda menggunakan gambar dari Amazon ECR.
- Secara otomatis skala. Ini AWS CDK menyediakan metode sehingga Anda dapat melakukan autoscalinginstances saat menggunakan klaster Amazon EC2. Ini terjadi secara otomatis ketika Anda menggunakan instance di cluster Fargate.

  Selain itu, AWS CDK mencegah instance dihapus saat penskalaan otomatis mencoba mematikan instance, tetapi tugas sedang berjalan atau dijadwalkan pada instance itu.

  Sebelumnya, Anda harus membuat fungsi Lambda untuk memiliki fungsi ini.
- Menyediakan dukungan aset, sehingga Anda dapat menyebarkan sumber dari mesin Anda ke Amazon ECS dalam satu langkah. Sebelumnya, untuk menggunakan sumber aplikasi Anda harus melakukan beberapa langkah manual, seperti mengunggah ke Amazon ECR dan membuat gambar Docker.

Lihat ECS untuk detailnya.

> **Important**
>
> `ApplicationLoadBalancedFargateService`Konstruksi yang akan kami gunakan mencakup banyak AWS komponen, beberapa di antaranya memiliki biaya non-sepele jika

dibiarkan disediakan di AWS akun Anda, bahkan jika Anda tidak menggunakannya. Pastikan untuk membersihkan (cdk destroy) setelah menyelesaikan contoh ini.

Membuat direktori dan menginisialisasi AWS CDK

Mari kita mulai dengan membuat direktori untuk menyimpan AWS CDK kode, dan kemudian membuat AWS CDK aplikasi di direktori itu.

TypeScript

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language typescript
```

JavaScript

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language javascript
```

Python

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language python
source .venv/bin/activate
pip install -r requirements.txt
```

Java

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language java
```

Anda sekarang dapat mengimpor proyek Maven ke IDE Anda.

C#

```
mkdir MyEcsConstruct
```

```
cd MyEcsConstruct
cdk init --language csharp
```

Anda sekarang dapat membuka `src/MyEcsConstruct.sln` di Visual Studio.

Jalankan aplikasi dan konfirmasikan bahwa itu membuat tumpukan kosong.

```
cdk synth
```

Buat layanan Fargate

Ada dua cara berbeda untuk menjalankan tugas penampung Anda dengan Amazon ECS:

- Gunakan jenis `Fargate` peluncuran, tempat Amazon ECS mengelola mesin fisik tempat kontainer Anda berjalan untuk Anda.
- Gunakan jenis `EC2` peluncuran, tempat Anda melakukan pengelolaan, seperti menentukan penskalaan otomatis.

Untuk contoh ini, kita akan membuat layanan Fargate yang berjalan pada cluster ECS yang dihadapkan oleh Application Load Balancer yang menghadap ke internet.

Tambahkan impor modul AWS Construct Library berikut ke file yang ditunjukkan.

TypeScript

Berkas: `lib/my_ecs_construct-stack.ts`

```
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as ecs from "aws-cdk-lib/aws-ecs";
import * as ecs_patterns from "aws-cdk-lib/aws-ecs-patterns";
```

JavaScript

Berkas: `lib/my_ecs_construct-stack.js`

```
const ec2 = require("aws-cdk-lib/aws-ec2");
const ecs = require("aws-cdk-lib/aws-ecs");
const ecs_patterns = require("aws-cdk-lib/aws-ecs-patterns");
```

Python

Berkas: my_ecs_construct/my_ecs_construct_stack.py

```
from aws_cdk import (aws_ec2 as ec2, aws_ecs as ecs,
                     aws_ecs_patterns as ecs_patterns)
```

Java

Berkas: src/main/java/com/myorg/MyEcsConstructStack.java

```
import software.amazon.awscdk.services.ec2.*;
import software.amazon.awscdk.services.ecs.*;
import software.amazon.awscdk.services.ecs.patterns.*;
```

C#

Berkas: src/MyEcsConstruct/MyEcsConstructStack.cs

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

Ganti komentar di akhir konstruktor dengan kode berikut.

TypeScript

```
    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService",
 {
      cluster: cluster, // Required
      cpu: 512, // Default is 256
```

```
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-
sample") },
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is true
    });
```

JavaScript

```
    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService",
 {
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-
sample") },
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is true
    });
```

Python

```
        vpc = ec2.Vpc(self, "MyVpc", max_azs=3)     # default is all AZs in region

        cluster = ecs.Cluster(self, "MyCluster", vpc=vpc)

        ecs_patterns.ApplicationLoadBalancedFargateService(self, "MyFargateService",
            cluster=cluster,            # Required
            cpu=512,                    # Default is 256
            desired_count=6,            # Default is 1
            task_image_options=ecs_patterns.ApplicationLoadBalancedTaskImageOptions(
                image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample")),
            memory_limit_mib=2048,      # Default is 512
            public_load_balancer=True)  # Default is True
```

Java

```
        Vpc vpc = Vpc.Builder.create(this, "MyVpc")
                             .maxAzs(3)  // Default is all AZs in region
                             .build();

        Cluster cluster = Cluster.Builder.create(this, "MyCluster")
                             .vpc(vpc).build();

        // Create a load-balanced Fargate service and make it public
        ApplicationLoadBalancedFargateService.Builder.create(this,
 "MyFargateService")
                        .cluster(cluster)           // Required
                        .cpu(512)                   // Default is 256
                         .desiredCount(6)            // Default is 1
                         .taskImageOptions(
                                 ApplicationLoadBalancedTaskImageOptions.builder()
                                         .image(ContainerImage.fromRegistry("amazon/
amazon-ecs-sample"))
                                         .build())
                         .memoryLimitMiB(2048)       // Default is 512
                         .publicLoadBalancer(true)   // Default is true
                         .build();
```

C#

```
            var vpc = new Vpc(this, "MyVpc", new VpcProps
            {
                MaxAzs = 3 // Default is all AZs in region
            });

            var cluster = new Cluster(this, "MyCluster", new ClusterProps
            {
                Vpc = vpc
            });

            // Create a load-balanced Fargate service and make it public
            new ApplicationLoadBalancedFargateService(this, "MyFargateService",
                new ApplicationLoadBalancedFargateServiceProps
                {
                    Cluster = cluster,          // Required
                    DesiredCount = 6,           // Default is 1
                    TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
```

```
                    {
                        Image = ContainerImage.FromRegistry("amazon/amazon-ecs-
sample")
                    },
                    MemoryLimitMiB = 2048,      // Default is 256
                    PublicLoadBalancer = true   // Default is true
                }
            );
```

Simpan dan pastikan itu berjalan dan membuat tumpukan.

```
cdk synth
```

Tumpukan ratusan baris, jadi kami tidak akan menunjukkannya di sini. Tumpukan harus berisi satu instance default, subnet pribadi dan subnet publik untuk tiga Availability Zones, dan grup keamanan.

Menyebarkan tumpukan.

```
cdk deploy
```

AWS CloudFormation menampilkan informasi tentang lusinan langkah yang diperlukan saat menerapkan aplikasi Anda.

Begitulah mudahnya membuat layanan Amazon ECS yang didukung Fargate untuk menjalankan image Docker.

Hapus

Untuk menghindari AWS biaya yang tidak terduga, hancurkan AWS CDK tumpukan Anda setelah Anda selesai dengan latihan ini.

```
cdk destroy
```

AWS CDK contoh

Untuk lebih banyak contoh AWS CDK tumpukan dan aplikasi dalam bahasa pemrograman favorit Anda yang didukung, lihat repositori AWS CDK Contoh aktif. GitHub

AWS CDK alat

Bagian ini berisi informasi tentang AWS CDK alat yang tercantum di bawah ini.

Topik

AWS CDK Toolkit (**cdk**perintah)

AWS CDK Toolkit, `cdk` perintah CLI, adalah alat utama untuk berinteraksi dengan aplikasi Anda. AWS CDK Ini mengeksekusi aplikasi Anda, menginterogasi model aplikasi yang Anda tentukan, dan menghasilkan serta menerapkan AWS CloudFormation template yang dihasilkan oleh. AWS CDK Ini juga menyediakan fitur lain yang berguna untuk membuat dan bekerja dengan AWS CDK proyek. Topik ini berisi informasi tentang kasus penggunaan umum CDK Toolkit.

AWS CDK Toolkit diinstal dengan Node Package Manager. Dalam kebanyakan kasus, kami sarankan untuk menginstalnya secara global.

```
npm install -g aws-cdk             # install latest version
npm install -g aws-cdk@X.YY.Z      # install specific version
```

> **Tip**
>
> Jika Anda secara teratur bekerja dengan beberapa versi AWS CDK, pertimbangkan untuk menginstal versi AWS CDK Toolkit yang cocok di masing-masing proyek CDK. Untuk melakukan ini, hilangkan `-g` dari `npm install` perintah. Kemudian gunakan `npx aws-cdk` untuk memanggilnya. Ini menjalankan versi lokal jika ada, kembali ke versi global jika tidak.

Perintah toolkit

Semua perintah CDK Toolkit dimulai dengan`cdk`, yang diikuti oleh subperintah (`list`,, `synthesize`dll.). Beberapa subperintah memiliki versi yang lebih pendek (`ls`,`synth`, dll.)

yang setara. Opsi dan argumen mengikuti subperintah dalam urutan apa pun. Perintah yang tersedia dirangkum di sini.

Perintah	Fungsi
`cdk list` (`ls`)	Daftar tumpukan di aplikasi
`cdk synthesize` (`synth`)	Mensintesis dan mencetak CloudFormation template untuk satu atau lebih tumpukan tertentu
`cdk bootstrap`	Menyebarkan tumpukan pementasan CDK Toolkit; lihat the section called “Bootstrapping”
`cdk deploy`	Menyebarkan satu atau lebih tumpukan tertentu
`cdk destroy`	Menghancurkan satu atau lebih tumpukan tertentu
`cdk diff`	Membandingkan tumpukan yang ditentukan dan dependensinya dengan tumpukan yang diterapkan atau templat lokal CloudFormation
`cdk import`	Menggunakan impor CloudFormation sumber daya untuk membawa sumber daya yang ada ke tumpukan yang dikelola oleh CDK
`cdk metadata`	Menampilkan metadata tentang tumpukan yang ditentukan
`cdk init`	Membuat proyek CDK baru di direktori saat ini dari template yang ditentukan
`cdk context`	Mengelola nilai konteks cache
`cdk docs` (`doc`)	Membuka Referensi API CDK di browser Anda
`cdk doctor`	Memeriksa proyek CDK Anda untuk potensi masalah

Untuk opsi yang tersedia untuk setiap perintah, lihatthe section called “Bantuan bawaan”.

Menentukan opsi dan nilainya

Opsi baris perintah dimulai dengan dua tanda hubung ()`--`. Beberapa opsi yang sering digunakan memiliki sinonim satu huruf yang dimulai dengan tanda hubung tunggal (misalnya, `--app` memiliki sinonim). `-a` Urutan opsi dalam perintah AWS CDK Toolkit tidak penting.

Semua opsi menerima nilai, yang harus mengikuti nama opsi. Nilai dapat dipisahkan dari nama dengan spasi putih atau dengan tanda = sama dengan. Dua opsi berikut ini setara.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

Beberapa opsi adalah bendera (Booleans). Anda dapat menentukan `true` atau `false` sebagai nilainya. Jika Anda tidak memberikan nilai, nilainya dianggap`true`. Anda juga dapat mengawali nama opsi dengan `no-` menyiratkan`false`.

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

Beberapa opsi, yaitu`--context`,`--parameters`,`--plugin`,`--tags`, dan`--trust`, dapat ditentukan lebih dari sekali untuk menentukan beberapa nilai. Ini dicatat sebagai `[array]` tipe dalam bantuan CDK Toolkit. Sebagai contoh:

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

Bantuan bawaan

AWS CDK Toolkit memiliki bantuan terintegrasi. Anda dapat melihat bantuan umum tentang utilitas dan daftar subperintah yang disediakan dengan mengeluarkan:

```
cdk --help
```

Untuk melihat bantuan untuk subperintah tertentu, misalnya`deploy`, tentukan sebelum `--help` bendera.

```
cdk deploy --help
```

Masalah `cdk version` untuk menampilkan versi AWS CDK Toolkit. Berikan informasi ini saat meminta dukungan.

Pelaporan versi

Untuk mendapatkan wawasan tentang bagaimana AWS CDK digunakan, konstruksi yang digunakan oleh AWS CDK aplikasi dikumpulkan dan dilaporkan dengan menggunakan sumber daya yang diidentifikasi sebagai`AWS::CDK::Metadata`. Sumber daya ini ditambahkan ke AWS CloudFormation templat, dan dapat dengan mudah ditinjau. Informasi ini juga dapat digunakan oleh AWS untuk mengidentifikasi tumpukan menggunakan konstruksi dengan masalah keamanan atau keandalan yang diketahui. Ini juga dapat digunakan untuk menghubungi pengguna mereka dengan informasi penting.

> **Note**
>
> Sebelum versi 1.93.0, AWS CDK dilaporkan nama dan versi modul dimuat selama sintesis, bukan konstruksi yang digunakan dalam tumpukan.

Secara default, AWS CDK laporan penggunaan konstruksi dalam modul NPM berikut yang digunakan dalam tumpukan:

- AWS CDK modul inti
- AWS Membangun modul Perpustakaan
- AWS Modul Konstruksi Solusi
- AWS Modul Kit Penyebaran Pertanian Render

Sumber `AWS::CDK::Metadata` daya terlihat seperti berikut ini.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
  Properties:
```

```
    Analytics:
 "v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/
XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz
+AqqLDt2Y3YMnLYWwAAAA="
```

`Analytics`Properti ini adalah daftar konstruksi yang di-gzip, dikodekan base64, dan dikodekan awalan di tumpukan.

Untuk memilih keluar dari pelaporan versi, gunakan salah satu metode berikut:

- Gunakan cdk perintah dengan --no-version-reporting argumen untuk memilih keluar untuk satu perintah.

  ```
  cdk --no-version-reporting synth
  ```

  Ingat, AWS CDK Toolkit mensintesis template baru sebelum menerapkan, jadi Anda juga harus menambahkan `--no-version-reporting` perintah. `cdk deploy`
- Setel versionReporting ke false di `./cdk.json` atau`~/.cdk.json`. Ini memilih keluar kecuali Anda ikut serta dengan menentukan `--version-reporting` pada perintah individual.

  ```
  {
    "app": "...",
    "versionReporting": false
  }
  ```

Otentikasi dengan AWS

Ada berbagai cara di mana Anda dapat mengonfigurasi akses terprogram ke AWS sumber daya, tergantung pada lingkungan dan AWS akses yang tersedia untuk Anda.

Untuk memilih metode otentikasi dan mengonfigurasinya untuk CDK Toolkit, lihat Autentikasi dan akses di Panduan Referensi AWS SDK dan Alat.

Pendekatan yang direkomendasikan untuk pengguna baru yang berkembang secara lokal, yang tidak diberi metode otentikasi oleh majikan mereka, adalah dengan mengatur. AWS IAM Identity Center Metode ini termasuk menginstal AWS CLI untuk kemudahan konfigurasi dan untuk masuk secara teratur ke portal AWS akses. Jika Anda memilih metode ini, lingkungan Anda harus berisi elemen-elemen berikut setelah Anda menyelesaikan prosedur untuk autentikasi IAM Identity Center di AWS SDK dan Tools Reference Guide:

- Itu AWS CLI, yang Anda gunakan untuk memulai sesi portal AWS akses sebelum Anda menjalankan aplikasi Anda.
- AWSconfigFile bersama yang memiliki `[default]` profil dengan serangkaian nilai konfigurasi yang dapat direferensikan dari file. AWS CDK Untuk menemukan lokasi file ini, lihat Lokasi file bersama di AWS SDK dan Panduan Referensi Alat.
- `config`File bersama menetapkan `region`pengaturan. Ini menetapkan default Wilayah AWS yang digunakan Toolkit AWS CDK dan CDK untuk AWS permintaan.
- CDK Toolkit menggunakan konfigurasi penyedia token SSO profil untuk memperoleh kredensional sebelum mengirim permintaan ke. AWS`sso_role_name`Nilai, yang merupakan peran IAM yang terhubung ke set izin Pusat Identitas IAM, harus memungkinkan akses ke yang Layanan AWS digunakan dalam aplikasi Anda.

  `config`File contoh berikut menunjukkan profil default yang diatur dengan konfigurasi penyedia token SSO. `sso_session`Pengaturan profil mengacu pada `sso-session`bagian bernama. `sso-session`Bagian ini berisi pengaturan untuk memulai sesi portal AWS akses.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Mulai sesi portal AWS akses

Sebelum mengakses Layanan AWS, Anda memerlukan sesi portal AWS akses aktif untuk CDK Toolkit untuk menggunakan autentikasi IAM Identity Center untuk menyelesaikan kredensi. Bergantung pada panjang sesi yang dikonfigurasi, akses Anda pada akhirnya akan kedaluwarsa dan CDK Toolkit akan mengalami kesalahan otentikasi. Jalankan perintah berikut di AWS CLI untuk masuk ke portal AWS akses.

```
aws sso login
```

Jika konfigurasi penyedia token SSO Anda menggunakan profil bernama alih-alih profil default, perintahnya adalah`aws sso login --profile` *NAME*. Tentukan juga profil ini saat mengeluarkan cdk perintah menggunakan --profile opsi atau variabel `AWS_PROFILE` lingkungan.

Untuk menguji apakah Anda sudah memiliki sesi aktif, jalankan AWS CLI perintah berikut.

```
aws sts get-caller-identity
```

Respons terhadap perintah ini harus melaporkan akun IAM Identity Center dan set izin yang dikonfigurasi dalam `config` file bersama.

> **Note**
>
> Jika Anda sudah memiliki sesi portal AWS akses aktif dan menjalankan`aws sso login`, Anda tidak akan diminta untuk memberikan kredensi.
> Proses masuk dapat meminta Anda untuk mengizinkan AWS CLI akses ke data Anda. Karena AWS CLI dibangun di atas SDK untuk Python, pesan izin mungkin berisi variasi nama. `botocore`

Menentukan Wilayah dan konfigurasi lainnya

CDK Toolkit perlu mengetahui AWS Wilayah yang Anda gunakan dan cara mengautentikasi. AWS Ini diperlukan untuk operasi penerapan dan untuk mengambil nilai konteks selama sintesis. Bersama-sama, akun dan Wilayah Anda membentuk lingkungan.

Wilayah dapat ditentukan menggunakan variabel lingkungan atau dalam file konfigurasi. Ini adalah variabel dan file yang sama yang digunakan oleh AWS alat lain seperti AWS CLI dan berbagai AWS SDK. CDK Toolkit mencari informasi ini dalam urutan berikut.

- Variabel `AWS_DEFAULT_REGION` lingkungan.
- Profil bernama didefinisikan dalam AWS `config` file standar dan ditentukan menggunakan `--profile` opsi pada `cdk` perintah.
- `[default]`Bagian dari AWS `config` file standar.

Selain menentukan AWS otentikasi dan Wilayah di `[default]` bagian, Anda juga dapat menambahkan satu atau beberapa `[profile` *NAME*`]` bagian, di mana *NAME* adalah nama profil.

Untuk informasi selengkapnya tentang profil bernama, lihat File konfigurasi dan kredensial bersama di Panduan Referensi AWS SDK dan Alat.

AWS `config`File standar terletak di `~/.aws/config` (MacOS/Linux) atau `%USERPROFILE%\.aws\config` (Windows). Untuk detail dan lokasi alternatif, lihat Lokasi file konfigurasi dan kredensial bersama di Panduan Referensi AWS SDK dan Alat

Lingkungan yang Anda tentukan dalam AWS CDK aplikasi menggunakan `env` properti stack digunakan selama sintesis. Ini digunakan untuk menghasilkan AWS CloudFormation template khusus lingkungan, dan selama penerapan, itu mengganti akun atau Wilayah yang ditentukan oleh salah satu metode sebelumnya. Untuk informasi selengkapnya, lihat the section called “Lingkungan”.

> **Note**
>
> AWS CDK Menggunakan kredensi dari file sumber yang sama dengan AWS alat dan SDK lainnya, termasuk file. AWS Command Line Interface Namun, AWS CDK mungkin berperilaku agak berbeda dari alat-alat ini. Ini menggunakan di AWS SDK for JavaScript bawah tenda. Untuk detail selengkapnya tentang menyiapkan kredensil AWS SDK for JavaScript, lihat Menyetel kredensil.

Anda dapat secara opsional menggunakan opsi `--role-arn` (atau`-r`) untuk menentukan ARN peran IAM yang harus digunakan untuk penerapan. Peran ini harus diasumsikan oleh AWS akun yang digunakan.

Menentukan perintah aplikasi

Banyak fitur CDK Toolkit memerlukan satu atau lebih AWS CloudFormation template disintesis, yang pada gilirannya memerlukan menjalankan aplikasi Anda. Program AWS CDK pendukung yang ditulis dalam berbagai bahasa. Oleh karena itu, ia menggunakan opsi konfigurasi untuk menentukan perintah yang tepat yang diperlukan untuk menjalankan aplikasi Anda. Opsi ini dapat ditentukan dalam dua cara.

Pertama, dan paling umum, dapat ditentukan menggunakan `app` kunci di dalam file`cdk.json`. Ini ada di direktori utama AWS CDK proyek Anda. CDK Toolkit menyediakan perintah yang sesuai saat membuat proyek baru dengan. `cdk init` Berikut adalah `cdk.json` dari TypeScript proyek baru, misalnya.

```
{
```

```
  "app": "npx ts-node bin/hello-cdk.ts"
}
```

CDK Toolkit mencari `cdk.json` di direktori kerja saat ini ketika mencoba menjalankan aplikasi Anda. Karena itu, Anda mungkin membiarkan shell tetap terbuka di direktori utama proyek Anda untuk mengeluarkan perintah CDK Toolkit.

CDK Toolkit juga mencari kunci aplikasi di `~/.cdk.json` (yaitu, di direktori home Anda) jika tidak dapat menemukannya. `./cdk.json` Menambahkan perintah aplikasi di sini dapat berguna jika Anda biasanya bekerja dengan kode CDK dalam bahasa yang sama.

Jika Anda berada di beberapa direktori lain, atau menjalankan aplikasi Anda menggunakan perintah selain yang ada di`cdk.json`, gunakan opsi `--app` (atau`-a`) untuk menentukannya.

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

Saat menerapkan, Anda juga dapat menentukan direktori yang berisi rakitan cloud yang disintesis, seperti`cdk.out`, sebagai nilai. --app Tumpukan yang ditentukan digunakan dari direktori ini; aplikasi tidak disintesis.

Menentukan tumpukan

Banyak perintah CDK Toolkit (misalnya,`cdk deploy`) berfungsi pada tumpukan yang ditentukan dalam aplikasi Anda. Jika aplikasi Anda hanya berisi satu tumpukan, CDK Toolkit mengasumsikan yang Anda maksud jika Anda tidak menentukan tumpukan secara eksplisit.

Jika tidak, Anda harus menentukan tumpukan atau tumpukan yang ingin Anda kerjakan. Anda dapat melakukan ini dengan menentukan tumpukan yang diinginkan berdasarkan ID satu per satu pada baris perintah. Ingat bahwa ID adalah nilai yang ditentukan oleh argumen kedua ketika Anda membuat instance tumpukan.

```
cdk synth PipelineStack LambdaStack
```

Anda juga dapat menggunakan wildcard untuk menentukan ID yang cocok dengan pola.

- ?cocok dengan karakter tunggal apa pun
- *cocok dengan sejumlah karakter (*sendiri cocok dengan semua tumpukan)
- **cocok dengan segala sesuatu dalam hierarki

Anda juga dapat menggunakan --all opsi untuk menentukan semua tumpukan.

Jika aplikasi Anda menggunakan CDK Pipelines, CDK Toolkit memahami tumpukan dan tahapan Anda sebagai hierarki. Selain itu, --all opsi dan * wildcard hanya cocok dengan tumpukan tingkat atas. Untuk mencocokkan semua tumpukan, gunakan**. Juga gunakan ** untuk menunjukkan semua tumpukan di bawah hierarki tertentu.

Saat menggunakan wildcard, lampirkan pola dalam tanda kutip, atau lepaskan wildcard dengan. \ Jika tidak, shell Anda mungkin mencoba memperluas pola ke nama-nama file di direktori saat ini. Paling-paling, ini tidak akan melakukan apa yang Anda harapkan; paling buruk, Anda bisa menggunakan tumpukan yang tidak Anda inginkan. Ini tidak sepenuhnya diperlukan di Windows karena `cmd.exe` tidak memperluas wildcard, tetapi tetap merupakan praktik yang baik.

```
cdk synth "*Stack"    # PipelineStack, LambdaStack, etc.
cdk synth 'Stack?'    # StackA, StackB, Stack1, etc.
cdk synth \*          # All stacks in the app, or all top-level stacks in a CDK
 Pipelines app
cdk synth '**'        # All stacks in a CDK Pipelines app
cdk synth 'PipelineStack/Prod/**'   # All stacks in Prod stage in a CDK Pipelines app
```

> **Note**
>
> Urutan di mana Anda menentukan tumpukan belum tentu urutan di mana mereka akan diproses. AWS CDK Toolkit memperhitungkan dependensi antar tumpukan saat memutuskan urutan untuk memprosesnya. Sebagai contoh, katakanlah satu tumpukan menggunakan nilai yang dihasilkan oleh yang lain (seperti ARN dari sumber daya yang didefinisikan dalam tumpukan kedua). Dalam hal ini, tumpukan kedua disintesis sebelum yang pertama karena ketergantungan ini. Anda dapat menambahkan dependensi antar tumpukan secara manual menggunakan metode tumpukan. `addDependency()`

Bootstrapping lingkungan Anda AWS

Menyebarkan tumpukan dengan CDK memerlukan AWS CDK sumber daya khusus khusus untuk disediakan. `cdk bootstrap`Perintah menciptakan sumber daya yang diperlukan untuk Anda. Anda hanya perlu bootstrap jika Anda menerapkan tumpukan yang membutuhkan sumber daya khusus ini. Lihat the section called “Bootstrapping” untuk detail.

```
cdk bootstrap
```

Jika dikeluarkan tanpa argumen, seperti yang ditunjukkan di sini, `cdk bootstrap` perintah mensintesis aplikasi saat ini dan bootstrap lingkungan tempat tumpukannya akan digunakan. Jika aplikasi berisi tumpukan agnostik lingkungan, yang tidak secara eksplisit menentukan lingkungan, akun default dan Wilayah akan di-bootstrap, atau lingkungan yang ditentukan menggunakan. `--profile`

Di luar aplikasi, Anda harus secara eksplisit menentukan lingkungan yang akan di-bootstrap. Anda juga dapat melakukannya untuk mem-bootstrap lingkungan yang tidak ditentukan dalam aplikasi atau AWS profil lokal Anda. Kredensil harus dikonfigurasi (misalnya dalam`~/.aws/credentials`) untuk akun dan Wilayah yang ditentukan. Anda dapat menentukan profil yang berisi kredensil yang diperlukan.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.
cdk bootstrap 1111111111/us-east-1
cdk bootstrap --profile test 1111111111/us-east-1
```

Important

Setiap lingkungan (kombinasi akun/wilayah) tempat Anda menerapkan tumpukan semacam itu harus di-bootstrap secara terpisah.

Anda mungkin dikenakan AWS biaya untuk apa yang AWS CDK disimpan di sumber daya bootstrap. Selain itu, jika Anda menggunakan`-bootstrap-customer-key`, kunci AWS KMS akan dibuat, yang juga dikenakan biaya per lingkungan.

Note

Versi sebelumnya dari template bootstrap membuat kunci KMS secara default. Untuk menghindari biaya, re-bootstrap menggunakan`--no-bootstrap-customer-key`.

Note

CDK Toolkit v2 tidak mendukung template bootstrap asli, dijuluki template lama, digunakan secara default dengan CDK v1.

> **Important**
>
> Template bootstrap modern secara efektif memberikan izin yang tersirat oleh `--cloudformation-execution-policies` ke AWS akun mana pun dalam daftar. `--trust` Secara default, ini memperluas izin untuk membaca dan menulis ke sumber daya apa pun di akun bootstrap. Pastikan untuk mengonfigurasi tumpukan bootstrap dengan kebijakan dan akun tepercaya yang nyaman bagi Anda.

Membuat aplikasi baru

Untuk membuat aplikasi baru, buat direktori untuknya, lalu, di dalam direktori, keluarkan`cdk init`.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

Bahasa yang didukung (*BAHASA*) adalah:

Kode	Bahasa
`typescript`	TypeScript
`javascript`	JavaScript
`python`	Python
`java`	Java
`csharp`	C#

TEMPLATE adalah template opsional. Jika template yang diinginkan adalah aplikasi, default, Anda dapat menghilangkannya. Template yang tersedia adalah:

Templat	Deskripsi
app(default)	Membuat AWS CDK aplikasi kosong.

Templat	Deskripsi
`sample-app`	Membuat AWS CDK aplikasi dengan tumpukan yang berisi antrian Amazon SQS dan topik Amazon SNS.

Template menggunakan nama folder project untuk menghasilkan nama file dan class di dalam aplikasi baru Anda.

Daftar tumpukan

Untuk melihat daftar ID tumpukan di AWS CDK aplikasi Anda, masukkan salah satu perintah setara berikut:

```
cdk list
cdk ls
```

Jika aplikasi Anda berisi tumpukan CDK Pipelines, CDK Toolkit menampilkan nama tumpukan sebagai jalur sesuai dengan lokasinya dalam hierarki pipeline. (Misalnya,`PipelineStack`,`PipelineStack/Prod`, dan`PipelineStack/Prod/MyService`.)

Jika aplikasi berisi banyak tumpukan, Anda dapat menentukan ID tumpukan penuh atau sebagian dari tumpukan yang akan dicantumkan. Untuk informasi selengkapnya, lihat the section called “Menentukan tumpukan”.

Tambahkan `--long` bendera untuk melihat informasi lebih lanjut tentang tumpukan, termasuk nama tumpukan dan lingkungannya (AWS akun dan Wilayah).

Mensintesis tumpukan

`cdk synthesize`Perintah (hampir selalu disingkat`synth`) mensintesis tumpukan yang ditentukan dalam aplikasi Anda ke dalam template. CloudFormation

```
cdk synth         # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"     # all stacks in app
```

> **Note**
>
> CDK Toolkit benar-benar menjalankan aplikasi Anda dan mensintesis template baru sebelum sebagian besar operasi (seperti saat menerapkan atau membandingkan tumpukan). Template ini disimpan secara default di `cdk.out` direktori. `cdk synth`Perintah hanya mencetak template yang dihasilkan untuk satu atau lebih tumpukan tertentu.

Lihat `cdk synth --help` untuk semua opsi yang tersedia. Beberapa opsi yang paling sering digunakan tercakup dalam bagian berikut.

Menentukan nilai konteks

Gunakan `-c` opsi `--context` or untuk meneruskan nilai konteks runtime ke aplikasi CDK Anda.

```
# specify a single context value
cdk synth --context key=value MyStack

# specify multiple context values (any number)
cdk synth --context key1=value1 --context key2=value2 MyStack
```

Saat menerapkan beberapa tumpukan, nilai konteks yang ditentukan biasanya diteruskan ke semuanya. Jika mau, Anda dapat menentukan nilai yang berbeda untuk setiap tumpukan dengan mengawali nama tumpukan ke nilai konteks.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

Menentukan format tampilan

Secara default, template yang disintesis ditampilkan dalam format YAMG. Tambahkan `--json` bendera untuk menampilkannya dalam format JSON sebagai gantinya.

```
cdk synth --json MyStack
```

Menentukan direktori output

Tambahkan opsi `--output` (`-o`) untuk menulis template yang disintesis ke direktori selain`cdk.out`.

```
cdk synth --output=~/templates
```

Menyebarkan tumpukan

`cdk deploy`Subperintah menyebarkan satu atau beberapa tumpukan tertentu ke akun Anda. AWS

```
cdk deploy        # if app contains only one stack
cdk deploy MyStack
cdk deploy Stack1 Stack2
cdk deploy "*"    # all stacks in app
```

> **Note**
>
> CDK Toolkit menjalankan aplikasi Anda dan mensintesis AWS CloudFormation template baru sebelum menerapkan apa pun. Oleh karena itu, sebagian besar opsi baris perintah yang dapat Anda gunakan `cdk synth` (misalnya,`--context`) juga dapat digunakan`cdk deploy`.

Lihat `cdk deploy --help` untuk semua opsi yang tersedia. Beberapa opsi yang paling berguna tercakup dalam bagian berikut.

Melewatkan sintesis

cdk deployPerintah biasanya mensintesis tumpukan aplikasi Anda sebelum menerapkan untuk memastikan bahwa penerapan mencerminkan versi terbaru aplikasi Anda. Jika Anda tahu bahwa Anda belum mengubah kode sejak terakhircdk synth, Anda dapat menekan langkah sintesis yang berlebihan saat menerapkan. Untuk melakukannya, tentukan `cdk.out` direktori proyek Anda di --app opsi.

```
cdk deploy --app cdk.out StackOne StackTwo
```

Menonaktifkan rollback

AWS CloudFormation memiliki kemampuan untuk memutar kembali perubahan sehingga penerapan bersifat atomik. Ini berarti bahwa mereka berhasil atau gagal secara keseluruhan. AWS CDK Mewarisi kemampuan ini karena mensintesis dan menyebarkan AWS CloudFormation template.

Rollback memastikan bahwa sumber daya Anda berada dalam keadaan konsisten setiap saat, yang sangat penting untuk tumpukan produksi. Namun, saat Anda masih mengembangkan infrastruktur, beberapa kegagalan tidak dapat dihindari, dan memutar kembali penerapan yang gagal dapat memperlambat Anda.

Untuk alasan ini, CDK Toolkit memungkinkan Anda menonaktifkan rollback dengan menambahkan `--no-rollback` ke perintah Anda. `cdk deploy` Dengan flag ini, penerapan yang gagal tidak dibatalkan. Sebagai gantinya, sumber daya yang digunakan sebelum sumber daya yang gagal tetap ada, dan penerapan berikutnya dimulai dengan sumber daya yang gagal. Anda akan menghabiskan lebih sedikit waktu menunggu penerapan dan lebih banyak waktu mengembangkan infrastruktur Anda.

Pertukaran panas

Gunakan `--hotswap` tanda dengan `cdk deploy` untuk mencoba memperbarui AWS sumber daya Anda secara langsung alih-alih membuat set AWS CloudFormation perubahan dan menerapkannya. Penerapan kembali ke AWS CloudFormation penerapan jika hot swapping tidak memungkinkan.

Saat ini hot swapping mendukung fungsi Lambda, mesin status Step Functions, dan image container Amazon ECS. `--hotswap`Bendera juga menonaktifkan rollback (yaitu, menyiratkan). `--no-rollback`

> **Important**
>
> Hot-swapping tidak disarankan untuk penerapan produksi.

Modus menonton

Mode tontonan CDK Toolkit (cdk deploy --watch, atau singkatnya) terus memantau file sumber dan aset aplikasi CDK Anda cdk watch untuk perubahan. Ini segera melakukan penyebaran tumpukan yang ditentukan ketika perubahan terdeteksi.

Secara default, penerapan ini menggunakan `--hotswap` flag, yang mempercepat penyebaran perubahan pada fungsi Lambda. Itu juga kembali ke penerapan AWS CloudFormation jika Anda telah mengubah konfigurasi infrastruktur. Agar `cdk watch` selalu melakukan AWS CloudFormation penerapan penuh, tambahkan `--no-hotswap` bendera ke. `cdk watch`

Setiap perubahan yang `cdk watch` dibuat saat sudah melakukan penerapan digabungkan menjadi satu penerapan, yang dimulai segera setelah penerapan yang sedang berlangsung selesai.

Mode menonton menggunakan `"watch"` kunci dalam proyek `cdk.json` untuk menentukan file mana yang akan dipantau. Secara default, file-file ini adalah file dan aset aplikasi Anda, tetapi ini dapat diubah dengan memodifikasi `"include"` dan `"exclude"` entri di `"watch"` kunci. `cdk.json`File berikut menunjukkan contoh entri ini.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

`cdk watch`mengeksekusi `"build"` perintah dari `cdk.json` untuk membangun aplikasi Anda sebelum sintesis. Jika penerapan Anda memerlukan perintah apa pun untuk membuat atau mengemas kode Lambda Anda (atau apa pun yang tidak ada di aplikasi CDK Anda), tambahkan di sini.

Wildcard bergaya Git, keduanya `*` dan`**`, dapat digunakan di tombol dan. `"watch"` `"build"` Setiap jalur ditafsirkan relatif terhadap direktori induk. `cdk.json` Nilai default `include` adalah`**/*`, yang berarti semua file dan direktori di direktori root proyek. `exclude`adalah opsional.

> **Important**
>
> Mode tontonan tidak disarankan untuk penerapan produksi.

Menentukan parameter AWS CloudFormation

AWS CDK Toolkit mendukung menentukan AWS CloudFormation parameter pada penerapan. Anda dapat memberikan ini pada baris perintah mengikuti `--parameters` bendera.

```
cdk deploy MyStack --parameters uploadBucketName=UploadBucket
```

Untuk menentukan beberapa parameter, gunakan beberapa `--parameters` bendera.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
 downloadBucketName=DownBucket
```

Jika Anda menerapkan beberapa tumpukan, Anda dapat menentukan nilai yang berbeda dari setiap parameter untuk setiap tumpukan. Untuk melakukannya, awali nama parameter dengan nama tumpukan dan titik dua. Jika tidak, nilai yang sama diteruskan ke semua tumpukan.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --
parameters YourStack:uploadBucketName=UpBucket
```

Secara default, AWS CDK mempertahankan nilai parameter dari penerapan sebelumnya dan menggunakannya dalam penerapan selanjutnya jika tidak ditentukan secara eksplisit. Gunakan `--no-previous-parameters` bendera untuk meminta semua parameter ditentukan.

Menentukan file output

Jika tumpukan Anda mendeklarasikan AWS CloudFormation output, ini biasanya ditampilkan di layar pada akhir penerapan. Untuk menulisnya ke file dalam format JSON, gunakan `--outputs-file` bendera.

```
cdk deploy --outputs-file outputs.json MyStack
```

Perubahan terkait keamanan

Untuk melindungi Anda dari perubahan yang tidak diinginkan yang memengaruhi postur keamanan Anda, AWS CDK Toolkit meminta Anda untuk menyetujui perubahan terkait keamanan sebelum menerapkannya. Anda dapat menentukan tingkat perubahan yang memerlukan persetujuan:

```
cdk deploy --require-approval LEVEL
```

`LEVEL` dapat menjadi salah satu dari berikut ini:

Jangka Waktu	Arti
`never`	Persetujuan tidak pernah diperlukan
`any-change`	Memerlukan persetujuan atas IAM atau perubahan security-group-related
`broadening` (default)	Memerlukan persetujuan ketika pernyataan IAM atau peraturan lalu lintas ditambahkan; penghapusan tidak memerlukan persetujuan

Pengaturan juga dapat dikonfigurasi dalam `cdk.json` file.

```
{
  "app": "...",
  "requireApproval": "never"
}
```

Membandingkan tumpukan

`cdk diff`Perintah membandingkan versi stack saat ini (dan dependensinya) yang ditentukan dalam aplikasi Anda dengan versi yang sudah di-deploy, atau dengan AWS CloudFormation template yang disimpan, dan menampilkan daftar perubahan.

```
Stack HelloCdkStack
IAM Statement Changes
###############################################################################################
#   # Resource                    # Effect # Action                      # Principal
                  # Condition #
###############################################################################################
# + # ${Custom::S3AutoDeleteObject # Allow  # sts:AssumeRole              #
 Service:lambda.amazonaws.com #           #
#   # sCustomResourceProvider/Role #        #                           #
                  #           #
#   # .Arn}                       #        #                           #
                  #           #
###############################################################################################
# + # ${MyFirstBucket.Arn}         # Allow  # s3:DeleteObject*           # AWS:
${Custom::S3AutoDeleteOb #           #
#   # ${MyFirstBucket.Arn}/*       #        # s3:GetBucket*             #
 jectsCustomResourceProvider/ #           #
#   #                             #        # s3:GetObject*             # Role.Arn}
                  #           #
#   #                             #        # s3:List*                  #
                  #           #
###############################################################################################
IAM Policy Changes
###############################################################################################
#   # Resource                                          # Managed Policy ARN
                              #
###############################################################################################
# + # ${Custom::S3AutoDeleteObjectsCustomResourceProvider/Ro # {"Fn::Sub":"arn:
${AWS::Partition}:iam::aws:policy/serv #
```

```
#   # le}                                                    # ice-role/
AWSLambdaBasicExecutionRole"}                 #
########################################################################################################
(NOTE: There may be security-related changes not in this list. See https://github.com/
aws/aws-cdk/issues/1299)

Parameters
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3Bucket
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3BucketBF7A7F3
 {"Type":"String","Description":"S3 bucket for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3VersionKey
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3VersionKeyFAF
 {"Type":"String","Description":"S3 key for asset version
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
ArtifactHash
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392ArtifactHashE56
 {"Type":"String","Description":"Artifact hash for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}

Resources
[+] AWS::S3::BucketPolicy MyFirstBucket/Policy MyFirstBucketPolicy3243DEFD
[+] Custom::S3AutoDeleteObjects MyFirstBucket/AutoDeleteObjectsCustomResource
 MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E
[+] AWS::IAM::Role Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092
[+] AWS::Lambda::Function Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F
[~] AWS::S3::Bucket MyFirstBucket MyFirstBucketB8884501
 ## [~] DeletionPolicy
 #   ## [-] Retain
 #   ## [+] Delete
 ## [~] UpdateReplacePolicy
     ## [-] Retain
     ## [+] Delete
```

Untuk membandingkan tumpukan aplikasi Anda dengan penerapan yang ada:

```
cdk diff MyStack
```

Untuk membandingkan tumpukan aplikasi Anda dengan CloudFormation templat yang disimpan:

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

Mengimpor sumber daya yang ada ke dalam tumpukan

Anda dapat menggunakan `cdk import` perintah untuk membawa sumber daya di bawah pengelolaan CloudFormation untuk AWS CDK tumpukan tertentu. Ini berguna jika Anda bermigrasi ke AWS CDK, atau memindahkan sumber daya antar tumpukan atau mengubah id logisnya. `cdk import` menggunakan impor CloudFormation sumber daya. Lihat daftar sumber daya yang dapat diimpor di sini.

Untuk mengimpor sumber daya yang ada ke AWS CDK tumpukan, ikuti langkah-langkah berikut:

- Pastikan sumber daya saat ini tidak dikelola oleh CloudFormation tumpukan lain. Jika ya, pertama-tama setel kebijakan penghapusan ke `RemovalPolicy.RETAIN` tumpukan sumber daya saat ini dan lakukan penerapan. Kemudian, hapus sumber daya dari tumpukan dan lakukan penerapan lain. Proses ini akan memastikan bahwa sumber daya tidak lagi dikelola oleh CloudFormation tetapi tidak menghapusnya.
- Jalankan a `cdk diff` untuk memastikan tidak ada perubahan yang tertunda pada AWS CDK tumpukan yang ingin Anda impor sumber daya. Satu-satunya perubahan yang diizinkan dalam operasi “impor” adalah penambahan sumber daya baru yang ingin Anda impor.
- Tambahkan konstruksi untuk sumber daya yang ingin Anda impor ke tumpukan Anda. Misalnya, jika Anda ingin mengimpor bucket Amazon S3, tambahkan sesuatu seperti. `new s3.Bucket(this, 'ImportedS3Bucket', {});` Jangan membuat modifikasi apa pun pada sumber daya lain.

  Anda juga harus memastikan untuk secara tepat memodelkan status yang dimiliki sumber daya saat ini ke dalam definisi. Untuk contoh bucket, pastikan untuk menyertakan AWS KMS kunci, kebijakan siklus hidup, dan hal lain yang relevan tentang bucket. Jika tidak, operasi pembaruan berikutnya mungkin tidak melakukan apa yang Anda harapkan.

  Anda dapat memilih apakah akan menyertakan nama bucket fisik atau tidak. Kami biasanya menyarankan untuk tidak memasukkan nama sumber daya ke dalam definisi AWS CDK sumber daya Anda sehingga menjadi lebih mudah untuk menyebarkan sumber daya Anda beberapa kali.

- Jalankan `cdk import` *`STACKNAME`*.
- Jika nama sumber daya tidak ada dalam model Anda, CLI akan meminta Anda untuk meneruskan nama sebenarnya dari sumber daya yang Anda impor. Setelah ini, impor dimulai.
- Ketika `cdk import` melaporkan keberhasilan, sumber daya sekarang dikelola oleh AWS CDK dan CloudFormation. Setiap perubahan berikutnya yang Anda buat pada properti resource di AWS CDK aplikasi Anda, konfigurasi build akan diterapkan pada penerapan berikutnya.
- Untuk mengonfirmasi bahwa definisi sumber daya di AWS CDK aplikasi Anda cocok dengan status sumber daya saat ini, Anda dapat memulai operasi deteksi CloudFormation drift.

Fitur ini saat ini tidak mendukung pengimporan sumber daya ke tumpukan bersarang.

Konfigurasi (**`cdk.json`**)

Nilai default untuk banyak flag baris perintah CDK Toolkit dapat disimpan dalam file proyek atau dalam `cdk.json` `.cdk.json` file di direktori pengguna Anda. Berikut ini adalah referensi abjad ke pengaturan konfigurasi yang didukung.

Kunci	Catatan	Opsi CDK Toolkit
`app`	Perintah yang menjalankan aplikasi CDK.	--app
`assetMetadata`	Jika`false`, CDK tidak menambahkan metadata ke sumber daya yang menggunakan aset.	--no-asset-metadata
`bootstrapKmsKeyId`	Mengganti ID AWS KMS kunci yang digunakan untuk mengenkripsi bucket penerapan Amazon S3.	--bootstrap-kms-key-id
`build`	Perintah yang mengkompilasi atau membangun aplikasi CDK sebelum sintesis. Tidak diizinkan masuk`~/.cdk.json` .	--build

Kunci	Catatan	Opsi CDK Toolkit
`browser`	Perintah untuk meluncurkan browser Web untuk `cdk docs` subperintah.	--browser
`context`	Lihat the section called "Konteks". Nilai konteks dalam file konfigurasi tidak akan dihapus olehcdk context --clear. (CDK Toolkit menempatkan nilai konteks yang di-cache di.) `cdk.context.json`	--context
`debug`	Jika`true`, CDK Toolkit memancarkan informasi lebih rinci yang berguna untuk debugging.	--debug
`language`	Bahasa yang akan digunakan untuk menginisialisasi proyek baru.	--language
`lookups`	Jika`false`, tidak ada pencarian konteks yang diizinkan. Sintesis akan gagal jika ada pencarian konteks yang perlu dilakukan.	--no-lookups
`notices`	Jika`false`, menekan tampilan pesan tentang kerentanan keamanan, regresi, dan versi yang tidak didukung.	--no-notices

Kunci	Catatan	Opsi CDK Toolkit
`output`	Nama direktori tempat perakitan cloud yang disintesis akan dipancarkan (default). `"cdk.out"`	--output
`outputsFile`	File yang AWS CloudFormation output dari tumpukan yang digunakan akan ditulis (dalam format JSON).	--outputs-file
`pathMetadata`	Jika`false`, metadata jalur CDK tidak ditambahkan ke templat yang disintesis.	--no-path-metadata
`plugin`	Array JSON yang menentukan nama paket atau jalur lokal paket yang memperluas CDK	--plugin
`profile`	Nama AWS profil default yang digunakan untuk menentukan Wilayah dan kredensil akun.	--profile
`progress`	Jika disetel ke`"events"`, CDK Toolkit menampilkan semua AWS CloudFormation peristiwa selama penerapan, bukan bilah kemajuan.	--progress
`requireApproval`	Tingkat persetujuan default untuk perubahan keamanan. Lihat the section called “Perubahan terkait keamanan”	--require-approval
`rollback`	Jika`false`, penerapan yang gagal tidak dibatalkan.	--no-rollback

Kunci	Catatan	Opsi CDK Toolkit
`staging`	Jika`false`, aset tidak disalin ke direktori output (gunakan untuk debugging lokal dari file sumber dengan AWS SAM).	--no-staging
`tags`	Objek JSON yang berisi tag (pasangan kunci-nilai) untuk tumpukan.	--tags
`toolkitBucketName`	Nama bucket Amazon S3 yang digunakan untuk menyebarkan aset seperti fungsi Lambda dan gambar kontainer (lihat. the section called “Bootstrapping lingkungan Anda AWS”	--toolkit-bucket-name
`toolkitStackName`	Nama tumpukan bootstrap (lihatthe section called “Bootstrapping lingkungan Anda AWS”.	--toolkit-stack-name
`versionReporting`	Jika`false`, memilih keluar dari pelaporan versi.	--no-version-reporting
`watch`	Objek JSON yang berisi `"include"` dan `"exclude"` kunci yang menunjukkan file mana yang harus (atau tidak boleh) memicu pembangunan kembali proyek saat diubah. Lihat the section called “Modus menonton”.	--watch

cdk migratereferensi perintah

Referensi untuk AWS Cloud Development Kit (AWS CDK) perintah Command Line Interface (CLI)`cdk migrate`. Untuk informasi lebih lanjut tentang penggunaan`cdk migrate`, lihatMigrasikan sumber daya dan AWS CloudFormation templat yang ada ke AWS CDK.

`cdk migrate`Perintah memigrasikan AWS sumber daya, AWS CloudFormation tumpukan, dan templat lokal AWS CloudFormation yang diterapkan ke. AWS CDK

Topik

Penggunaan

```
$ cdk migrate <options>
```

Opsi

Opsi yang diperlukan

`--stack-name` *`STRING`*

Nama AWS CloudFormation tumpukan yang akan dibuat dalam aplikasi CDK setelah bermigrasi.

Wajib: Ya

Opsi bersyarat

`--from-path` *`PATH`*

Jalur ke AWS CloudFormation template untuk bermigrasi. Berikan opsi ini untuk menentukan template lokal.

Diperlukan: Bersyarat. Diperlukan jika bermigrasi dari AWS CloudFormation templat lokal.

`--from-scan` *STRING*

Saat memigrasikan sumber daya yang digunakan dari AWS lingkungan, gunakan opsi ini untuk menentukan apakah pemindaian baru harus dimulai atau apakah AWS CDK CLI pemindaian terakhir yang berhasil digunakan.

Diperlukan: Bersyarat. Diperlukan saat bermigrasi dari sumber daya yang diterapkan AWS .

Nilai yang diterima:`most-recent`, `new`

`--from-stack`

Berikan opsi ini untuk bermigrasi dari tumpukan yang diterapkan AWS CloudFormation . Gunakan `--stack-name` untuk menentukan nama AWS CloudFormation tumpukan yang digunakan.

Diperlukan: Bersyarat. Diperlukan jika bermigrasi dari tumpukan yang diterapkan AWS CloudFormation .

Opsi opsional

`--account` *STRING*

Akun untuk mengambil template AWS CloudFormation tumpukan dari.

Wajib: Tidak

Default: AWS CDK CLI Memperoleh informasi akun dari sumber default.

`--compress`

Berikan opsi ini untuk mengompres proyek CDK yang dihasilkan menjadi ZIP file.

Wajib: Tidak

`--filter` *ARRAY*

Gunakan saat memigrasikan sumber daya yang diterapkan dari AWS akun dan. Wilayah AWS Opsi ini menentukan filter untuk menentukan sumber daya yang digunakan untuk bermigrasi.

Opsi ini menerima array pasangan kunci-nilai, di mana kunci mewakili jenis filter dan nilai mewakili nilai untuk memfilter.

Berikut ini adalah kunci yang diterima:

- `resource-identifier`— Pengenal untuk sumber daya. Nilai dapat berupa ID logis atau fisik sumber daya. Misalnya, `resource-identifier="ClusterName"`.
- `resource-type-prefix`— Awalan tipe AWS CloudFormation sumber daya. Misalnya, tentukan `resource-type-prefix="AWS::DynamoDB::"` untuk memfilter semua sumber daya Amazon DynamoDB.
- `tag-key`— Kunci dari tag sumber daya. Misalnya, `tag-key="myTagKey"`.
- `tag-value`— Nilai tag sumber daya. Misalnya, `tag-value="myTagValue"`.

Berikan beberapa pasangan kunci-nilai untuk logika `AND` bersyarat. Contoh berikut memfilter untuk sumber daya DynamoDB apa pun yang ditandai `myTagKey` dengan sebagai kunci tag:. `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

Berikan `--filter` opsi beberapa kali dalam satu perintah untuk logika `OR` bersyarat. Contoh berikut memfilter untuk sumber daya apa pun yang merupakan sumber daya DynamoDB atau ditandai `myTagKey` dengan sebagai kunci tag:. `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

Wajib: Tidak

`--language` *`STRING`*

Bahasa pemrograman yang digunakan untuk proyek CDK yang dibuat selama migrasi.

Wajib: Tidak

Nilai yang diterima:`typescript`,`python`,`java`,`csharp`,`go`.

Default: `typescript`

`--output-path` *`PATH`*

Jalur keluaran untuk proyek CDK yang dimigrasi.

Wajib: Tidak

Default: Secara default, AWS CDK CLI akan menggunakan direktori kerja Anda saat ini.

`--region` *`STRING`*

Wilayah AWS Untuk mengambil template AWS CloudFormation tumpukan dari.

Wajib: Tidak

Default: AWS CDK CLI Memperoleh Wilayah AWS informasi dari sumber default.

AWS Toolkit for Visual Studio Code

AWS Toolkit for Visual Studio Code adalah plugin open source untuk Visual Studio Code yang membuatnya lebih mudah untuk membuat, men-debug, dan menyebarkan aplikasi. AWS Toolkit ini memberikan pengalaman terintegrasi untuk mengembangkan AWS CDK aplikasi. Ini termasuk fitur AWS CDK Explorer untuk daftar AWS CDK proyek Anda dan menelusuri berbagai komponen aplikasi CDK. Instal AWS Toolkit dan pelajari lebih lanjut tentang menggunakan AWS CDK Explorer.

AWS SAM integrasi

The AWS CDK and the AWS Serverless Application Model (AWS SAM) dapat bekerja sama untuk memungkinkan Anda membangun dan menguji aplikasi tanpa server secara lokal yang ditentukan dalam CDK. Untuk informasi selengkapnya, lihat AWS Cloud Development Kit (AWS CDK)di Panduan AWS SAM Pengembang. Untuk menginstal SAM CLI, lihat Menginstal CLI AWS SAM.

Menguji konstruksi

Dengan AWS CDK, infrastruktur Anda dapat diuji seperti kode lain yang Anda tulis. Pendekatan standar untuk menguji AWS CDK aplikasi menggunakan modul pernyataan dan kerangka pengujian populer seperti Jest for TypeScript dan atau Pytest untuk JavaScript Python. AWS CDK

Ada dua kategori tes yang dapat Anda tulis untuk AWS CDK aplikasi.

- Pernyataan berbutir halus menguji aspek spesifik dari AWS CloudFormation templat yang dihasilkan, seperti “sumber daya ini memiliki properti ini dengan nilai ini.” Tes ini dapat mendeteksi regresi. Mereka juga berguna saat Anda mengembangkan fitur baru menggunakan pengembangan berbasis tes. (Anda dapat menulis tes terlebih dahulu, kemudian membuatnya lulus dengan menulis implementasi yang benar.) Pernyataan berbutir halus adalah tes yang paling sering digunakan.
- Tes snapshot menguji template yang disintesis terhadap AWS CloudFormation template dasar yang disimpan sebelumnya. Tes snapshot memungkinkan Anda melakukan refactor secara bebas, karena Anda dapat yakin bahwa kode refactored bekerja persis dengan cara yang sama seperti aslinya. Jika perubahan itu disengaja, Anda dapat menerima baseline baru untuk pengujian masa depan. Namun, peningkatan CDK juga dapat menyebabkan templat yang disintesis berubah, jadi Anda tidak dapat hanya mengandalkan snapshot untuk memastikan implementasi Anda benar.

> **Note**
>
> Versi lengkap aplikasi TypeScript, Python, dan Java yang digunakan sebagai contoh dalam topik ini tersedia di. GitHub

Memulai

Untuk mengilustrasikan cara menulis tes ini, kita akan membuat tumpukan yang berisi mesin AWS Step Functions status dan AWS Lambda fungsi. Fungsi Lambda berlangganan topik Amazon SNS dan hanya meneruskan pesan ke mesin status.

Pertama, buat proyek aplikasi CDK kosong menggunakan CDK Toolkit dan instal perpustakaan yang kita perlukan. Konstruksi yang akan kita gunakan semuanya ada dalam paket CDK utama, yang merupakan ketergantungan default dalam proyek yang dibuat dengan CDK Toolkit. Namun, Anda harus menginstal kerangka pengujian Anda.

TypeScript

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

Buat direktori untuk pengujian Anda.

```
mkdir test
```

Edit proyek `package.json` untuk memberi tahu NPM cara menjalankan Jest, dan untuk memberi tahu Jest jenis file apa yang harus dikumpulkan. Perubahan yang diperlukan adalah sebagai berikut.

- Tambahkan `test` kunci baru ke `scripts` bagian
- Tambahkan Jest dan tipenya ke bagian `devDependencies`
- Tambahkan kunci `jest` tingkat atas baru dengan deklarasi `moduleFileExtensions`

Perubahan ini ditunjukkan pada garis besar berikut. Tempatkan teks baru di tempat yang ditunjukkan`package.json`. Placeholder "..." menunjukkan bagian file yang ada yang tidak boleh diubah.

```
{
  ...
  "scripts": {
    ...
    "test": "jest"
  },
  "devDependencies": {
    ...
    "@types/jest": "^24.0.18",
    "jest": "^24.9.0"
  },
  "jest": {
    "moduleFileExtensions": ["js"]
  }
}
```

JavaScript

```
mkdir state-machine && cd state-machine
cdk init --language=javascript
npm install --save-dev jest
```

Buat direktori untuk pengujian Anda.

```
mkdir test
```

Edit proyek `package.json` untuk memberi tahu NPM cara menjalankan Jest, dan untuk memberi tahu Jest jenis file apa yang harus dikumpulkan. Perubahan yang diperlukan adalah sebagai berikut.

- Tambahkan `test` kunci baru ke `scripts` bagian
- Tambahkan Jest ke bagian `devDependencies`
- Tambahkan kunci `jest` tingkat atas baru dengan deklarasi `moduleFileExtensions`

Perubahan ini ditunjukkan pada garis besar berikut. Tempatkan teks baru di tempat yang ditunjukkan`package.json`. Placeholder "..." menunjukkan bagian file yang ada yang tidak boleh diubah.

```
{
  ...
  "scripts": {
    ...
    "test": "jest"
  },
  "devDependencies": {
    ...
    "jest": "^24.9.0"
  },
  "jest": {
    "moduleFileExtensions": ["js"]
  }
}
```

Python

```
mkdir state-machine && cd state-machine
```

```
cdk init --language=python
source .venv/bin/activate
python -m pip install -r requirements.txt
python -m pip install -r requirements-dev.txt
```

Java

```
mkdir state-machine && cd-state-machine
cdk init --language=java
```

Buka proyek di IDE Java pilihan Anda. (Di Eclipse, gunakan File> Import > Existing Maven Projects.)

C#

```
mkdir state-machine && cd-state-machine
cdk init --language=csharp
```

Buka `src\StateMachine.sln` di Visual Studio.

Klik kanan solusi di Solution Explorer dan pilih Add > New Project. Cari MStest C # dan tambahkan Proyek Uji MStest untuk C #. (Nama `TestProject1` defaultnya baik-baik saja.)

Klik kanan `TestProject1` dan pilih Add > Project Reference, dan tambahkan `StateMachine` proyek sebagai referensi.

Contoh tumpukan

Inilah tumpukan yang akan diuji dalam topik ini. Seperti yang telah kami jelaskan sebelumnya, ini berisi fungsi Lambda dan mesin status Step Functions, dan menerima satu atau beberapa topik Amazon SNS. Fungsi Lambda berlangganan topik Amazon SNS dan meneruskannya ke mesin status.

Anda tidak perlu melakukan sesuatu yang istimewa untuk membuat aplikasi dapat diuji. Faktanya, tumpukan CDK ini tidak berbeda dengan cara yang penting dari tumpukan contoh lainnya dalam Panduan ini.

TypeScript

Tempatkan kode berikut di`lib/state-machine-stack.ts`:

```
import * as cdk from "aws-cdk-lib";
import * as sns from "aws-cdk-lib/aws-sns";
import * as sns_subscriptions from "aws-cdk-lib/aws-sns-subscriptions";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as sfn from "aws-cdk-lib/aws-stepfunctions";
import { Construct } from "constructs";

export interface StateMachineStackProps extends cdk.StackProps {
  readonly topics: sns.Topic[];
}

export class StateMachineStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: StateMachineStackProps) {
    super(scope, id, props);

    // In the future this state machine will do some work...
    const stateMachine = new sfn.StateMachine(this, "StateMachine", {
      definition: new sfn.Pass(this, "StartState"),
    });

    // This Lambda function starts the state machine.
    const func = new lambda.Function(this, "LambdaFunction", {
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "handler",
      code: lambda.Code.fromAsset("./start-state-machine"),
      environment: {
        STATE_MACHINE_ARN: stateMachine.stateMachineArn,
      },
    });
    stateMachine.grantStartExecution(func);

    const subscription = new sns_subscriptions.LambdaSubscription(func);
    for (const topic of props.topics) {
      topic.addSubscription(subscription);
    }
  }
}
```

JavaScript

Tempatkan kode berikut di`lib/state-machine-stack.js`:

```
const cdk = require("aws-cdk-lib");
```

```
const sns = require("aws-cdk-lib/aws-sns");
const sns_subscriptions = require("aws-cdk-lib/aws-sns-subscriptions");
const lambda = require("aws-cdk-lib/aws-lambda");
const sfn = require("aws-cdk-lib/aws-stepfunctions");

class StateMachineStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // In the future this state machine will do some work...
    const stateMachine = new sfn.StateMachine(this, "StateMachine", {
      definition: new sfn.Pass(this, "StartState"),
    });

    // This Lambda function starts the state machine.
    const func = new lambda.Function(this, "LambdaFunction", {
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "handler",
      code: lambda.Code.fromAsset("./start-state-machine"),
      environment: {
        STATE_MACHINE_ARN: stateMachine.stateMachineArn,
      },
    });
    stateMachine.grantStartExecution(func);

    const subscription = new sns_subscriptions.LambdaSubscription(func);
    for (const topic of props.topics) {
      topic.addSubscription(subscription);
    }
  }
}

module.exports = { StateMachineStack }
```

Python

Tempatkan kode berikut di`state_machine/state_machine_stack.py`:

```
from typing import List

import aws_cdk.aws_lambda as lambda_
import aws_cdk.aws_sns as sns
import aws_cdk.aws_sns_subscriptions as sns_subscriptions
import aws_cdk.aws_stepfunctions as sfn
```

```
import aws_cdk as cdk

class StateMachineStack(cdk.Stack):
    def __init__(
        self,
        scope: cdk.Construct,
        construct_id: str,
        *,
        topics: List[sns.Topic],
        **kwargs
    ) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # In the future this state machine will do some work...
        state_machine = sfn.StateMachine(
            self, "StateMachine", definition=sfn.Pass(self, "StartState")
        )

        # This Lambda function starts the state machine.
        func = lambda_.Function(
            self,
            "LambdaFunction",
            runtime=lambda_.Runtime.NODEJS_18_X,
            handler="handler",
            code=lambda_.Code.from_asset("./start-state-machine"),
            environment={
                "STATE_MACHINE_ARN": state_machine.state_machine_arn,
            },
        )
        state_machine.grant_start_execution(func)

        subscription = sns_subscriptions.LambdaSubscription(func)
        for topic in topics:
            topic.add_subscription(subscription)
```

Java

```
package software.amazon.samples.awscdkassertionssamples;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.lambda.Code;
```

```
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.sns.ITopicSubscription;
import software.amazon.awscdk.services.sns.Topic;
import software.amazon.awscdk.services.sns.subscriptions.LambdaSubscription;
import software.amazon.awscdk.services.stepfunctions.Pass;
import software.amazon.awscdk.services.stepfunctions.StateMachine;

import java.util.Collections;
import java.util.List;

public class StateMachineStack extends Stack {
    public StateMachineStack(final Construct scope, final String id, final
 List<Topic> topics) {
        this(scope, id, null, topics);
    }

    public StateMachineStack(final Construct scope, final String id, final
 StackProps props, final List<Topic> topics) {
        super(scope, id, props);

        // In the future this state machine will do some work...
        final StateMachine stateMachine = StateMachine.Builder.create(this,
 "StateMachine")
                .definition(new Pass(this, "StartState"))
                .build();

        // This Lambda function starts the state machine.
        final Function func = Function.Builder.create(this, "LambdaFunction")
                .runtime(Runtime.NODEJS_18_X)
                .handler("handler")
                .code(Code.fromAsset("./start-state-machine"))
                .environment(Collections.singletonMap("STATE_MACHINE_ARN",
 stateMachine.getStateMachineArn()))
                .build();
        stateMachine.grantStartExecution(func);

        final ITopicSubscription subscription = new LambdaSubscription(func);
        for (final Topic topic : topics) {
            topic.addSubscription(subscription);
        }
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.StepFunctions;
using Amazon.CDK.AWS.SNS;
using Amazon.CDK.AWS.SNS.Subscriptions;
using Constructs;

using System.Collections.Generic;

namespace AwsCdkAssertionSamples
{
    public class StateMachineStackProps : StackProps
    {
        public Topic[] Topics;
    }

    public class StateMachineStack : Stack
    {

        internal StateMachineStack(Construct scope, string id,
 StateMachineStackProps props = null) : base(scope, id, props)
        {
            // In the future this state machine will do some work...
            var stateMachine = new StateMachine(this, "StateMachine", new
 StateMachineProps
            {
                Definition = new Pass(this, "StartState")
            });

            // This Lambda function starts the state machine.
            var func = new Function(this, "LambdaFunction", new FunctionProps
            {
                Runtime = Runtime.NODEJS_18_X,
                Handler = "handler",
                Code = Code.FromAsset("./start-state-machine"),
                Environment = new Dictionary<string, string>
                {
                    { "STATE_MACHINE_ARN", stateMachine.StateMachineArn }
                }
            });
            stateMachine.GrantStartExecution(func);
```

```
            foreach (Topic topic in props?.Topics ?? new Topic[0])
            {
                var subscription = new LambdaSubscription(func);
            }

        }
    }
}
```

Kita akan memodifikasi titik masuk utama aplikasi sehingga kita tidak benar-benar membuat instance stack kita. Kami tidak ingin menyebarkannya secara tidak sengaja. Pengujian kami akan membuat aplikasi dan instance tumpukan untuk pengujian. Ini adalah taktik yang berguna bila dikombinasikan dengan pengembangan berbasis pengujian: pastikan tumpukan melewati semua pengujian sebelum Anda mengaktifkan penerapan.

TypeScript

Dalam `bin/state-machine.ts`:

```
#!/usr/bin/env node
import * as cdk from "aws-cdk-lib";

const app = new cdk.App();

// Stacks are intentionally not created here -- this application isn't meant to
// be deployed.
```

JavaScript

Dalam `bin/state-machine.js`:

```
#!/usr/bin/env node
const cdk = require("aws-cdk-lib");

const app = new cdk.App();

// Stacks are intentionally not created here -- this application isn't meant to
// be deployed.
```

Python

Dalam `app.py`:

```
#!/usr/bin/env python3
import os

import aws_cdk  as cdk

app = cdk.App()

# Stacks are intentionally not created here -- this application isn't meant to
# be deployed.

app.synth()
```

Java

```
package software.amazon.samples.awscdkassertionssamples;

import software.amazon.awscdk.App;

public class SampleApp {
    public static void main(final String[] args) {
        App app = new App();

        // Stacks are intentionally not created here -- this application isn't meant
 to be deployed.

        app.synth();
    }
}
```

C#

```
using Amazon.CDK;

namespace AwsCdkAssertionSamples
{
    sealed class Program
    {
        public static void Main(string[] args)
```

```
        {
            var app = new App();

            // Stacks are intentionally not created here -- this application isn't
 meant to be deployed.

            app.Synth();
        }
    }
}
```

Fungsi Lambda

Contoh tumpukan kami mencakup fungsi Lambda yang memulai mesin status kami. Kita harus menyediakan kode sumber untuk fungsi ini sehingga CDK dapat menggabungkan dan menyebarkannya sebagai bagian dari pembuatan sumber daya fungsi Lambda.

- Buat folder `start-state-machine` di direktori utama aplikasi.
- Di folder ini, buat setidaknya satu file. Misalnya, Anda dapat menyimpan kode berikut di`start-state-machines/index.js`.

```
exports.handler = async function (event, context) {
 return 'hello world';
};
```

Namun, file apa pun akan berfungsi, karena kami tidak akan benar-benar menyebarkan tumpukan.

Menjalankan tes

Sebagai referensi, berikut adalah perintah yang Anda gunakan untuk menjalankan pengujian di AWS CDK aplikasi Anda. Ini adalah perintah yang sama yang akan Anda gunakan untuk menjalankan pengujian dalam proyek apa pun menggunakan kerangka pengujian yang sama. Untuk bahasa yang memerlukan langkah build, sertakan itu untuk memastikan pengujian Anda telah dikompilasi.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

Bangun solusi Anda (F6) untuk menemukan pengujian, lalu jalankan pengujian (Test > Run All Tests). Untuk memilih tes mana yang akan dijalankan, buka Test Explorer (Test > Test Explorer).

Atau:

```
dotnet test src
```

Pernyataan berbutir halus

Langkah pertama untuk menguji tumpukan dengan pernyataan berbutir halus adalah mensintesis tumpukan, karena kami menulis pernyataan terhadap template yang dihasilkan. AWS CloudFormation

Kami `StateMachineStackStack` mengharuskan kami meneruskan topik Amazon SNS untuk diteruskan ke mesin negara. Jadi dalam pengujian kami, kami akan membuat tumpukan terpisah untuk memuat topik.

Biasanya, saat menulis aplikasi CDK, Anda dapat mensubkelas `Stack` dan membuat instance topik Amazon SNS di konstruktor tumpukan. Dalam pengujian kami, kami membuat instance `Stack` secara langsung, lalu meneruskan tumpukan ini sebagai cakupan, melampirkannya ke tumpukan. `Topic` Ini setara secara fungsional dan kurang bertele-tele. Ini juga membantu membuat tumpukan yang hanya digunakan dalam pengujian “terlihat berbeda” dari tumpukan yang ingin Anda terapkan.

TypeScript

```
import { Capture, Match, Template } from "aws-cdk-lib/assertions";
import * as cdk from "aws-cdk-lib";
import * as sns from "aws-cdk-lib/aws-sns";
import { StateMachineStack } from "../lib/state-machine-stack";

describe("StateMachineStack", () => {
  test("synthesizes the way we expect", () => {
    const app = new cdk.App();

    // Since the StateMachineStack consumes resources from a separate stack
    // (cross-stack references), we create a stack for our SNS topics to live
    // in here. These topics can then be passed to the StateMachineStack later,
    // creating a cross-stack reference.
    const topicsStack = new cdk.Stack(app, "TopicsStack");

    // Create the topic the stack we're testing will reference.
    const topics = [new sns.Topic(topicsStack, "Topic1", {})];

    // Create the StateMachineStack.
    const stateMachineStack = new StateMachineStack(app, "StateMachineStack", {
      topics: topics, // Cross-stack reference
    });

    // Prepare the stack for assertions.
    const template = Template.fromStack(stateMachineStack);


}
```

JavaScript

```
const { Capture, Match, Template } = require("aws-cdk-lib/assertions");
const cdk = require("aws-cdk-lib");
const sns = require("aws-cdk-lib/aws-sns");
const { StateMachineStack } = require("../lib/state-machine-stack");

describe("StateMachineStack", () => {
  test("synthesizes the way we expect", () => {
    const app = new cdk.App();

    // Since the StateMachineStack consumes resources from a separate stack
```

```
    // (cross-stack references), we create a stack for our SNS topics to live
    // in here. These topics can then be passed to the StateMachineStack later,
    // creating a cross-stack reference.
    const topicsStack = new cdk.Stack(app, "TopicsStack");

    // Create the topic the stack we're testing will reference.
    const topics = [new sns.Topic(topicsStack, "Topic1", {})];

    // Create the StateMachineStack.
    const StateMachineStack = new StateMachineStack(app, "StateMachineStack", {
      topics: topics, // Cross-stack reference
    });

    // Prepare the stack for assertions.
    const template = Template.fromStack(stateMachineStack);
```

Python

```
from aws_cdk import aws_sns as sns
import aws_cdk as cdk
from aws_cdk.assertions import Template

from app.state_machine_stack import StateMachineStack

def test_synthesizes_properly():
    app = cdk.App()

    # Since the StateMachineStack consumes resources from a separate stack
    # (cross-stack references), we create a stack for our SNS topics to live
    # in here. These topics can then be passed to the StateMachineStack later,
    # creating a cross-stack reference.
    topics_stack = cdk.Stack(app, "TopicsStack")

    # Create the topic the stack we're testing will reference.
    topics = [sns.Topic(topics_stack, "Topic1")]

    # Create the StateMachineStack.
    state_machine_stack = StateMachineStack(
        app, "StateMachineStack", topics=topics  # Cross-stack reference
    )

    # Prepare the stack for assertions.
    template = Template.from_stack(state_machine_stack)
```

Java

```
package software.amazon.samples.awscdkassertionssamples;

import org.junit.jupiter.api.Test;
import software.amazon.awscdk.assertions.Capture;
import software.amazon.awscdk.assertions.Match;
import software.amazon.awscdk.assertions.Template;
import software.amazon.awscdk.App;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.services.sns.Topic;

import java.util.*;

import static org.assertj.core.api.Assertions.assertThat;

public class StateMachineStackTest {
    @Test
    public void testSynthesizesProperly() {
        final App app = new App();

        // Since the StateMachineStack consumes resources from a separate stack
 (cross-stack references), we create a stack
        // for our SNS topics to live in here. These topics can then be passed to
 the StateMachineStack later, creating a
        // cross-stack reference.
        final Stack topicsStack = new Stack(app, "TopicsStack");

        // Create the topic the stack we're testing will reference.
        final List<Topic> topics =
 Collections.singletonList(Topic.Builder.create(topicsStack, "Topic1").build());

        // Create the StateMachineStack.
        final StateMachineStack stateMachineStack = new StateMachineStack(
                app,
                "StateMachineStack",
                topics // Cross-stack reference
        );

        // Prepare the stack for assertions.
        final Template template = Template.fromStack(stateMachineStack)
```

C#

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

using Amazon.CDK;
using Amazon.CDK.AWS.SNS;
using Amazon.CDK.Assertions;
using AwsCdkAssertionSamples;

using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
using StringDict = System.Collections.Generic.Dictionary<string, string>;

namespace TestProject1
{
    [TestClass]
    public class StateMachineStackTest
    {
        [TestMethod]
        public void TestMethod1()
        {
            var app = new App();

            // Since the StateMachineStack consumes resources from a separate stack
 (cross-stack references), we create a stack
            // for our SNS topics to live in here. These topics can then be passed
 to the StateMachineStack later, creating a
            // cross-stack reference.
            var topicsStack = new Stack(app, "TopicsStack");

            // Create the topic the stack we're testing will reference.
            var topics = new Topic[] { new Topic(topicsStack, "Topic1") };

            // Create the StateMachineStack.
            var StateMachineStack = new StateMachineStack(app, "StateMachineStack",
 new StateMachineStackProps
            {
                Topics = topics
            });

            // Prepare the stack for assertions.
            var template = Template.FromStack(stateMachineStack);

            // test will go here
        }
```

```
    }
}
```

Sekarang kita dapat menegaskan bahwa fungsi Lambda dan langganan Amazon SNS telah dibuat.

TypeScript

```
    // Assert it creates the function with the correct properties...
    template.hasResourceProperties("AWS::Lambda::Function", {
      Handler: "handler",
      Runtime: "nodejs14.x",
    });

    // Creates the subscription...
    template.resourceCountIs("AWS::SNS::Subscription", 1);
```

JavaScript

```
    // Assert it creates the function with the correct properties...
    template.hasResourceProperties("AWS::Lambda::Function", {
      Handler: "handler",
      Runtime: "nodejs14.x",
    });

    // Creates the subscription...
    template.resourceCountIs("AWS::SNS::Subscription", 1);
```

Python

```
# Assert that we have created the function with the correct properties
    template.has_resource_properties(
        "AWS::Lambda::Function",
        {
            "Handler": "handler",
            "Runtime": "nodejs14.x",
        },
    )

    # Assert that we have created a subscription
    template.resource_count_is("AWS::SNS::Subscription", 1)
```

Java

```
        // Assert it creates the function with the correct properties...
        template.hasResourceProperties("AWS::Lambda::Function", Map.of(
                "Handler", "handler",
                "Runtime", "nodejs14.x"
        ));

         // Creates the subscription...
        template.resourceCountIs("AWS::SNS::Subscription", 1);
```

C#

```
            // Prepare the stack for assertions.
            var template = Template.FromStack(stateMachineStack);

            // Assert it creates the function with the correct properties...
            template.HasResourceProperties("AWS::Lambda::Function", new StringDict {
                { "Handler", "handler"},
                { "Runtime", "nodejs14x" }
            });

            // Creates the subscription...
            template.ResourceCountIs("AWS::SNS::Subscription", 1);
```

Uji fungsi Lambda kami menegaskan bahwa dua properti tertentu dari sumber daya fungsi memiliki nilai tertentu. Secara default, `hasResourceProperties` metode melakukan kecocokan sebagian pada properti sumber daya seperti yang diberikan dalam CloudFormation template yang disintesis. Tes ini mengharuskan properti yang disediakan ada dan memiliki nilai yang ditentukan, tetapi sumber daya juga dapat memiliki properti lain, yang tidak diuji.

Pernyataan Amazon SNS kami menegaskan bahwa template yang disintesis berisi langganan, tetapi tidak ada tentang langganan itu sendiri. Kami menyertakan pernyataan ini terutama untuk menggambarkan bagaimana menegaskan jumlah sumber daya. `Template`Kelas menawarkan metode yang lebih spesifik untuk menulis pernyataan terhadap`Resources`,`Outputs`, dan `Mapping` bagian dari template. CloudFormation

Pencocokan

Perilaku pencocokan paral default `hasResourceProperties` dapat diubah menggunakan matcher dari kelas. `Match`

Matcher berkisar dari lunak () hingga ketat (`Match.anyValue`). `Match.objectEquals` Mereka dapat disarangkan untuk menerapkan metode pencocokan yang berbeda ke berbagai bagian properti sumber daya. Menggunakan `Match.objectEquals` dan `Match.anyValue` bersama-sama, misalnya, kita dapat menguji peran IAM mesin status lebih lengkap, sementara tidak memerlukan nilai spesifik untuk properti yang dapat berubah.

TypeScript

```
// Fully assert on the state machine's IAM role with matchers.
template.hasResourceProperties(
  "AWS::IAM::Role",
  Match.objectEquals({
    AssumeRolePolicyDocument: {
      Version: "2012-10-17",
      Statement: [
        {
          Action: "sts:AssumeRole",
          Effect: "Allow",
          Principal: {
            Service: {
              "Fn::Join": [
                "",
                ["states.", Match.anyValue(), ".amazonaws.com"],
              ],
            },
          },
        },
      ],
    },
  })
);
```

JavaScript

```
// Fully assert on the state machine's IAM role with matchers.
template.hasResourceProperties(
  "AWS::IAM::Role",
```

```
      Match.objectEquals({
        AssumeRolePolicyDocument: {
          Version: "2012-10-17",
          Statement: [
            {
              Action: "sts:AssumeRole",
              Effect: "Allow",
              Principal: {
                Service: {
                  "Fn::Join": [
                    "",
                    ["states.", Match.anyValue(), ".amazonaws.com"],
                  ],
                },
              },
            },
          ],
        },
      })
    );
```

Python

```
from aws_cdk.assertions import Match

    # Fully assert on the state machine's IAM role with matchers.
    template.has_resource_properties(
        "AWS::IAM::Role",
        Match.object_equals(
            {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Action": "sts:AssumeRole",
                            "Effect": "Allow",
                            "Principal": {
                                "Service": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "states.",
                                            Match.any_value(),
```

```
                                        ".amazonaws.com",
                                    ],
                                ],
                            },
                        },
                    },
                ],
            },
        }
    ),
)
```

Java

```
        // Fully assert on the state machine's IAM role with matchers.
        template.hasResourceProperties("AWS::IAM::Role", Match.objectEquals(
                Collections.singletonMap("AssumeRolePolicyDocument", Map.of(
                        "Version", "2012-10-17",
                        "Statement", Collections.singletonList(Map.of(
                                "Action", "sts:AssumeRole",
                                "Effect", "Allow",
                                "Principal", Collections.singletonMap(
                                        "Service", Collections.singletonMap(
                                                "Fn::Join", Arrays.asList(
                                                        "",
                                                        Arrays.asList("states.",
Match.anyValue(), ".amazonaws.com")
                                                )
                                        )
                                )
                        ))
                ))
        ));
```

C#

```
            // Fully assert on the state machine's IAM role with matchers.
            template.HasResource("AWS::IAM::Role", Match.ObjectEquals(new ObjectDict
            {
                { "AssumeRolePolicyDocument", new ObjectDict
                    {
                        { "Version", "2012-10-17" },
                        { "Action", "sts:AssumeRole" },
```

```
                        { "Principal", new ObjectDict
                            {
                                { "Version", "2012-10-17" },
                                { "Statement", new object[]
                                    {
                                        new ObjectDict {
                                            { "Action", "sts:AssumeRole" },
                                            { "Effect", "Allow" },
                                            { "Principal", new ObjectDict
                                                {
                                                    { "Service", new ObjectDict
                                                        {
                                                            { "", new object[]
                                                                { "states",
Match.AnyValue(), ".amazonaws.com" }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }));
```

Banyak CloudFormation sumber daya termasuk objek JSON serial yang direpresentasikan sebagai string. `Match.serializedJson()`Matcher dapat digunakan untuk mencocokkan properti di dalam JSON ini.

Misalnya, mesin status Step Functions didefinisikan menggunakan string dalam Amazon States Language berbasis JSON. Kita akan gunakan `Match.serializedJson()` untuk memastikan bahwa status awal kita adalah satu-satunya langkah. Sekali lagi, kita akan menggunakan pencocokan bersarang untuk menerapkan berbagai jenis pencocokan ke berbagai bagian objek.

TypeScript

```
    // Assert on the state machine's definition with the Match.serializedJson()
    // matcher.
```

```
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    // Match.objectEquals() is used implicitly, but we use it explicitly
    // here for extra clarity.
    Match.objectEquals({
      StartAt: "StartState",
      States: {
        StartState: {
          Type: "Pass",
          End: true,
          // Make sure this state doesn't provide a next state -- we can't
          // provide both Next and set End to true.
          Next: Match.absent(),
        },
      },
    })
  ),
});
```

JavaScript

```
// Assert on the state machine's definition with the Match.serializedJson()
// matcher.
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    // Match.objectEquals() is used implicitly, but we use it explicitly
    // here for extra clarity.
    Match.objectEquals({
      StartAt: "StartState",
      States: {
        StartState: {
          Type: "Pass",
          End: true,
          // Make sure this state doesn't provide a next state -- we can't
          // provide both Next and set End to true.
          Next: Match.absent(),
        },
      },
    })
  ),
});
```

Python

```
   # Assert on the state machine's definition with the serialized_json matcher.
   template.has_resource_properties(
       "AWS::StepFunctions::StateMachine",
       {
           "DefinitionString": Match.serialized_json(
               # Match.object_equals() is the default, but specify it here for
clarity
               Match.object_equals(
                   {
                       "StartAt": "StartState",
                       "States": {
                           "StartState": {
                               "Type": "Pass",
                               "End": True,
                               # Make sure this state doesn't provide a next state
--
                               # we can't provide both Next and set End to true.
                               "Next": Match.absent(),
                           },
                       },
                   }
               )
           ),
       },
   )
```

Java

```
       // Assert on the state machine's definition with the Match.serializedJson()
matcher.
       template.hasResourceProperties("AWS::StepFunctions::StateMachine",
Collections.singletonMap(
               "DefinitionString", Match.serializedJson(
                       // Match.objectEquals() is used implicitly, but we use it
explicitly here for extra clarity.
                       Match.objectEquals(Map.of(
                               "StartAt", "StartState",
                               "States", Collections.singletonMap(
                                       "StartState", Map.of(
                                               "Type", "Pass",
                                               "End", true,
```

```
                                                    // Make sure this state doesn't
provide a next state -- we can't provide
                                                    // both Next and set End to true.
                                                    "Next", Match.absent()
                                                )
                                    )
                                ))
                        )
                ));
```

C#

```
            // Assert on the state machine's definition with the
Match.serializedJson() matcher
            template.HasResourceProperties("AWS::StepFunctions::StateMachine", new
ObjectDict
            {
                { "DefinitionString", Match.SerializedJson(
                    // Match.objectEquals() is used implicitly, but we use it
explicitly here for extra clarity.
                    Match.ObjectEquals(new ObjectDict {
                        { "StartAt", "StartState" },
                        { "States", new ObjectDict
                        {
                            { "StartState", new ObjectDict {
                                { "Type", "Pass" },
                                { "End", "True" },
                                // Make sure this state doesn't provide a next state
-- we can't provide
                                // both Next and set End to true.
                                { "Next", Match.Absent() }
                            }}
                        }}
                    })
                )}});
```

Menangkap

Seringkali berguna untuk menguji properti untuk memastikan mereka mengikuti format tertentu, atau memiliki nilai yang sama dengan properti lain, tanpa perlu mengetahui nilai pastinya sebelumnya. `assertions`Modul ini menyediakan kemampuan ini di `Capture`kelasnya.

Dengan menentukan `Capture` instance di tempat nilai di`hasResourceProperties`, nilai itu dipertahankan dalam objek. `Capture` Nilai yang ditangkap sebenarnya dapat diambil menggunakan `as` metode objek, termasuk`asNumber()`,`asString()`, dan`asObject`, dan diuji. Gunakan `Capture` dengan matcher untuk menentukan lokasi yang tepat dari nilai yang akan ditangkap dalam properti sumber daya, termasuk properti JSON serial.

Contoh berikut menguji untuk memastikan bahwa status awal mesin status kami memiliki nama yang dimulai dengan`Start`. Ini juga menguji bahwa keadaan ini ada dalam daftar negara bagian dalam mesin.

TypeScript

```
// Capture some data from the state machine's definition.
const startAtCapture = new Capture();
const statesCapture = new Capture();
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    Match.objectLike({
      StartAt: startAtCapture,
      States: statesCapture,
    })
  ),
});

// Assert that the start state starts with "Start".
expect(startAtCapture.asString()).toEqual(expect.stringMatching(/^Start/));

// Assert that the start state actually exists in the states object of the
// state machine definition.
expect(statesCapture.asObject()).toHaveProperty(startAtCapture.asString());
```

JavaScript

```
// Capture some data from the state machine's definition.
const startAtCapture = new Capture();
const statesCapture = new Capture();
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    Match.objectLike({
      StartAt: startAtCapture,
      States: statesCapture,
    })
```

```
      ),
    });

    // Assert that the start state starts with "Start".
    expect(startAtCapture.asString()).toEqual(expect.stringMatching(/^Start/));

    // Assert that the start state actually exists in the states object of the
    // state machine definition.
    expect(statesCapture.asObject()).toHaveProperty(startAtCapture.asString());
```

Python

```
import re

    from aws_cdk.assertions import Capture

    # ...

    # Capture some data from the state machine's definition.
    start_at_capture = Capture()
    states_capture = Capture()
    template.has_resource_properties(
        "AWS::StepFunctions::StateMachine",
        {
            "DefinitionString": Match.serialized_json(
                Match.object_like(
                    {
                        "StartAt": start_at_capture,
                        "States": states_capture,
                    }
                )
            ),
        },
    )

    # Assert that the start state starts with "Start".
    assert re.match("^Start", start_at_capture.as_string())

    # Assert that the start state actually exists in the states object of the
    # state machine definition.
    assert start_at_capture.as_string() in states_capture.as_object()
```

Java

```
        // Capture some data from the state machine's definition.
        final Capture startAtCapture = new Capture();
        final Capture statesCapture = new Capture();
        template.hasResourceProperties("AWS::StepFunctions::StateMachine",
Collections.singletonMap(
                "DefinitionString", Match.serializedJson(
                        Match.objectLike(Map.of(
                                "StartAt", startAtCapture,
                                "States", statesCapture
                        ))
                )
        ));

        // Assert that the start state starts with "Start".
        assertThat(startAtCapture.asString()).matches("^Start.+");

        // Assert that the start state actually exists in the states object of the
state machine definition.
        assertThat(statesCapture.asObject()).containsKey(startAtCapture.asString());
```

C#

```
            // Capture some data from the state machine's definition.
            var startAtCapture = new Capture();
            var statesCapture = new Capture();
            template.HasResourceProperties("AWS::StepFunctions::StateMachine", new
ObjectDict
            {
                { "DefinitionString", Match.SerializedJson(
                    new ObjectDict
                    {
                        { "StartAt", startAtCapture },
                        { "States", statesCapture }
                    }
                )}
            });

            Assert.IsTrue(startAtCapture.ToString().StartsWith("Start"));

Assert.IsTrue(statesCapture.AsObject().ContainsKey(startAtCapture.ToString()));
```

Tes snapshot

Dalam pengujian snapshot, Anda membandingkan seluruh template yang disintesis dengan CloudFormation template baseline yang disimpan sebelumnya (sering disebut “master”). Tidak seperti pernyataan berbutir halus, pengujian snapshot tidak berguna dalam menangkap regresi. Ini karena pengujian snapshot berlaku untuk seluruh template, dan hal-hal selain perubahan kode dapat menyebabkan perbedaan kecil (atau not-so-small) dalam hasil sintesis. Perubahan ini bahkan mungkin tidak memengaruhi penerapan Anda, tetapi masih akan menyebabkan pengujian snapshot gagal.

Misalnya, Anda dapat memperbarui konstruksi CDK untuk menggabungkan praktik terbaik baru, yang dapat menyebabkan perubahan pada sumber daya yang disintesis atau bagaimana mereka diatur. Atau, Anda dapat memperbarui CDK Toolkit ke versi yang melaporkan metadata tambahan. Perubahan nilai konteks juga dapat mempengaruhi template yang disintesis.

Namun, tes snapshot dapat sangat membantu dalam refactoring, selama Anda mempertahankan semua faktor lain yang dapat memengaruhi templat yang disintesis. Anda akan segera tahu jika perubahan yang Anda buat telah mengubah template secara tidak sengaja. Jika perubahan disengaja, cukup terima template baru sebagai baseline.

Misalnya, jika kita memiliki `DeadLetterQueue` konstruksi ini:

TypeScript

```
export class DeadLetterQueue extends sqs.Queue {
  public readonly messagesInQueueAlarm: cloudwatch.IAlarm;

  constructor(scope: Construct, id: string) {
    super(scope, id);

    // Add the alarm
    this.messagesInQueueAlarm = new cloudwatch.Alarm(this, 'Alarm', {
      alarmDescription: 'There are messages in the Dead Letter Queue',
      evaluationPeriods: 1,
      threshold: 1,
      metric: this.metricApproximateNumberOfMessagesVisible(),
    });
  }
}
```

JavaScript

```
class DeadLetterQueue extends sqs.Queue {

  constructor(scope, id) {
    super(scope, id);

    // Add the alarm
    this.messagesInQueueAlarm = new cloudwatch.Alarm(this, 'Alarm', {
      alarmDescription: 'There are messages in the Dead Letter Queue',
      evaluationPeriods: 1,
      threshold: 1,
      metric: this.metricApproximateNumberOfMessagesVisible(),
    });
  }
}

module.exports = { DeadLetterQueue }
```

Python

```
class DeadLetterQueue(sqs.Queue):
    def __init__(self, scope: Construct, id: str):
        super().__init__(scope, id)

        self.messages_in_queue_alarm = cloudwatch.Alarm(
            self,
            "Alarm",
            alarm_description="There are messages in the Dead Letter Queue.",
            evaluation_periods=1,
            threshold=1,
            metric=self.metric_approximate_number_of_messages_visible(),
        )
```

Java

```
public class DeadLetterQueue extends Queue {
    private final IAlarm messagesInQueueAlarm;

    public DeadLetterQueue(@NotNull Construct scope, @NotNull String id) {
        super(scope, id);

        this.messagesInQueueAlarm = Alarm.Builder.create(this, "Alarm")
```

```
                .alarmDescription("There are messages in the Dead Letter Queue.")
                .evaluationPeriods(1)
                .threshold(1)
                .metric(this.metricApproximateNumberOfMessagesVisible())
                .build();
    }

    public IAlarm getMessagesInQueueAlarm() {
        return messagesInQueueAlarm;
    }
}
```

C#

```
namespace AwsCdkAssertionSamples
{
    public class DeadLetterQueue : Queue
    {
        public IAlarm messagesInQueueAlarm;

        public DeadLetterQueue(Construct scope, string id) : base(scope, id)
        {
            messagesInQueueAlarm = new Alarm(this, "Alarm", new AlarmProps
            {
                AlarmDescription = "There are messages in the Dead Letter Queue.",
                EvaluationPeriods = 1,
                Threshold = 1,
                Metric = this.MetricApproximateNumberOfMessagesVisible()
            });
        }
    }
}
```

Kita bisa mengujinya seperti ini:

TypeScript

```
import { Match, Template } from "aws-cdk-lib/assertions";
import * as cdk from "aws-cdk-lib";
import { DeadLetterQueue } from "../lib/dead-letter-queue";

describe("DeadLetterQueue", () => {
```

```
  test("matches the snapshot", () => {
    const stack = new cdk.Stack();
    new DeadLetterQueue(stack, "DeadLetterQueue");

    const template = Template.fromStack(stack);
    expect(template.toJSON()).toMatchSnapshot();
  });
});
```

JavaScript

```
const { Match, Template } = require("aws-cdk-lib/assertions");
const cdk = require("aws-cdk-lib");
const { DeadLetterQueue } = require("../lib/dead-letter-queue");

describe("DeadLetterQueue", () => {
  test("matches the snapshot", () => {
    const stack = new cdk.Stack();
    new DeadLetterQueue(stack, "DeadLetterQueue");

    const template = Template.fromStack(stack);
    expect(template.toJSON()).toMatchSnapshot();
  });
});
```

Python

```
import aws_cdk_lib as cdk
from aws_cdk_lib.assertions import Match, Template

from app.dead_letter_queue import DeadLetterQueue

def snapshot_test():
    stack = cdk.Stack()
    DeadLetterQueue(stack, "DeadLetterQueue")

    template = Template.from_stack(stack)
    assert template.to_json() == snapshot
```

Java

```
package software.amazon.samples.awscdkassertionssamples;
```

```
import org.junit.jupiter.api.Test;
import au.com.origin.snapshots.Expect;
import software.amazon.awscdk.assertions.Match;
import software.amazon.awscdk.assertions.Template;
import software.amazon.awscdk.Stack;

import java.util.Collections;
import java.util.Map;

public class DeadLetterQueueTest {
    @Test
    public void snapshotTest() {
        final Stack stack = new Stack();
        new DeadLetterQueue(stack, "DeadLetterQueue");

        final Template template = Template.fromStack(stack);
        expect.toMatchSnapshot(template.toJSON());
    }
}
```

C#

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

using Amazon.CDK;
using Amazon.CDK.Assertions;
using AwsCdkAssertionSamples;

using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
using StringDict = System.Collections.Generic.Dictionary<string, string>;

namespace TestProject1
{
    [TestClass]
    public class StateMachineStackTest

    [TestClass]
    public class DeadLetterQueueTest
    {
    [TestMethod]
        public void SnapshotTest()
        {
```

```
            var stack = new Stack();
            new DeadLetterQueue(stack, "DeadLetterQueue");

            var template = Template.FromStack(stack);

            return Verifier.Verify(template.ToJSON());
        }
    }
}
```

Kiat untuk tes

Ingat, tes Anda akan hidup selama kode yang mereka uji, dan mereka akan dibaca dan dimodifikasi sesering mungkin. Oleh karena itu, perlu waktu sejenak untuk mempertimbangkan cara terbaik untuk menulisnya.

Jangan salin dan tempel baris penyiapan atau pernyataan umum. Sebaliknya, refaktorkan logika ini menjadi perlengkapan atau fungsi pembantu. Gunakan nama baik yang mencerminkan apa yang sebenarnya diuji oleh setiap tes.

Jangan mencoba melakukan terlalu banyak dalam satu tes. Lebih disukai, tes harus menguji satu dan hanya satu perilaku. Jika Anda secara tidak sengaja merusak perilaku itu, tepat satu tes akan gagal, dan nama tes akan memberi tahu Anda apa yang gagal. Namun, ini lebih ideal untuk diperjuangkan; terkadang Anda pasti akan (atau tidak sengaja) menulis tes yang menguji lebih dari satu perilaku. Tes snapshot, untuk alasan yang telah kami jelaskan, terutama rentan terhadap masalah ini, jadi gunakan dengan hemat.

Keamanan untuk AWS Cloud Development Kit (AWS CDK)

Keamanan cloud di Amazon Web Services (AWS) merupakan prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan. Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model Tanggung Jawab Bersama menggambarkan ini sebagai Keamanan dari Cloud dan Keamanan dalam Cloud.

Security of the Cloud - AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan semua layanan yang ditawarkan di AWS Cloud dan memberi Anda layanan yang dapat Anda gunakan dengan aman. Tanggung jawab keamanan kami adalah prioritas tertinggi di AWS, dan efektivitas keamanan kami secara teratur diuji dan diverifikasi oleh auditor pihak ketiga sebagai bagian dari Program AWS Kepatuhan.

Keamanan di Cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan, dan faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

AWS CDK Berikut model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Topik

Identitas dan manajemen akses untuk AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya. AWS IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan. AWS

Pengguna layanan — Jika Anda menggunakan Layanan AWS untuk melakukan pekerjaan Anda, maka administrator Anda memberi Anda kredensi dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak AWS fitur untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda.

Administrator layanan — Jika Anda bertanggung jawab atas sumber AWS daya di perusahaan Anda, Anda mungkin memiliki akses penuh ke AWS sumber daya. Tugas Anda adalah menentukan sumber daya Layanan AWS dan mana yang harus diakses pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep Basic IAM.

Administrator IAM – Jika Anda adalah administrator IAM, Anda mungkin ingin belajar dengan lebih detail tentang cara Anda menulis kebijakan untuk mengelola akses ke Layanan AWS.

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensil yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensi Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas gabungan, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Untuk mengakses AWS secara terprogram, AWS sediakan AWS CDK, perangkat pengembangan perangkat lunak (SDK), dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda

secara kriptografis menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang menggunakan metode yang disarankan untuk menandatangani permintaan sendiri, lihat proses penandatanganan Versi Tanda Tangan 4 di Referensi Umum AWS.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor dalam Panduan Pengguna AWS IAM Identity Center dan Menggunakan autentikasi multi-faktor (MFA) di AWS dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari Anda. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya andalkan kredensial sementara daripada membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami sarankan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Rotasikan kunci akses secara rutin untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan kumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin untuk beberapa pengguna sekaligus. Grup membuat izin lebih mudah dikelola untuk sekelompok besar pengguna. Misalnya, Anda dapat memiliki grup yang bernama IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, silakan lihat Kapan harus membuat pengguna IAM (bukan peran) dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil peran IAM untuk sementara AWS Management Console dengan beralih peran. Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat Menggunakan peran IAM dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna gabungan – Untuk menetapkan izin ke sebuah identitas gabungan, Anda dapat membuat peran dan menentukan izin untuk peran tersebut. Saat identitas terfederasi mengautentikasi, identitas tersebut akan dikaitkan dengan peran dan diberi izin yang ditentukan oleh peran tersebut. Untuk informasi tentang peran-peran untuk federasi, lihat Membuat peran untuk Penyedia Identitas pihak ketiga dalam Panduan Pengguna IAM. Jika Anda menggunakan Pusat Identitas IAM, Anda perlu mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses

identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM mengorelasikan izin yang diatur ke peran dalam IAM. Untuk informasi tentang rangkaian izin, lihat Rangkaian izin dalam Panduan Pengguna AWS IAM Identity Center .

- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.
- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (pengguna utama tepercaya) dengan akun berbeda untuk mengakses sumber daya yang ada di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara kebijakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Bagaimana peran IAM berbeda dari kebijakan berbasis sumber daya dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Sebagai contoh, ketika Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.
  - Peran layanan – Peran layanan adalah peran IAM yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, memodifikasi, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
  - Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.
- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada instans EC2 dan membuat atau permintaan API. AWS CLI AWS Cara ini lebih dianjurkan daripada menyimpan kunci akses dalam instans EC2. Untuk menetapkan AWS peran ke instans EC2 dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instans berisi peran dan memungkinkan program yang berjalan di instans EC2 mendapatkan kredensial sementara. Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di instans Amazon EC2 dalam Panduan Pengguna IAM.

Untuk mempelajari apakah kita harus menggunakan peran IAM atau pengguna IAM, lihat Kapan harus membuat peran IAM (bukan pengguna) dalam Panduan Pengguna IAM.

Validasi kepatuhan untuk AWS Cloud Development Kit (AWS CDK)

AWS CDK Berikut model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Keamanan dan kepatuhan AWS layanan dinilai oleh auditor pihak ketiga sebagai bagian dari beberapa program AWS kepatuhan. Ini termasuk SOC, PCI, FedRAMP, HIPAA, dan lainnya. AWS menyediakan daftar AWS layanan yang sering diperbarui dalam lingkup program kepatuhan khusus di AWS Layanan dalam Lingkup oleh Program Kepatuhan.

Laporan audit pihak ketiga tersedia untuk Anda unduh menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifact.

Untuk informasi selengkapnya tentang program AWS kepatuhan, lihat Program AWS Kepatuhan.

Tanggung jawab kepatuhan Anda saat menggunakan AWS layanan AWS CDK untuk mengakses layanan ditentukan oleh sensitivitas data Anda, tujuan kepatuhan organisasi Anda, dan hukum dan peraturan yang berlaku. Jika penggunaan AWS layanan Anda tunduk pada kepatuhan terhadap standar seperti HIPAA, PCI, atau FedRAMP, menyediakan sumber daya untuk membantu: AWS

- Panduan Memulai Cepat Keamanan dan Kepatuhan — Panduan penerapan yang membahas pertimbangan arsitektur dan memberikan langkah-langkah untuk menerapkan lingkungan dasar yang berfokus pada keamanan dan berfokus pada kepatuhan. AWS
- AWS Sumber Daya Kepatuhan — Kumpulan buku kerja dan panduan yang mungkin berlaku untuk industri dan lokasi Anda.
- AWS Config – Layanan yang menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS yang membantu Anda memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik.

Ketahanan untuk AWS Cloud Development Kit (AWS CDK)

Infrastruktur global Amazon Web Services (AWS) dibangun di sekitar AWS Wilayah dan Zona Ketersediaan.

AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan.

Dengan Zona Ketersediaan, Anda dapat merancang dan mengoperasikan aplikasi dan basis data yang melakukan secara otomatis pinda saat gagal/failover di antara zona-zona tanpa terputus. Zona Ketersediaan lebih sangat tersedia, lebih toleran kesalahan, dan lebih dapat diskalakan daripadai infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

AWS CDK Berikut model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Infrastruktur keamanan untuk AWS Cloud Development Kit (AWS CDK)

AWS CDK Berikut model tanggung jawab bersama melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat halaman dokumentasi keamanan AWS layanan dan AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan.

Memecahkan masalah umum AWS CDK

Topik ini menjelaskan cara memecahkan masalah berikut dengan. AWS CDK

Setelah memperbarui AWS CDK, AWS CDK Toolkit (CLI) melaporkan ketidakcocokan dengan Construct Library AWS

Versi AWS CDK Toolkit (yang menyediakan `cdk` perintah) harus setidaknya sama dengan versi modul AWS Construct Library utama,. `aws-cdk-lib` Toolkit dimaksudkan agar kompatibel ke belakang. Versi 2.x terbaru dari toolkit dapat digunakan dengan rilis pustaka 1.x atau 2.x apa pun. Untuk alasan ini, kami sarankan Anda menginstal komponen ini secara global dan tetap up to date.

```
npm update -g aws-cdk
```

Jika Anda perlu bekerja dengan beberapa versi AWS CDK Toolkit, instal versi toolkit tertentu secara lokal di folder proyek Anda.

Jika Anda menggunakan TypeScript atau JavaScript, direktori proyek Anda sudah berisi salinan lokal berversi CDK Toolkit.

Jika Anda menggunakan bahasa lain, gunakan `npm` untuk menginstal AWS CDK Toolkit, menghilangkan `-g` bendera dan menentukan versi yang diinginkan. Sebagai contoh:

```
npm install aws-cdk@2.0
```

Untuk menjalankan AWS CDK Toolkit yang diinstal secara lokal, gunakan perintah `npx aws-cdk` bukan hanya. `cdk` Sebagai contoh:

```
npx aws-cdk deploy MyStack
```

`npx aws-cdk`menjalankan versi lokal AWS CDK Toolkit jika ada. Ini kembali ke versi global ketika sebuah proyek tidak memiliki instalasi lokal. Anda mungkin merasa nyaman untuk mengatur alias shell untuk memastikan selalu `cdk` dipanggil dengan cara ini.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

(kembali ke daftar)

Saat menerapkan AWS CDK tumpukan saya, saya menerima kesalahan **`NoSuchBucket`**

AWS Lingkungan Anda belum di-bootstrap, sehingga tidak memiliki bucket Amazon S3 untuk menyimpan sumber daya selama penerapan. Anda dapat membuat bucket pementasan dan sumber daya lain yang diperlukan dengan perintah berikut:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

Untuk menghindari menghasilkan AWS muatan yang tidak terduga, AWS CDK tidak secara otomatis mem-bootstrap lingkungan apa pun. Anda harus secara eksplisit mem-bootstrap setiap lingkungan yang akan Anda gunakan.

Secara default, sumber daya bootstrap dibuat di Wilayah atau Wilayah yang digunakan oleh tumpukan dalam AWS CDK aplikasi saat ini. Atau, mereka dibuat di Wilayah yang ditentukan dalam AWS profil lokal Anda (ditetapkan oleh`aws configure`), menggunakan akun profil itu. Anda dapat menentukan akun dan Wilayah yang berbeda pada baris perintah sebagai berikut. (Anda harus menentukan akun dan Wilayah jika Anda tidak berada di direktori aplikasi.)

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

Untuk informasi selengkapnya, lihat the section called “Bootstrapping”.

(kembali ke daftar)

Saat menerapkan AWS CDK tumpukan saya, saya menerima pesan **`forbidden: null`**

Anda menerapkan tumpukan yang membutuhkan sumber daya bootstrap, tetapi menggunakan peran IAM atau akun yang tidak memiliki izin untuk menulis ke sana. (Bucket staging digunakan saat menerapkan tumpukan yang berisi aset atau yang mensintesis AWS CloudFormation templat yang lebih besar dari 50K.) Gunakan akun atau peran yang memiliki izin untuk melakukan tindakan `s3:*` terhadap bucket yang disebutkan dalam pesan kesalahan.

(kembali ke daftar)

Saat mensintesis AWS CDK tumpukan, saya mendapatkan pesannya **`--app is required either in command-line, in cdk.json or in ~/.cdk.json`**

Pesan ini biasanya berarti bahwa Anda tidak berada di direktori utama AWS CDK proyek Anda ketika Anda mengeluarkan`cdk synth`. File `cdk.json` dalam direktori ini, yang dibuat oleh `cdk init` perintah, berisi baris perintah yang diperlukan untuk menjalankan (dan dengan demikian mensintesis) AWS CDK aplikasi Anda. Untuk TypeScript aplikasi, misalnya, `cdk.json` defaultnya terlihat seperti ini:

```
{
  "app": "npx ts-node bin/my-cdk-app.ts"
}
```

Sebaiknya Anda mengeluarkan `cdk` perintah hanya di direktori utama proyek Anda, sehingga AWS CDK toolkit dapat menemukannya di `cdk.json` sana dan berhasil menjalankan aplikasi Anda.

Jika ini tidak praktis karena alasan tertentu, AWS CDK Toolkit mencari baris perintah aplikasi di dua lokasi lain:

- `cdk.json`Di direktori home Anda
- Pada `cdk synth` perintah itu sendiri menggunakan `-a` opsi

Misalnya, Anda mungkin mensintesis tumpukan dari TypeScript aplikasi sebagai berikut.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

(kembali ke daftar)

Saat mensintesis AWS CDK tumpukan, saya menerima kesalahan karena AWS CloudFormation template berisi terlalu banyak sumber daya

AWS CDK Menghasilkan dan menyebarkan AWS CloudFormation template. AWS CloudFormation memiliki batas keras pada jumlah sumber daya yang dapat dikandung oleh tumpukan. Dengan itu AWS CDK, Anda dapat berlari melawan batas ini lebih cepat dari yang Anda harapkan.

> **Note**
>
> Batas AWS CloudFormation sumber daya adalah 500 pada tulisan ini. Lihat AWS CloudFormation kuota untuk batas sumber daya saat ini.

AWS Konstruksi berbasis niat tingkat tinggi dari Perpustakaan Konstruksi secara otomatis menyediakan sumber daya tambahan apa pun yang diperlukan untuk pencatatan, manajemen kunci, otorisasi, dan tujuan lainnya. Misalnya, memberikan satu akses sumber daya ke sumber daya lain menghasilkan objek IAM apa pun yang diperlukan untuk layanan yang relevan untuk berkomunikasi.

Dalam pengalaman kami, penggunaan konstruksi berbasis niat di dunia nyata menghasilkan 1-5 AWS CloudFormation sumber daya per konstruksi, meskipun ini dapat bervariasi. Untuk aplikasi tanpa server, 5—8 AWS sumber daya per titik akhir API adalah tipikal.

Pola, yang mewakili tingkat abstraksi yang lebih tinggi, memungkinkan Anda mendefinisikan lebih banyak AWS sumber daya dengan kode yang lebih sedikit. AWS CDK Kode dalamthe section called “ECS”, misalnya, menghasilkan lebih dari 50 AWS CloudFormation sumber daya sambil mendefinisikan hanya tiga konstruksi!

Melebihi batas AWS CloudFormation sumber daya adalah kesalahan selama AWS CloudFormation sintesis. AWS CDK Masalah peringatan jika tumpukan Anda melebihi 80% dari batas. Anda dapat menggunakan batas yang berbeda dengan menyetel `maxResources` properti di tumpukan Anda, atau menonaktifkan validasi dengan menyetel `maxResources` ke 0.

> **Tip**
>
> Anda bisa mendapatkan jumlah yang tepat dari sumber daya dalam output yang disintesis menggunakan skrip utilitas berikut. (Karena setiap AWS CDK pengembang membutuhkan Node.js, skrip ditulis dalam JavaScript.)

```
// rescount.js - count the resources defined in a stack
// invoke with: node rescount.js <path-to-stack-json>
// e.g. node rescount.js cdk.out/MyStack.template.json

import * as fs from 'fs';
const path = process.argv[2];

if (path) fs.readFile(path, 'utf8', function(err, contents) {
  console.log(err ? `${err}` :
  `${Object.keys(JSON.parse(contents).Resources).length} resources defined in
 ${path}`);
}); else console.log("Please specify the path to the stack's output .json
 file");
```

Saat jumlah sumber daya tumpukan Anda mendekati batas, pertimbangkan untuk merancang ulang untuk mengurangi jumlah sumber daya yang terkandung dalam tumpukan Anda: misalnya, dengan menggabungkan beberapa fungsi Lambda, atau dengan memecah tumpukan Anda menjadi beberapa tumpukan. CDK mendukung referensi antar tumpukan, sehingga Anda dapat memisahkan fungsionalitas aplikasi menjadi tumpukan yang berbeda dengan cara apa pun yang paling masuk akal bagi Anda.

> **Note**
>
> AWS CloudFormation para ahli sering menyarankan penggunaan tumpukan bersarang sebagai solusi untuk batas sumber daya. AWS CDK Mendukung pendekatan ini melalui `NestedStack`konstruksi.

(kembali ke daftar)

Saya menentukan tiga (atau lebih) Availability Zone untuk grup Auto Scaling atau VPC saya, tetapi hanya diterapkan dalam dua

Untuk mendapatkan jumlah Availability Zone yang Anda minta, tentukan akun dan Region di `env` properti stack. Jika Anda tidak menentukan keduanya, secara default AWS CDK, mensintesis tumpukan sebagai agnostik lingkungan. Anda kemudian dapat menerapkan tumpukan ke Wilayah tertentu menggunakan AWS CloudFormation. Karena beberapa Wilayah hanya memiliki dua Availability Zone, template agnostik lingkungan tidak menggunakan lebih dari dua.

> **Note**
>
> Di masa lalu, Wilayah sesekali diluncurkan dengan hanya satu Availability Zone. AWS CDK Tumpukan agnostik lingkungan tidak dapat digunakan ke Wilayah tersebut. Namun, pada tulisan ini, semua AWS Wilayah memiliki setidaknya dua AZ.

Anda dapat mengubah perilaku ini dengan mengganti properti stack `availabilityZones`(Python`availability_zones`:) Anda untuk secara eksplisit menentukan zona yang ingin Anda gunakan.

Untuk informasi selengkapnya tentang menentukan akun dan wilayah tumpukan pada waktu sintesis, sambil mempertahankan fleksibilitas untuk menerapkan ke wilayah mana pun, lihat. the section called “Lingkungan”

(kembali ke daftar)

Bucket S3 saya, tabel DynamoDB, atau sumber daya lainnya tidak dihapus saat saya mengeluarkan **`cdk destroy`**

Secara default, sumber daya yang dapat berisi data pengguna memiliki properti `removalPolicy` (Python:`removal_policy`)`RETAIN`, dan sumber daya tidak dihapus ketika tumpukan dihancurkan. Sebaliknya, sumber daya menjadi yatim piatu dari tumpukan. Anda kemudian harus menghapus sumber daya secara manual setelah tumpukan dihancurkan. Sampai Anda melakukannya, penempatan kembali tumpukan gagal. Ini karena nama sumber daya baru yang dibuat selama penyebaran bertentangan dengan nama sumber daya yatim piatu.

Jika Anda menyetel kebijakan penghapusan sumber daya ke`DESTROY`, sumber daya tersebut akan dihapus saat tumpukan dihancurkan.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
```

```
import { Construct } from 'constructs';
import * as s3 from 'aws-cdk-lib/aws-s3';

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

    const bucket = new s3.Bucket(this, 'Bucket', {
      removalPolicy: cdk.RemovalPolicy.DESTROY,
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const s3 = require('aws-cdk-lib/aws-s3');

class CdkTestStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const bucket = new s3.Bucket(this, 'Bucket', {
      removalPolicy: cdk.RemovalPolicy.DESTROY
    });
  }
}

module.exports = { CdkTestStack }
```

Python

```
import aws_cdk as cdk
from constructs import Construct
import aws_cdk.aws_s3 as s3

class CdkTestStack(cdk.stack):
    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        bucket = s3.Bucket(self, "Bucket",
            removal_policy=cdk.RemovalPolicy.DESTROY)
```

Java

```
software.amazon.awscdk.*;
import software.amazon.awscdk.services.s3.*;
import software.constructs;

public class CdkTestStack extends Stack {
    public CdkTestStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public CdkTestStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        Bucket.Builder.create(this, "Bucket")
                .removalPolicy(RemovalPolicy.DESTROY).build();
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.S3;

public CdkTestStack(Construct scope, string id, IStackProps props) : base(scope, id,
 props)
{
    new Bucket(this, "Bucket", new BucketProps {
        RemovalPolicy = RemovalPolicy.DESTROY
    });
}
```

> **Note**
>
> AWS CloudFormation tidak dapat menghapus bucket Amazon S3 yang tidak kosong. Jika Anda menyetel kebijakan penghapusan bucket Amazon S3 keDESTROY, dan berisi data, upaya menghancurkan tumpukan akan gagal karena bucket tidak dapat dihapus. Anda dapat AWS CDK menghapus objek di ember sebelum mencoba menghancurkannya dengan menyetel `autoDeleteObjects` penyangga ember ke. `true`

(kembali ke daftar)

Kunci OpenPGP untuk dan jsii AWS CDK

Topik ini berisi kunci OpenPGP saat ini dan historis untuk dan jsii AWS CDK .

Kunci saat ini

Kunci ini harus digunakan untuk memvalidasi rilis saat ini dari AWS CDK dan jsii.

AWS CDK Kunci OpenPGP

ID kunci:	0x42B9CF2286CD987A
Jenis:	RSA
Ukuran :	4096/4096
Dibuat:	2022-07-05
Kedaluwarsa:	2026-07-04
ID Pengguna:	AWS Cloud Development Kit < aws-cdk@amazon.com >
Sidik jari kunci:	69B5 2D5B A295 1D11 FA65 413B 42B9 CF22 86CD 987A

Pilih ikon “Salin” untuk menyalin kunci OpenPGP berikut:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGLEgOsBEADCoAMwvnszMLybJ+AD9cHhVyX6+rYIUEXYSgVnfkl6Z7qawIwv
wgd/a5fEs9Kiz2XJmfwS9Rxb4d+0+Y1ls1A+gnpw9FMLcZlqkC9KLnS2MqvuXWLB
t3z4kjZaL9fQ+58PoD4gy/M2hDg6gZrYqR3gtJuw8FcFpb/1KlkzRQUM8eAMFxf2
TyfjP0V0tSHwcB+84oushX7fUXVMyc3+OHsCPOe/WBFMIlWgKA+n33JKIQlUUC8f
kCWBAsAFupil0lCveT6mZu5slNRlc1I3iBLjUZ3/MtLygfqAMKwUVXeawtDvRIZe
PrAFc2NyODEhly2JG6K0FW7eIcvBqR3rg8U49t9Y74ELTM0kKnfd+flvq35xWqQC
0zghnk3kDppRTN4zWBgTKiCMxBcsHXGOoGn57t4B9VY9Zy3vkeySigeiwl/Tw9nJ
PE0SRnwEc/HnjTTfX+GTG1aQVE0xSVyZ4m5ymRNCu6+rNH8lKwo5FujlXJ+GXPkp
```

```
qT+Lx6Ix/Ny7PaoweWxwtZUkLRS4pWUsg0yotZrGyIbS+X3yMEG8WBTFI9hf6HTq
0ryfi5/TsBrdrGKqWB99EC9xYEGgtHp4fKO5X0ynOagVOhf0jSe8t1uyuJPGb2Gc
MQagSys5xMhdG/ZnEY4Cb+JDtH/4jc3tca0+4Z5RQ7kF9IhCncFtrbjJbwARAQAB
tC5BV1MgQ2xvdWQgRGV2ZWxvcG1lbnQgS2l0IDxhd3MtY2RrQGFtYXpvbi5jb20+
iQI/BBMBAgApBQJixIDrAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYCAwECHgEC
F4AACgkQQrnPIobNmHo2qg//Zt9p/kN1DevflzxWKouUX0AS7UmUtRYXu5k/EEbu
wkYNHpUr7+lZ+Me5YyjcIpt6UwuG9cW4SvwuxIfXucyKAWiwEbydCQauvnrYDxDa
J6Yr/ntk7Sii6An9re99qic3IsvX+xlUXh+qJ/34ooP/1PHziCMqykvW/DwAIyhx
2qvTXy+9+OlOWSUbhkCnNz5XKb4XQGq73DqalZX1nH4dG6fckZmYRX+dpw2njfTw
ZLdZ7bkrfiL84FI4A21RfSbEU4s4ngiV17lZ9ivilBKTbDv3da7+yc919M7C5N4J
yrlxvtyYNDoqKAD2WYZAnpEbG/shu3f56RyOJd56tXGwl9nKPh+F9y+379XthSwA
xZTURFtjWf7wWHaDZadU0DKi+Oeeszjg2f/VJaGmmS8PIg7q6GiSHHpqHqNvACHm
ZXMw12QFd3qt3xu0JMmE11ZC5VBgblwpkQTrO04Sq1rOpJwXI9ODMS/ZEhAIoYmT
OR7ouknlAx6mj9fwpavWDAAJHLdVUMYBZTXiQYFzDvx51ivvTRWkB1zTJcFdqShY
B37+Jz2jLDNdMrcHk2yfVp/VvfbxKcexg8wEwrrtQUslTUenl5jBZJouoz/wW81s
Y4U1nCPCdTK5/C7JCKzR2gVnCpe6uaxAWkkM2feQhjqJZkTC4cFVgBT+4M6WcT1r
yq4=
=ahbs
-----END PGP PUBLIC KEY BLOCK-----
```

kunci OpenPGP jsii

ID kunci:	0x056C4E15DAE3D8D9
Jenis:	RSA
Ukuran :	4096/4096
Dibuat:	2022-07-05
Kedaluwarsa:	2026-07-04
ID Pengguna:	AWS JSII Tim < aws-jsii@amazon.com >
Sidik jari kunci:	1E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

Pilih ikon “Salin” untuk menyalin kunci OpenPGP berikut:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGLEgOkBEAD27EPVG9g2mHQ3+M6tF6le+tfhARJ2EV7m7NKIrTdSlCZATLWn
AVLlxG1unW34NlkKZbcbR86gAxRnnAhuEhPuloU/S5wAqPGbRiFl58YjYZDNJw6U
1SSMpE4O1sfjxv9yAbiRihLYtvksyHHZmaDhYner2aK1PdeWu+BKq/tjfm3Yzsd2
uuVEduJ72YoQk/29dEiGOHfT+2kUKxUX+0tJSJ9MGlEf4NtQE4WLzrT6Xqb2SG4+
alIiIVxIEi0XKDn7n8ZLjFwfJwOYxVYLtEUkqFWM8e8vgoc9/nYc+vDXZVED2g3Z
FWrwSnDSXbQpnMa2cLhD4xLpDHUS3i2p7r3dkJQGLo/5JGOopLibrOAbYZ72izhu
H/TuPFogSz0mNFPglrWdnLF04UIjIq420+06V4WQZC9n55Zjcbki/OhnC3B9pAdU
tiy8zg070bWq45dPGf5STkPPn7G8A2zmKefyO51iLIi26ZzW78siB+FvcGRhdg25
39sHJ1cmrTeC+B+k4KeV5sQ/m3UucimrZnk1xdaiVp8mWzRqWb8bB6Rs8K9RMrMV
tFBOKOBAT2QxOQtRGAantVgm193E1T1cmNpD0FKAKkDdPs64rKBEwFiHxccXHbah
eMd1weVwn3AKFD6uAm8ZRMV+dyssfcQxqpo/kfT1XpA6cQeOmGDOcKBfdwARAQAB
tCNBV1MgSlNJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQIAKQUC
YsSA6QIbLwUJB4TOAAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEAVsThXa
49jZjU4QANoyqOJUT4gRrXshE3N0mW5Ad4i8Ke09GA62HyvTtfbsA+2nkNVGJpXm
sFMzdaFO95Q65RkLS9vW4nhhjXBEc2XYNCt2AnARudA/41ykjDPwU112z9ZTB9he
y4ItIeNGpHvMWr51fihl0y2nkpODOBeiv44jscLbHyOmZfki1f5fuIu2U2IbUGK3
5FtYyeHcgRHnpYkzLuzK4PfayOywqQPJ7M9DWrHf+v5Cu4ZCZDOIKfzF+ew7MWwc
6KaoWHCYbFpX8jxFppbGsSFOQ8Sl2quoP0TLz9Wsq70Khi6C2P8JI6lm0HRLO+1M
jFbQxNOwAcN3k4HSwunAjXBlmT/6oc1RsdBdpXBaZ2AWseIXwSYZqNXp+5L179uZ
vSiD3DSSUqLJbdQRVOsJi3/87V5QU59byq2dToHveRjtSbVnK0TkTx9ZlgkcpjvM
BwHNqWhratV6af2Upjq2YQ0fdSB42f3pgopInxNJPMvlAb+cCfr0Pfwu7ge7UooQ
WHTxbpCvwtn/HNctMGpWscO02WsWgoYVjnVFay/XphE77pQ9rRUkhMe6VKXfxj/n
OCZJKrydluIIwR8vvONNqO+QwZ1xDEhO7MaSZlOm1AuUZIXFPgaWQkPZHKiiwFA/
QWnL/+shuRtMH2geTjkev198Jgb5HyXFm4SyYtZferQROyliEhik
=BuGv
-----END PGP PUBLIC KEY BLOCK-----
```

Kunci sejarah

Kunci ini dapat digunakan untuk memvalidasi rilis AWS CDK dan jsii sebelum 2022-07-05.

> **Important**
>
> Kunci baru dibuat sebelum yang sebelumnya kedaluwarsa. Akibatnya, pada saat tertentu, lebih dari satu kunci mungkin valid. Kunci digunakan untuk menandatangani artefak mulai hari pembuatannya, jadi gunakan kunci yang baru-baru ini diterbitkan di mana validitas kunci tumpang tindih.

AWS CDK Kunci OpenPGP (2022-04-07)

> **Note**
>
> Kunci ini tidak digunakan untuk menandatangani AWS CDK artefak setelah 2022-07-05.

ID kunci:	0x015584281F44A3C3
Jenis:	RSA
Ukuran :	4096/4096
Dibuat:	2022-04-07
Kedaluwarsa:	2026-04-06
ID Pengguna:	AWS Cloud Development Kit < aws-cdk@amazon.com >
Sidik jari kunci:	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

Pilih ikon “Salin” untuk menyalin kunci OpenPGP berikut:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGJPLgUBEADtlR5jQtxtBmROQvmWlPOViqqnJNhk0dULc3tXnq8NS/l6X81r
wHk+/CHG5kBunwvM0qaqLFRC6z9NnnNDxEHcTi47n+OAjWyDM6unxxWOPz8Dfaps
Uq/ZWa4by292ZeqRC9Ir2wdrizb69JbRjeshBwlJDAS/qtqCAqBRH/f7Zw7QSD6/
XTxyIy+KOVjZwFPFNHMRQ/NmgUc/Rfxsa0pUjk1YAj/AkvQlwwD8DEnASoBhO0DP
QonZxouLqIpgp4LsGo8TZdQv30ocIj0C9DuYUiUXWlCPlYPgDj6IWf3rgpMQ6nB9
wC9lx4t/L3Zg1HUD52y8aymndmbdHVn90mzlNg4XWyc58rioYrEk57YwbDnea/Kk
Hv4kVHZRfJ4/OFPyqs5ex1X3X6rb07VvA1tfLgPywO9XF2Xws8YWOWcEobaWTcnb
AzyVC6wKya8rEQzxkYJ6UkJlhDB6g6bZwIpsI2zlimG+kSBsyFvE2oRYMS0cXPqU
o+tX0+4TvxEyW3RrUQzQHIpqXrb0X1Q8Z2idPn5dwsipDEa4gsFXtrSXmbB/0Cee
eJVvKWQAsxol3+NE9L/yozq3cz5PWh0SSbmCLRcs78lMJ23MmzbMWV7BWC9DXdY+
TywY5IkDUPjGCKlD8VlrI3TgC222bH6qaua6LYCiTtRtvpDYuJNAlUjhawARAQAB
tC5BV1MgQ2xvdWQgRGV2ZWxvcG1lbnQgS2l0IDxhd3MtY2RrQGFtYXpvbi5jb20+
iQI/BBMBAgApBQJiTy4FAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYCAwECHgEC
```

```
F4AACgkQAVWEKB9Eo8NpbxAAiBF0kR/lVw3vuam60mk4l0iGMVsP8Xq6g/buzbEO
2MEB4Ftk04qOnoa+93S0ZiLR9PqxrwsGSp4ADDX3Vtc4uxwzUlKUi1ywEhQ1cwyL
YHQI3Hd75K1J81ozMEu6qJH+yF0TtTDZMeZHtH/XvuIYJW3Lx4o5ZFlsEegFPAgX
YCCpUS+k9qC6M8g2VjcltQJpyjGswsKm6FWaKHW+B9dfjdOHlImB9E2jaknJ8eoY
zb9zHgFANluMzpZ6rYVSiCuXiEgYmazQWCvlPcMOP7nX+1hq1z11LMqeSnfE09gX
H+OYho9cMEJkb1dzx1H9MRpylFIn9tL+2iCp4UPJjnqi6uawWyLZ2tp4G11haqQq
1yAh69u233I8GZKFUySzjHwH5qWGRgBTjrZ6FdcjSS2w/wMkVKuCPkWtdvo/TJrm
msCd1Reye8SEKYqrs0ujTwmlvWmUZmO06AdUjo1kWiBKeslTJrWEuG7Yk4pFOoA4
dsaq83gxpOJNVCh6M3y4DLNrvl7dhF95NwTWMROPj2otw7NIjF4/cdzve2+P7YNN
pVAtyCtTJdD3eZbQPVaL3T8cf1VGqt6++pnLGnWJ0+X3TyvfmTohdJvN3TE+tq7A
7cprDX/q9c56HaXdJzVpxEzuf/YC+JuYKeHwsX3QouDhyRg3PsigdZES/02Wr8so
l6U=
=MQI4
-----END PGP PUBLIC KEY BLOCK-----
```

kunci OpenPGP jsii (2022-04-07)

> **Note**
>
> Kunci ini tidak digunakan untuk menandatangani artefak jsii setelah 2022-07-05.

ID kunci:	0x985F5BC974B79356
Jenis:	RSA
Ukuran :	4096/4096
Dibuat:	2022-04-07
Kedaluwarsa:	2026-04-06
ID Pengguna:	AWS JSII Tim < aws-jsii@amazon.com >
Sidik jari kunci:	35A7 1785 8FA6 282D C5AC CD95 985F 5BC9 74B7 9356

Pilih ikon “Salin” untuk menyalin kunci OpenPGP berikut:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGJPLewBEADHH4TXup/gOlHrKDZRbj8MvsMTdM6eDteA6/c32UYV/YsK9rDA
jN8Jv/xlfosOebcHrfnFpHF9VTkmjuOpN695XdwMrW/NvlEPISTGEJf21x6ZTQ2r
1xWFYzC3sl3FZmvj9XAXTmygdv+XM3TqsFgZeCaBkZVdiLbQf+FhYrovUlgotb5D
YiCQI3ofV5QTE+141jhO5Pkd3ZIoBG+P826LaT8NXhwS0o1XqVk39DCZNoFshNmR
WFZpkVCTHyv5ZhVey1NWXnD8opO375htGNV4AeSmSIH9YkURD1g5F+2t7RiosKFo
kJrfPmUjhHn8IFpReGc8qmMMZX0WaV3t+VAWfOHGGyrXDfQ4xz1VCot75C2+qypM
+qhwOAOOP0zA7CfI96ULZzSH/j8HuQk3O0DsUCybpMuKEazEMxP3tgGtRerwDaFG
jQvAlK8Rbq3v8buBI6YJuXTwSzJE8KLjleUiTFumE6WP4rsAvlP/5rBvubeMfa3n
NIMm5Rkl36Z+jt3e2Z2ZqWDPpBRta8m7QHccrZhkvqu3YC3Gl6kdnm4Vio3Xfpg2
qtWhIQutQ6DmItewV+weQHas3hl88RPJtSrfWWIIMkpbF7Y4vbX9xcnsYCLlp2Mz
tWbbnU+EWATNSsufml/Kdnu9iEEuLmeovE11I69nwjNOq9P+GJ3r/FUb2wARAQAB
tCNBV1MgSlNJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQIAKQUC
Yk8t7AIbLwUJB4TOAAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEJhfW8l0
t5NWo64P/2y7gcMRylLLW/wbrCjton2O4+YRocwQxKm1cBml9FVDUR5967YczNuu
EwEOfH/Pu3UAlrBfKAfxPNhKchLwYiOBNh2Wk5UUxRcldNHTLb5jn5gxCeWNAsl/
Tc46qY+ObdBMdOf2Vu33UCOg83WLbg1bfBoA8Bm1cd0XObtLGucu606EBt1dBrKq
9UTcbJfuGivY2Xjy5r4kEiMHBoLKcFrSo2Mm7VtYlE4Mabjyj9+orqUio7qxOl60
aa7Psa6rMvs1Ip9IOrAdG7o5Y29tQpeINH0R1/u47BrlTEAgG63Dfy49w2h/1g0G
c9KPXVuN55OWRIu0hsiySDMk/2ERsF348TU3NURZltnCOxp6pHlbPJIxRVTNa9Cn
f8tbLB3y3HfA80516g+qwNYIYiqksDdV2bz+VbvmCWcO+FellDZli831gyMGa5JJ
rq7d0lEr6nqjcnKiVwItTQXyFYmKTAXweQtVC72g1sd3oZIyqa7T8pvhWpKXxoJV
WP+OPBhGg/JEVC9sguhuv53tzVwayrNwb54JxJsD2nemfhQm1Wyvb2bPTEaJ3mrv
mhPUvXZj/I9rgsEq3L/sm2Xjy09nra4o3oe3bhEL8nOj11wkIodi17VaGP0y+H3s
I5zB5UztS6dy+cH+J7DoRaxzVzq7qtH/ZY2quClt30wwqDHUX1ef
=+iYX
-----END PGP PUBLIC KEY BLOCK-----
```

AWS CDK Kunci OpenPGP (2018-06-19)

ID kunci:	0x0566A784E17F3870
Jenis:	RSA
Ukuran :	4096/4096
Dibuat:	2018-06-19
Kedaluwarsa:	2022-06-18
ID Pengguna:	AWS CDK Tim < aws-cdk@amazon.com >

Sidik jari kunci:	E88B E3B6 F0B1 E350 9E36 4F96 0566 A784 E17F 3870

Pilih ikon “Salin” untuk menyalin kunci OpenPGP berikut:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFsovE8BEADEFVCHeAVPvoQgsjVu9FPUczxy9P+2zGIT/MLI3/vPLiULQwRy
IN2oxyBNDtcDToNa/fTkW3Ev0NTP4V1h+uBoKDZD/p+dTmSDRfByECMI0sGZ3UsG
Ohhyl2Of44s0sL8gdLtDnqSRLf+ZrfT3gpgUnplW7VitkwLxr78jDpW4QD8p8dZ9
WNm3JgB55jyPgaJKqA1Ln4Vduni/1XkrG42nxrrU71uUdZPvPZ2ELLJa6n0/raG8
jq3le+xQh45gAIs6PGaAgy7jAsfbwkGTBHjjujITAY1DwvQH5iS31OaCM9n4JNpc
xGZeJAVYTLilznf2QtS/a50t+ZOmpq67Ssp2j6qYpiumm0Lo9q3K/R4/yF0FZ8SL
1TuNX0ecXEptiMVUfTiqrLsANg18EPtLZZOYW+ZkbcVytkDpiqj7bMwA7mI7zGCJ
1gjaTbcEmOmVdQYS1G6ZptwbTtvrgA6AfnZxX1HUxLRQ7tT/wvRtABfbQKAh85Ff
a3U9W4oC3c1MP5IyhNV1Wo8Zm0flZiZc0iZnojTtSG6UbcxNNL4Q8e08FWjhungj
yxSsIBnQO1Aeo1N4BbzlI+n9iaXVDUN7Kz1QEyS4PNpjvUyrUiQ+a9C5sRA7WP+x
IEOaBBGpoAXB3oLsdTNO6AcwcDd9+r2NlXlhWC4/uH2YHQUIegPqHmPWxwARAQAB
tCFBV1MgQ0RLIFRlYW0gPGF3cy1jZGtAYW1hem9uLmNvbT6JAj8EEwEIACkFAlso
vE8CGy8FCQeEzgAHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIXgAAKCRAFZqeE4X84
cLGxD/0XHnhoR2xvz38GM8HQlwlZy9W1wVhQKmNDQUavw8Zx7+iRR3m7nq3xM7Qq
BDbcbKSg1lVLSBQ6H2V6vRpysOhkPSH1nN2dO8DtvSKIPcxK48+1x7lmO+ksSs/+
oo1UvOmTDaRzOitYh3kOGXHHXk/l11GtF2FGQzYssX5iM4PHcjBsK1unThs56IMh
OJeZezEYzBaskTu/ytRJ236bPP2kZIEXfzAvhmTytuXWUXEftxOxc6fIAcYiKTha
aofG7WyR+Fvb1j5gNLcbY552QMxa23NZd5cSZH7468WEW1SGJ3AdLA7k5xvsPPOC
2YvQFD+vUOZ1JJuu6B5rHkiEMhRTLklkvqXEShTxuXiCp7iTOo6TBCmrWAT4eQr7
htLmqlXrgKi8qPkWmRdXXG+MQBzI/UyZq2q8KC6cx2md1PhANmeefhiM7FZZfeNM
WLonWfh8gVCsNH5h8WJ9fxsQCADd3Xxx3NelS2zDYBPRoaqZEEBbgUP6LnWFprA2
EkSlc/RoDqZCpBGgcoy1FFWvV/ZLgNU6OTQlYH6oYOWiylSJnaTDyurrktsxJI6d
4gdsFb6tqwTGecuUPvvZaEuvhWExLxAebhu780FdAPXgVTX+YCLI2zf+dWQvkFQf
80RE7ayn7BsiaLzFBVux/zz/WgvudsZX18r8tDiVQBL51ORmqw==
=0wuQ
-----END PGP PUBLIC KEY BLOCK-----
```

kunci OpenPGP jsii (2018-08-06)

ID kunci:	0x1c7ace4cb2a1b93a
Jenis:	RSA

Ukuran :	4096/4096
Dibuat:	2018-08-06
Kedaluwarsa:	2022-08-05
ID Pengguna:	AWS JSII Tim < aws-jsii@amazon.com >
Sidik jari kunci:	85EF 6522 4CE2 1E8C 72DB 28EC 1C7A CE4C B2A1 B93A

Pilih ikon “Salin” untuk menyalin kunci OpenPGP berikut:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFtoSs0BEAD6WweLD0B26h0F7Jo9iR6tVQ4PgQBK1Va5H/eP+A2Iqw79UyxZ
WNzHYhzQ5MjYYI1SgcPavXy5/LV1N8HJ7QzyKszybnLYpNTLPYArWE8ZM9ZmjvIR
p1GzwnVBGQfoOlxyeutE9T5ZkAn45dTS5jlno4unji4gHjnwXKf2nP1APU2CZfdK
8vDpLOgj9LeeGlerYNbx+7xtY/I+csFIQvK09FPLSNMJQLlkBhY0r6Rt9ZQG+653
tJn+AUjyM237w0UIX1IqyYc5IONXu8HklPGu0NYuX9AY/63Ak2Cyfj0w/PZlvueQ
noQNM3j0nkOEsTOEXCyaLQw9iBKpxvLnm5RjMSODDCkj8c9uu0LHr7J4EOtgt2S1
pem7Y/c/N+/Z+Ksg9fP8fVTfYwRPvdI1x2sCiRDfLoQSG9tdrN5VwPFi4sGV04sI
x7Al8Vf/OBjAGZrDaJgM/gVvb9SKAQUA6t3ofeP14gDrS0eYodEXZ+lamnxFglxF
Sn8NRC4JFNmkXSUaTNGUdFf//F0D69PRNT8CnFfmniGj0CphN5037PCA2LC/Buq2
3+K6mTPkCcCHYPC/SwItp/xIDAQsGuDc1i1SfDYXrjsK7uOuwC5jLA9X6wZ/jgXQ
4umRRJBAV1aW8b1+yfaYYCO2AfXXO6caObv8IvH7Pc4leC2DoqylD3KklQARAQAB
tCNBV1MgSlNJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQgAKQUC
W2hKzQIbLwUJB4TOAAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEBx6zkyy
obk6B34P/iNb5QjKyhT0glZiq1wK7tuDDRpR6fC/sp6Jd/GhaNjO4BzlDbUPSjW5
950VT+qwaHXbIma/QVP7EIRztfwWy7m8eOodjpiu7JyJprhwG9nocXiNsLADcMoH
BvabkDRWXWIWSurq2wbcFMlTVwxjHPIQs6kt2oojpzP985CDS/KTzyjow6/gfMim
DLdhSSbDUM34STEgew79L2sQzL7cvM/N59k+AGyEMHZDXHkEw/Bge5Ovz50YOnsp
lisH4BzPRIw7uWqPlkVPzJKwMuo2WvMjDfgbYLbyjfvs5mqDxT2GTwAx/rd2taU6
iSqP0QmLM54BtTVVdoVXZSmJyTmXAAGlITq8ECZ/coUW9K2pUSgVuWyu63lktFP6
MyCQYRmXPh9aSd4+ielteXM9Y39snlyLgEJBhMxioZXVO2oszwluPuhPoAp4ekwj
/umVsBf6As6PoAchg7Qzr+lRZGmV9YTJOgDn2Z7jf/7tOes0g/mdiXTQMSGtp/Fp
ggnifTBx3iXkrQhqHlwtam8XTHGHy3MvX17ZslNuB8Pjh+07hhCxv0VUVZPUHJqJ
ZsLa398LMteQ8UMxwJ3t06jwDWAd7mbr2tatIilLHtWWBFoCwBh1XLe/03ENCpDp
njZ7OsBsBK2nVVcN0H2v5ey0T1yE93o6r7xOwCwBiVp5skTCRUob
=2Tag
-----END PGP PUBLIC KEY BLOCK-----
```

AWS CDK Sejarah Panduan Pengembang

Lihat Rilis untuk informasi tentang AWS CDK rilis. AWS CDK Diperbarui kira-kira seminggu sekali. Versi pemeliharaan dapat dirilis antara rilis mingguan untuk mengatasi masalah kritis. Setiap rilis menyertakan AWS CDK Toolkit yang cocok (CDK CLI) AWS , Perpustakaan Konstruksi, dan Referensi API. Pembaruan pada Panduan ini umumnya tidak disinkronkan dengan AWS CDK rilis.

> **Note**
>
> Tabel di bawah ini mewakili tonggak dokumentasi yang signifikan. Kami memperbaiki kesalahan dan meningkatkan konten secara berkelanjutan.

Perubahan	Deskripsi	Tanggal
Tambahkan dokumentasi untuk fitur CDK Migrate	Gunakan AWS CDK CLI `cdk migrate` perintah untuk memigrasikan AWS sumber daya yang diterapkan, AWS CloudFormation tumpukan yang diterapkan, dan templat lokal ke. AWS CloudFormation AWS CDK Untuk informasi selengkapnya, lihat Migrasi ke AWS CDK.	Februari 2, 2024
Pembaruan praktik terbaik IAM	Panduan yang diperbarui untuk menyelaraskan dengan praktik terbaik IAM. Untuk informasi selengkapnya, lihat Praktik terbaik keamanan di IAM.	Maret 23, 2023
Dokumen `cdk.json`	Tambahkan dokumentasi nilai `cdk.json` konfigurasi.	20 April 2022

Manajemen ketergantungan	Tambahkan topik tentang mengelola dependensi dengan file. AWS CDK	7 April 2022
Hapus tanda kurung ganda dari contoh Java	Ganti anti-pola ini dengan Java 9 secara `Map.of` keseluruhan.	9 Maret 2022
AWS CDK rilis v2	Versi 2 dari Panduan AWS CDK Pengembang dirilis. Riwayat dokumen untuk CDK v1.	Desember 4, 2021

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.