Guia do Desenvolvedor

AWS Cloud Development Kit (AWS CDK) v2

Versão 2

AWS Cloud Development Kit (AWS CDK) v2: Guia do Desenvolvedor

Table of Contents

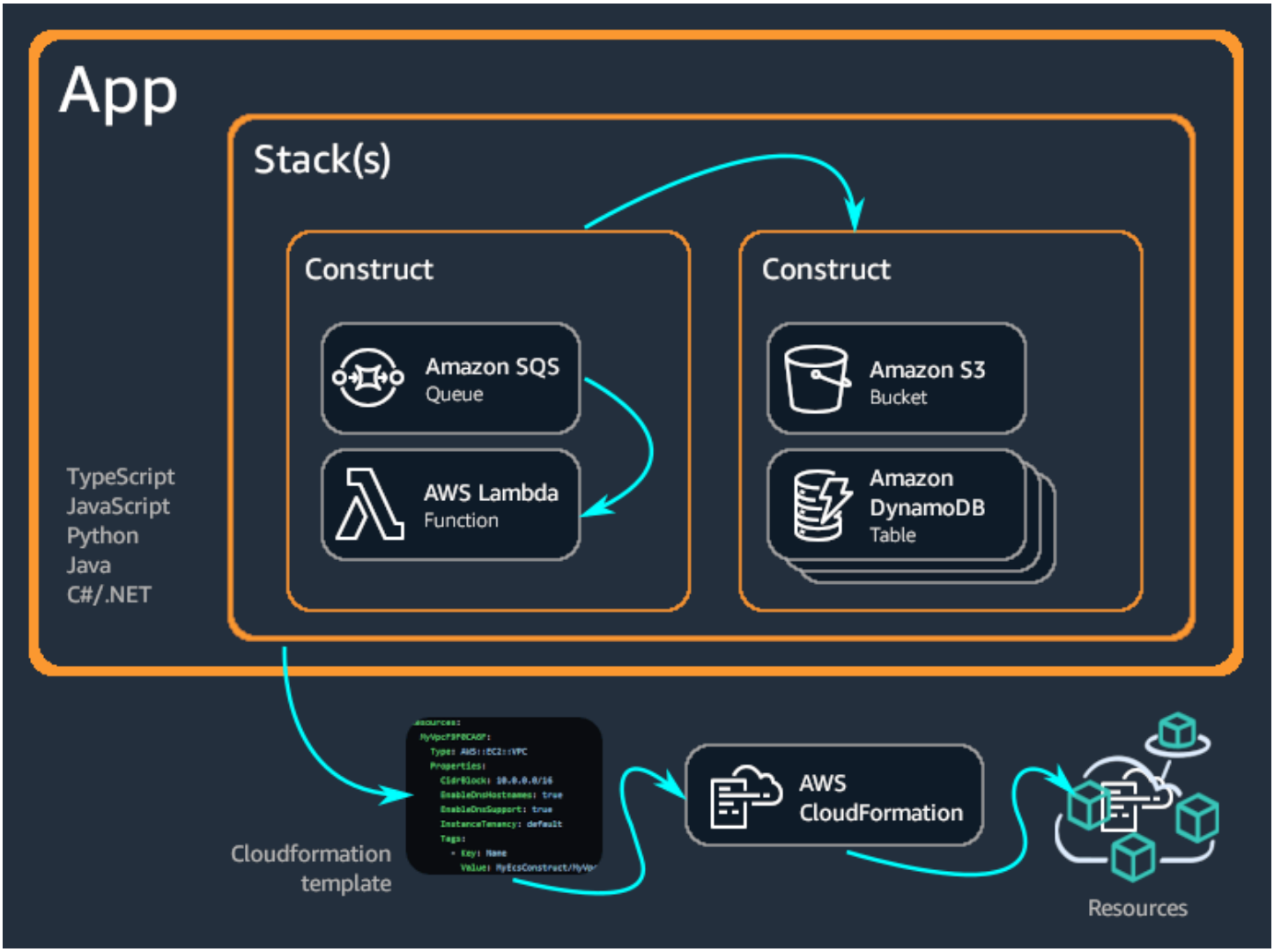
O que é o AWS CDK?

AWS Cloud Development Kit (AWS CDK) É uma estrutura de desenvolvimento de software de código aberto para definir a infraestrutura de nuvem em código e provisioná-la por meio dela. AWS CloudFormation

O AWS CDK consiste em duas partes principais:

- AWS CDK Construct Library — Uma coleção de partes de código modulares e reutilizáveis pré-escritas, chamadas de construções, que você pode usar, modificar e integrar para desenvolver sua infraestrutura rapidamente. O objetivo da AWS CDK Construct Library é reduzir a complexidade necessária para definir e integrar AWS serviços ao criar aplicativos AWS.
- AWS CDK Kit de ferramentas — Uma ferramenta de linha de comando para interagir com aplicativos CDK. Use o AWS CDK kit de ferramentas para criar, gerenciar e implantar seus AWS CDK projetos.

Os AWS CDK suportes TypeScript JavaScriptPython,Java,C#/.Net,, Go e. Você pode usar qualquer uma dessas linguagens de programação compatíveis para definir componentes de nuvem reutilizáveis conhecidos como construções. Você os compõe em pilhas e aplicativos. Em seguida, você implanta seus aplicativos CDK AWS CloudFormation para provisionar ou atualizar seus recursos.



Tópicos

- Benefícios do AWS CDK
- Exemplo do AWS CDK
- AWS CDK features
- Próximas etapas
- Saiba mais

Benefícios do AWS CDK

Use o AWS CDK para desenvolver aplicativos confiáveis, escaláveis e econômicos na nuvem com o considerável poder expressivo de uma linguagem de programação. Essa abordagem gera muitos benefícios, incluindo:

Desenvolva e gerencie sua infraestrutura como código (IaC)

Pratique a infraestrutura como código para criar, implantar e manter a infraestrutura de forma programática, descritiva e declarativa. Com o IaC, você trata a infraestrutura da mesma forma que os desenvolvedores tratam o código. Isso resulta em uma abordagem escalável e estruturada para gerenciar a infraestrutura. Para saber mais sobre o IaC, consulte Infraestrutura como código na Introdução ao DevOps AWS Whitepaper.

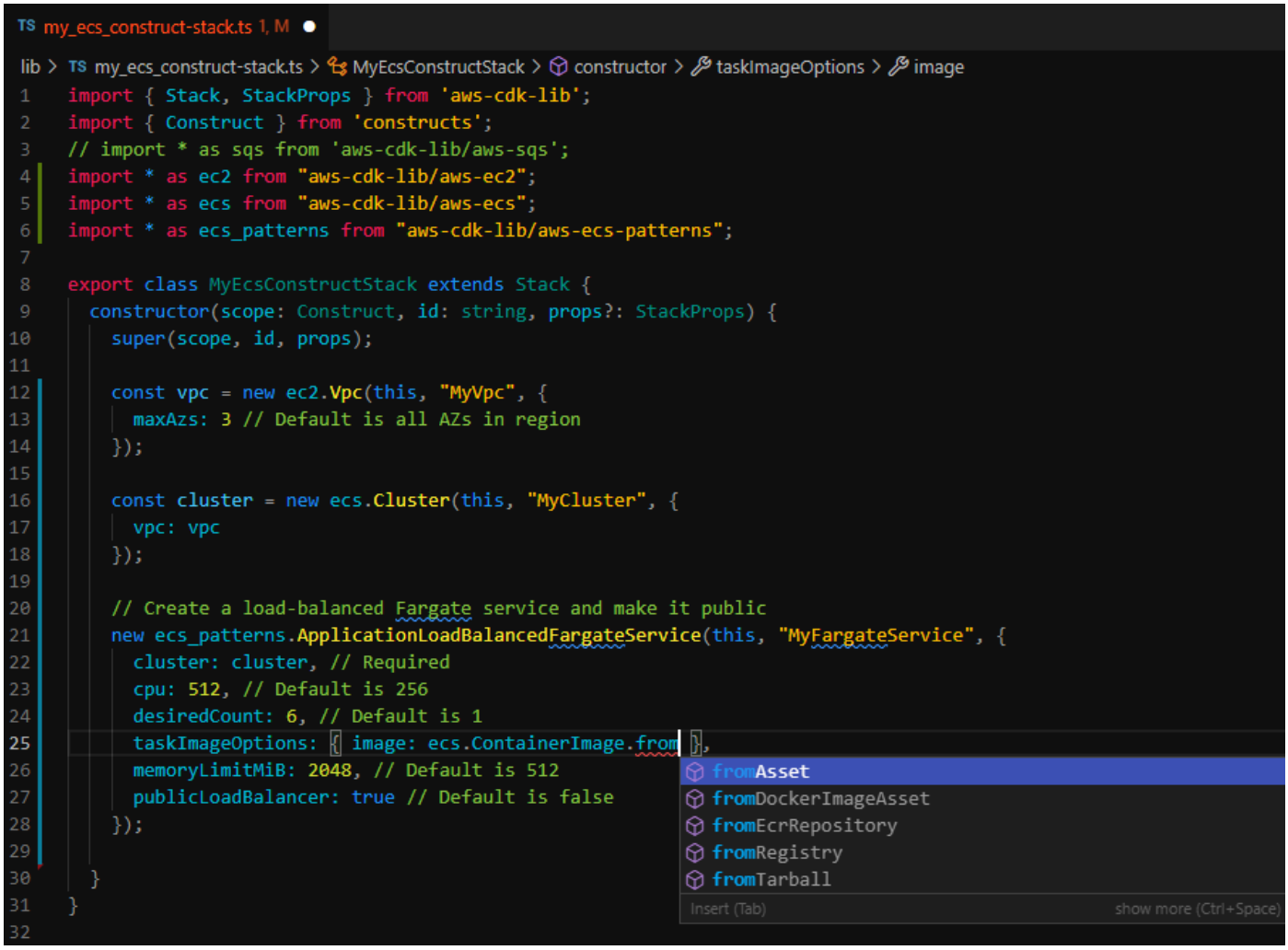
Com o AWS CDK, você pode colocar sua infraestrutura, código de aplicativo e configuração em um só lugar, garantindo que você tenha um sistema completo e implantável na nuvem em cada etapa. Empregue as melhores práticas de engenharia de software, como análises de código, testes unitários e controle de origem, para tornar sua infraestrutura mais robusta.

Defina sua infraestrutura de nuvem usando linguagens de programação de uso geral

Com o AWS CDK, você pode usar qualquer uma das seguintes linguagens de programação para definir sua infraestrutura de nuvem: TypeScript JavaScriptPython,Java,C#/.Net,, Go e. Escolha sua linguagem preferida e use elementos de programação como parâmetros, condicionais, loops, composição e herança para definir o resultado desejado de sua infraestrutura.

Use a mesma linguagem de programação para definir sua infraestrutura e sua lógica de aplicação.

Receba os benefícios de desenvolver a infraestrutura em seu IDE (Ambiente de Desenvolvimento Integrado) preferido, como destaque de sintaxe e preenchimento inteligente de código.



Implante a infraestrutura por meio de AWS CloudFormation

AWS CDK se integra AWS CloudFormation para implantar e provisionar sua infraestrutura. AWS AWS CloudFormation é um gerenciador AWS service (Serviço da AWS) que oferece amplo suporte de configurações de recursos e propriedades para serviços de provisionamento em. AWS Com AWS CloudFormation, você pode realizar implantações de infraestrutura de forma previsível e repetida, com reversão em caso de erro. Se você já está familiarizado com AWS CloudFormation, não precisa aprender um novo serviço de gerenciamento de IaC ao começar a AWS CDK usar o.

Comece a desenvolver seu aplicativo rapidamente com construções

Desenvolva mais rápido usando e compartilhando componentes reutilizáveis chamados construções. Use construções de baixo nível para definir AWS CloudFormation recursos individuais e suas propriedades. Use construções de alto nível para definir rapidamente

componentes maiores do seu aplicativo, com padrões sensatos e seguros para seus AWS recursos, definindo mais infraestrutura com menos código.

Crie suas próprias construções personalizadas para seus casos de uso exclusivos e compartilhe-as em toda a organização ou até mesmo com o público.

Exemplo do AWS CDK

Veja a seguir um exemplo do uso da AWS CDK Constructs Library para criar um serviço Amazon Elastic Container Service (Amazon ECS) com tipo de lançamento. AWS Fargate (Fargate) Para obter mais detalhes desse exemplo, consultethe 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

}
```

Essa classe produz um AWS CloudFormation modelo de mais de 500 linhas. A implantação do AWS CDK aplicativo produz mais de 50 recursos dos seguintes tipos.

- 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::VPCGatewayAttachment
- 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 features

O AWS CDKGitHub repositório

Para o AWS CDK GitHub repositório oficial, consulte aws-cdk. Aqui, você pode enviar edições, ver nossa licença, acompanhar lançamentos e muito mais.

Por ser de AWS CDK código aberto, a equipe incentiva você a contribuir para torná-la uma ferramenta ainda melhor. Para obter detalhes, consulte Contribuindo para AWS Cloud Development Kit (AWS CDK) o.

A referência AWS CDK da API

A AWS CDK Construct Library fornece APIs para definir seu aplicativo CDK e adicionar construções CDK ao aplicativo. Para obter mais informações, consulte a AWS CDK Referência da API.

O modelo de programação Construct

O Construct Programming Model (CPM) estende os conceitos por trás do AWS CDK em domínios adicionais. Outras ferramentas que usam o CPM incluem:

- CDK para Terraform (CDktf)
- CDK para Kubernetes (CDK8s)
- Projen, para criar configurações de projeto

O Construct Hub

O Construct Hub é um registro on-line onde você pode encontrar, publicar e compartilhar AWS CDK bibliotecas de código aberto.

Próximas etapas

Para começar a usar o AWS CDK, consulteComeçando com o AWS CDK.

Saiba mais

Para continuar aprendendo sobre o AWS CDK, consulte o seguinte:

- AWS CDK conceitos— Conceitos e termos importantes para AWS CDK o.
- AWS CDK Workshop — Workshop prático para aprender e usar o. AWS CDK
- AWS CDK Padrões — Coleção de código aberto de padrões de arquitetura AWS sem servidor, criada para o AWS CDK por especialistas. AWS
- AWS CDK exemplos de código — GitHub repositório de AWS CDK projetos de exemplo.
- cdk.dev — Hub orientado pela comunidade para o AWS CDK, incluindo um espaço de trabalho comunitário. Slack
- Awesome CDK — GitHub repositório contendo uma lista selecionada de projetos, guias, blogs e outros recursos de AWS CDK código aberto.
- AWS Construções de soluções — padrões aprovados de infraestrutura de configuração como código (IaC) que podem ser facilmente montados em aplicativos prontos para produção.
- AWS Blog de ferramentas para desenvolvedores — Postagens de blog filtradas para o. AWS CDK
- AWS CDK on Stack Overflow — Perguntas marcadas com aws-cdk ativado. Stack Overflow
- AWS CDK tutorial para AWS Cloud9 — Tutorial sobre como usar o AWS CDK com o ambiente de AWS Cloud9 desenvolvimento.

Para saber mais sobre tópicos relacionados ao AWS CDK, consulte o seguinte:

- AWS CloudFormation conceitos — Como o foi AWS CDK desenvolvido para funcionar AWS CloudFormation, recomendamos que você aprenda e compreenda AWS CloudFormation os principais conceitos.
- AWS Glossário — Definições dos principais termos usados em todos AWS os.

Para saber mais sobre ferramentas relacionadas ao AWS CDK que podem ser usadas para simplificar o desenvolvimento e a implantação de aplicativos sem servidor, consulte o seguinte:

- AWS Serverless Application Model— Uma ferramenta de desenvolvedor de código aberto que simplifica e melhora a experiência de criar e executar aplicativos sem servidor no. AWS
- AWSChalice— Uma estrutura para escrever aplicativos sem servidor em. Python

AWS CDK conceitos

Aprenda os principais conceitos por trás do AWS Cloud Development Kit (AWS CDK).

AWS CDK e IaC

AWS CDK É uma estrutura de código aberto que você pode usar para gerenciar sua AWS infraestrutura usando código. Essa abordagem é conhecida como infraestrutura como código (IaC). Ao gerenciar e provisionar sua infraestrutura como código, você trata sua infraestrutura da mesma forma que os desenvolvedores tratam o código. Isso oferece muitos benefícios, como controle de versão e escalabilidade. Para saber mais sobre o IaC, consulte O que é infraestrutura como código?

AWS CDK e AWS CloudFormation

O AWS CDK está totalmente integrado com AWS CloudFormation. AWS CloudFormation é um serviço totalmente gerenciado que você pode usar para gerenciar e provisionar sua infraestrutura AWS. Com AWS CloudFormation, você define sua infraestrutura em modelos e os implanta em AWS CloudFormation. O AWS CloudFormation serviço então provisiona sua infraestrutura de acordo com a configuração definida em seus modelos.

AWS CloudFormation os modelos são declarativos, o que significa que eles declaram o estado ou o resultado desejado de sua infraestrutura. Usando JSON ou YAML, você declara sua AWS infraestrutura definindo AWS recursos e propriedades. Os recursos representam os vários serviços AWS e as propriedades representam a configuração desejada desses serviços. Quando você implanta seu modelo em AWS CloudFormation, seus recursos e suas propriedades configuradas são provisionados conforme descrito em seu modelo.

Com o AWS CDK, você pode gerenciar sua infraestrutura de forma imperativa, usando linguagens de programação de uso geral. Em vez de apenas definir um estado desejado declarativamente, você pode definir a lógica ou a sequência necessária para alcançar o estado desejado. Por exemplo, você pode usar `if` instruções ou loops condicionais que determinam como alcançar o estado final desejado para sua infraestrutura.

A infraestrutura criada com o AWS CDK é eventualmente traduzida ou sintetizada em AWS CloudFormation modelos e implantada usando o serviço. AWS CloudFormation Portanto, embora AWS CDK ofereça uma abordagem diferente para criar sua infraestrutura, você ainda recebe os benefícios de AWS CloudFormation, como amplo suporte à configuração de AWS recursos e processos robustos de implantação.

Para saber mais AWS CloudFormation, consulte O que é AWS CloudFormation? no Guia do AWS CloudFormation usuário.

AWS CDK e abstrações

Com AWS CloudFormation, você deve definir todos os detalhes de como seus recursos são configurados. Isso oferece a vantagem de ter controle total sobre sua infraestrutura. No entanto, isso exige que você aprenda, compreenda e crie modelos robustos que contenham detalhes de configuração de recursos e relacionamentos entre recursos, como permissões e interações orientadas por eventos.

Com o AWS CDK, você pode ter o mesmo controle sobre suas configurações de recursos. No entanto, AWS CDK também oferece abstrações poderosas, que podem acelerar e simplificar o processo de desenvolvimento da infraestrutura. Por exemplo, AWS CDK inclui construções que fornecem configurações padrão sensatas e métodos auxiliares que geram código padronizado para você. O AWS CDK também oferece ferramentas, como a Interface de Linha de AWS CDK Comando (AWS CDK CLI), que executam ações de gerenciamento de infraestrutura para você.

Saiba mais sobre os principais AWS CDK conceitos

Interagindo com o AWS CDK

Ao usar com o AWS CDK, você interagirá principalmente com a AWS Construct Library e AWS CDK CLI o.

Desenvolvendo com o AWS CDK

Eles AWS CDK podem ser escritos em qualquer linguagem de programação compatível. Você começa com um projeto CDK, que contém uma estrutura de pastas e arquivos, incluindo ativos. Dentro do projeto, você cria um aplicativo CDK. No aplicativo, você define uma pilha, que representa diretamente uma CloudFormation pilha. Dentro da pilha, você define seus AWS recursos e propriedades usando construções.

Implantando com o AWS CDK

Você implanta aplicativos CDK em um AWS ambiente. Antes da implantação, você deve executar uma única inicialização para preparar seu ambiente.

Saiba mais

Para saber mais sobre os AWS CDK principais conceitos, consulte os tópicos desta seção.

Linguagens de programação compatíveis

O AWS Cloud Development Kit (AWS CDK) tem suporte de primeira classe para as seguintes linguagens de programação de uso geral:

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

Outros JVM .NET CLR idiomas também podem ser usados em teoria, mas não oferecemos suporte oficial no momento.

> **Note**
>
> Atualmente, este guia não inclui instruções ou exemplos de código para Go além dethe section called “Em Go”.

O AWS CDK é desenvolvido em um idioma,TypeScript. Para oferecer suporte aos outros idiomas, AWS CDK ele utiliza uma ferramenta chamada JSIIpara gerar vinculações de linguagem.

Tentamos oferecer as convenções usuais de cada linguagem para tornar o desenvolvimento o AWS CDK mais natural e intuitivo possível. Por exemplo, distribuímos módulos da AWS Construct Library usando o repositório padrão do seu idioma preferido e você os instala usando o gerenciador de pacotes padrão do idioma. Os métodos e propriedades também são nomeados usando os padrões de nomenclatura recomendados pelo seu idioma.

Veja a seguir alguns exemplos de código:

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**
>
> Esses trechos de código servem apenas para fins ilustrativos. Eles estão incompletos e não funcionarão como estão.

A AWS Construct Library é distribuída usando as ferramentas padrão de gerenciamento de pacotes de cada linguagem NPMPyPi, incluindoMaven,, NuGet e. Também fornecemos uma versão da Referência da AWS CDK API para cada idioma.

Para ajudá-lo a usar o AWS CDK em seu idioma preferido, este guia inclui os seguintes tópicos para idiomas compatíveis:

- the section called “Em TypeScript”
- the section called “Em JavaScript”
- the section called “Em Python”
- the section called “Em Java”
- the section called “Em C#”
- the section called “Em Go”

TypeScriptfoi a primeira linguagem suportada pelo AWS CDK, e grande parte do código de AWS CDK exemplo está escrito emTypeScript. Este guia inclui um tópico específico para mostrar como adaptar o TypeScript AWS CDK código para uso com as outras linguagens suportadas. Para ter mais informações, consulte AWS CDK Comparando TypeScript com outros idiomas.

AWS CDK projetos

Um AWS Cloud Development Kit (AWS CDK) projeto representa os arquivos e pastas que contêm seu código CDK. O conteúdo variará de acordo com sua linguagem de programação.

Você pode criar seu AWS CDK projeto manualmente ou com o AWS CDK comando Command Line Interface (AWS CDK CLI)`cdk init`. Neste tópico, vamos nos referir à estrutura do projeto e às convenções de nomenclatura de arquivos e pastas criados pela CLI do AWS CDK. Você pode personalizar e organizar seus projetos de CDK para atender às suas necessidades.

> **Note**
>
> A estrutura do projeto criada pelo AWS CDK CLI pode variar entre as versões ao longo do tempo.

Tópicos

Arquivos e pastas universais

.git

Se você tiver `git` instalado, o AWS CDK CLI inicializará automaticamente um Git repositório para seu projeto. O `.git` diretório contém informações sobre o repositório.

.gitignore

Arquivo de texto usado por Git para especificar arquivos e pastas a serem ignorados.

README.md

Arquivo de texto que fornece orientações básicas e informações importantes para gerenciar seu AWS CDK projeto. Modifique esse arquivo conforme necessário para documentar informações importantes sobre seu projeto CDK.

cdk.json

Arquivo de configuração para AWS CDK o. Esse arquivo fornece instruções AWS CDK CLI sobre como executar seu aplicativo.

Arquivos e pastas específicos do idioma

Os arquivos e pastas a seguir são exclusivos para cada linguagem de programação compatível.

TypeScript

Veja a seguir um exemplo de projeto criado no `my-cdk-ts-project` diretório usando o `cdk init --language typescript` comando:

```
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
```

.npm ignore

Arquivo que especifica quais arquivos e pastas devem ser ignorados ao publicar um pacote no npm registro. Esse arquivo é semelhante `.gitignore`, mas é específico para npm pacotes.

bin/.ts my-cdk-ts-project

O arquivo do aplicativo define seu aplicativo CDK. Os projetos CDK podem conter um ou mais arquivos de aplicativos. Os arquivos do aplicativo são armazenados na `bin` pasta.

Veja a seguir um exemplo de um arquivo de aplicativo básico que define um aplicativo 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

Arquivo de configuração paraJest. Jesté uma estrutura JavaScript de teste popular.

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

O arquivo de pilha define sua pilha de CDK. Na sua pilha, você define AWS recursos e propriedades usando construções.

Veja a seguir um exemplo de um arquivo de pilha básico que define uma pilha 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
 }
}
```

módulos_nódulos

Pasta comum em Node.js projetos que contêm dependências para seu projeto.

package-lock.json

Arquivo de metadados que funciona com o `package.json` arquivo para gerenciar versões de dependências.

pacote.json

Arquivo de metadados que é comumente usado em Node.js projetos. Esse arquivo contém informações sobre seu projeto CDK, como nome do projeto, definições de script, dependências e outras informações de importação em nível de projeto.

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

Uma pasta de teste é criada para organizar os testes do seu projeto CDK. Um arquivo de teste de amostra também é criado.

Você pode escrever testes TypeScript e usá-los Jest para compilar seu TypeScript código antes de executá-los.

tsconfig.json

Arquivo de configuração usado em TypeScript projetos que especifica as opções do compilador e as configurações do projeto.

JavaScript

Veja a seguir um exemplo de projeto criado no `my-cdk-js-project` diretório usando o `cdk init --language javascript` comando:

```
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
```

.npm ignore

Arquivo que especifica quais arquivos e pastas devem ser ignorados ao publicar um pacote no npm registro. Esse arquivo é semelhante`.gitignore`, mas é específico para npm pacotes.

bin/.js my-cdk-js-project

O arquivo do aplicativo define seu aplicativo CDK. Os projetos CDK podem conter um ou mais arquivos de aplicativos. Os arquivos do aplicativo são armazenados na `bin` pasta.

Veja a seguir um exemplo de um arquivo de aplicativo básico que define um aplicativo 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

Arquivo de configuração paraJest. Jesté uma estrutura JavaScript de teste popular.

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

O arquivo de pilha define sua pilha de CDK. Na sua pilha, você define AWS recursos e propriedades usando construções.

Veja a seguir um exemplo de um arquivo de pilha básico que define uma pilha 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 }
```

módulos_nódulos

Pasta comum em Node.js projetos que contêm dependências para seu projeto.

package-lock.json

Arquivo de metadados que funciona com o `package.json` arquivo para gerenciar versões de dependências.

pacote.json

Arquivo de metadados que é comumente usado em Node.js projetos. Esse arquivo contém informações sobre seu projeto CDK, como nome do projeto, definições de script, dependências e outras informações de importação em nível de projeto.

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

Uma pasta de teste é criada para organizar os testes do seu projeto CDK. Um arquivo de teste de amostra também é criado.

Você pode escrever testes JavaScript e usá-los Jest para compilar seu JavaScript código antes de executá-los.

Python

Veja a seguir um exemplo de projeto criado no `my-cdk-py-project` diretório usando o `cdk init --language python` comando:

```
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

O CDK cria CLI automaticamente um ambiente virtual para seu projeto. O `.venv` diretório se refere a esse ambiente virtual.

app.py

O arquivo do aplicativo define seu aplicativo CDK. Os projetos CDK podem conter um ou mais arquivos de aplicativos.

Veja a seguir um exemplo de um arquivo de aplicativo básico que define um aplicativo 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

Diretório que contém seus arquivos de pilha. O CDK CLI cria o seguinte aqui:

- `__init__.py` — Um arquivo de definição de Python pacote vazio.
- `my_cdk_py_project`— Arquivo que define sua pilha de CDK. Em seguida, você define AWS recursos e propriedades dentro da pilha usando construções.

Veja a seguir um exemplo de um arquivo de pilha:

```
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

Arquivo semelhante`requirements.txt`, mas usado para gerenciar dependências especificamente para fins de desenvolvimento em vez de produção.

requirements.txt

Arquivo comum usado em Python projetos para especificar e gerenciar dependências do projeto.

source.bat

O arquivo Batch para Windows isso é usado para configurar o ambiente Python virtual.

testes

Diretório que contém testes para seu projeto CDK.

Veja a seguir um exemplo de um teste unitário:

```
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

Veja a seguir um exemplo de projeto criado no `my-cdk-java-project` diretório usando o `cdk init --language java` comando:

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

pom.xml

Arquivo que contém informações de configuração e metadados sobre seu projeto CDK. Este arquivo faz parte doMaven.

src/main

Diretório contendo seu aplicativo e arquivos de pilha.

Veja a seguir um exemplo de arquivo de aplicativo:

```
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();
 }
}
```

Veja a seguir um exemplo de arquivo de pilha:

```
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/test

Diretório contendo seus arquivos de teste. Veja um exemplo a seguir:

```
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#

Veja a seguir um exemplo de projeto criado no `my-cdk-csharp-project` diretório usando o `cdk init --language csharp` comando:

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

src/ MyCdkCsharpProject

Diretório contendo seu aplicativo e arquivos de pilha.

Veja a seguir um exemplo de arquivo de aplicativo:

```
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();
  }
 }
}
```

Veja a seguir um exemplo de arquivo de pilha:

```
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
  }
 }
}
```

Esse diretório também contém o seguinte:

- `GlobalSuppressions.cs`— Arquivo usado para suprimir avisos ou erros específicos do compilador em seu projeto.

- `.csproj`— Arquivo baseado em XML usado para definir configurações de projeto, dependências e configurações de compilação.

src/.sln MyCdkCsharpProject

Microsoft Visual Studio Solution Fileusado para organizar e gerenciar projetos relacionados.

Go

Veja a seguir um exemplo de projeto criado no `my-cdk-go-project` diretório usando o `cdk init --language go` comando:

```
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

Arquivo que contém informações do módulo e é usado para gerenciar dependências e controle de versão do seu projeto. Go

my-cdk-go-project.go

Arquivo que define seu aplicativo e suas pilhas de CDK.

Veja um exemplo a seguir:

```
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_teste.go

Arquivo que define um teste de amostra.

Veja um exemplo a seguir:

```
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 aplicativos

O AWS Cloud Development Kit (AWS CDK) aplicativo ou aplicativo é uma coleção de uma ou mais pilhas de CDK. As pilhas são uma coleção de uma ou mais construções, que definem AWS recursos e propriedades. Portanto, o agrupamento geral de suas pilhas e construções é conhecido como seu aplicativo CDK.

Tópicos

- Definindo aplicativos
- A árvore de construção
- O ciclo de vida do aplicativo

Definindo aplicativos

Você cria um aplicativo definindo uma instância do aplicativo no arquivo do seu projeto. Para fazer isso, você importa e usa a Appconstrução da AWS Construct Library. A App construção não requer nenhum argumento de inicialização. É a única construção que pode ser usada como raiz.

As `Stack` classes `App` e da AWS Construct Library são construções exclusivas. Em comparação com outras construções, elas não configuram AWS recursos sozinhas. Em vez disso, eles são usados para fornecer contexto para suas outras construções. Todas as construções que

representam AWS recursos devem ser definidas, direta ou indiretamente, dentro do escopo de uma `Stack` construção. `Stack`construções são definidas dentro do escopo de uma `App` construção.

Os aplicativos são então sintetizados para criar AWS CloudFormation modelos para suas pilhas. Veja um exemplo a seguir:

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)
```

As pilhas em um único aplicativo podem se referir facilmente aos recursos e propriedades umas das outras. Ele AWS CDK infere dependências entre as pilhas para que elas possam ser implantadas na ordem correta. Você pode implantar qualquer uma ou todas as pilhas em um aplicativo com um único `cdk deploy` comando.

A árvore de construção

As construções são definidas dentro de outras construções usando o `scope` argumento que é passado para cada construção, com a App classe como raiz. Dessa forma, um AWS CDK aplicativo define uma hierarquia de construções conhecida como árvore de construção.

A raiz dessa árvore é seu aplicativo, que é uma instância da App classe. Dentro do aplicativo, você instancia uma ou mais pilhas. Nas pilhas, você instancia construções, que podem elas mesmas instanciar recursos ou outras construções, e assim por diante na árvore.

As construções são sempre definidas explicitamente dentro do escopo de outra construção, o que cria relacionamentos entre as construções. Quase sempre, você deve passar `this` (em Python,`self`) como escopo, indicando que a nova construção é filha da construção atual. O padrão pretendido é que você derive sua construção e, em seguida `Construct`, instancie as construções que ela usa em seu construtor.

Passar o escopo explicitamente permite que cada construção se adicione à árvore, com esse comportamento inteiramente contido na classe `Construct` base. Funciona da mesma forma em todos os idiomas suportados pelo AWS CDK e não requer personalização adicional.

> **Important**
>
> Tecnicamente, é possível passar algum escopo além `this` da instanciação de uma construção. Você pode adicionar construções em qualquer lugar da árvore ou até mesmo em outra pilha no mesmo aplicativo. Por exemplo, você pode escrever uma função no estilo mixin que adiciona construções a um escopo passado como argumento. A dificuldade prática aqui é que você não pode garantir facilmente que os IDs escolhidos para suas construções

sejam exclusivos dentro do escopo de outra pessoa. A prática também torna seu código mais difícil de entender, manter e reutilizar. Quase sempre é melhor encontrar uma maneira de expressar sua intenção sem recorrer ao abuso da discussão. `scope`

O AWS CDK usa os IDs de todas as construções no caminho da raiz da árvore até cada construção secundária para gerar os IDs exclusivos exigidos pelo AWS CloudFormation. Essa abordagem significa que os IDs de construção só precisam ser exclusivos em seu escopo, e não em toda a pilha, como em nativo AWS CloudFormation. No entanto, se você mover uma construção para um escopo diferente, a ID exclusiva da pilha gerada será alterada e AWS CloudFormation não a considerará o mesmo recurso.

A árvore de construção é separada das construções que você define no seu AWS CDK código. No entanto, é acessível por meio de qualquer `node` atributo de construção, que é uma referência ao nó que representa essa construção na árvore. Cada nó é uma Nodeinstância, cujos atributos fornecem acesso à raiz da árvore e aos escopos e filhos principais do nó.

1. `node.children`— Os filhos diretos da construção.
2. `node.id`— O identificador da construção dentro de seu escopo.
3. `node.path`— O caminho completo da construção, incluindo os IDs de todos os seus pais.
4. `node.root`— A raiz da árvore de construção (o aplicativo).
5. `node.scope`— O escopo (pai) da construção ou indefinido se o nó for a raiz.
6. `node.scopes`— Todos os pais da construção, até a raiz.
7. `node.uniqueId`— O identificador alfanumérico exclusivo para essa construção na árvore (por padrão, gerado a partir de `node.path` e um hash).

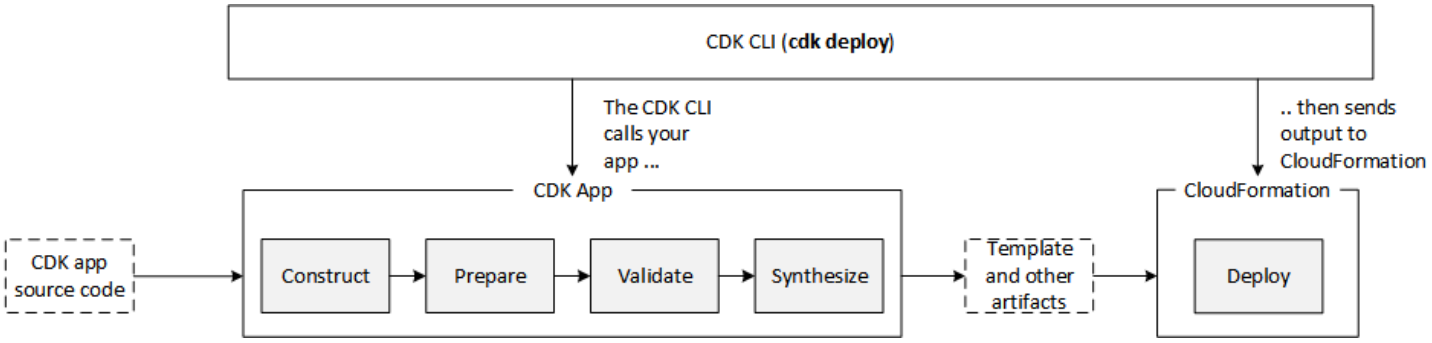
A árvore de construção define uma ordem implícita na qual as construções são sintetizadas em recursos no modelo final. AWS CloudFormation Onde um recurso deve ser criado antes do outro, AWS CloudFormation ou a AWS Construct Library geralmente infere a dependência. Em seguida, eles garantem que os recursos sejam criados na ordem correta.

Você também pode adicionar uma dependência explícita entre dois nós usando. `node.addDependency()` Para obter mais informações, consulte Dependências na Referência da AWS CDK API.

Isso AWS CDK fornece uma maneira simples de visitar cada nó na árvore de construção e realizar uma operação em cada um. Para ter mais informações, consulte the section called “Aspectos”.

O ciclo de vida do aplicativo

Quando você implanta seu aplicativo CDK, as seguintes fases ocorrem. Isso é conhecido como ciclo de vida do aplicativo:



Um AWS CDK aplicativo passa pelas seguintes fases em seu ciclo de vida.

- Construção (ou inicialização) — Seu código instancia todas as construções definidas e, em seguida, as vincula. Nesse estágio, todas as construções (aplicativo, pilhas e suas construções secundárias) são instanciadas e a cadeia de construtores é executada. A maior parte do código do seu aplicativo é executada nesse estágio.
- Preparação — Todas as construções que implementaram o `prepare` método participam de uma rodada final de modificações, para configurar seu estado final. A fase de preparação acontece automaticamente. Como usuário, você não vê nenhum feedback dessa fase. É raro precisar usar o gancho de “preparação” e geralmente não é recomendado. Tenha muito cuidado ao alterar a árvore de construção durante essa fase, pois a ordem das operações pode afetar o comportamento.
- Validação — Todas as construções que implementaram o `validate` método podem se validar para garantir que estejam em um estado de implantação correta. Você será notificado sobre quaisquer falhas de validação que ocorram durante essa fase. Geralmente, recomendamos realizar a validação o mais rápido possível (geralmente assim que você receber alguma informação) e lançar exceções o mais rápido possível. A validação antecipada melhora a confiabilidade, pois os rastreamentos de pilha serão mais precisos e garantirão que seu código possa continuar sendo executado com segurança.
- Síntese — Esse é o estágio final da execução do seu AWS CDK aplicativo. É acionado por uma chamada para`app.synth()`, percorre a árvore de construção e invoca o `synthesize` método em todas as construções. As construções implementadas `synthesize` podem participar da síntese e emitir artefatos de implantação para o conjunto de nuvem resultante. Esses artefatos incluem AWS CloudFormation modelos, pacotes de AWS Lambda aplicativos, ativos de

arquivos e Docker imagens e outros artefatos de implantação. the section called “Montagens em nuvem”descreve a saída dessa fase. Na maioria dos casos, você não precisará implementar o `synthesize` método.

- Implantação — Nessa fase, ele AWS CDK CLI pega o conjunto de artefatos de implantação na nuvem produzido pela fase de síntese e o implanta em um AWS ambiente. Ele carrega ativos para o Amazon S3 e o Amazon ECR, ou onde quer que eles precisem ir. Em seguida, ele inicia uma AWS CloudFormation implantação para implantar o aplicativo e criar os recursos.

Quando a fase de AWS CloudFormation implantação começa, seu AWS CDK aplicativo já foi concluído e encerrado. Isso ocasiona o seguinte:

- O AWS CDK aplicativo não pode responder a eventos que acontecem durante a implantação, como a criação de um recurso ou o término de toda a implantação. Para executar o código durante a fase de implantação, você deve injetá-lo no AWS CloudFormation modelo como um recurso personalizado. Para obter mais informações sobre como adicionar um recurso personalizado ao seu aplicativo, consulte o AWS CloudFormation módulo ou o exemplo de recurso personalizado.
- Talvez o AWS CDK aplicativo precise funcionar com valores que não podem ser conhecidos no momento em que é executado. Por exemplo, se o AWS CDK aplicativo define um bucket do Amazon S3 com um nome gerado automaticamente e você recupera o atributo (`bucket.bucketName`Python:`bucket_name`), esse valor não é o nome do bucket implantado. Em vez disso, você obtém um `Token` valor. Para determinar se um valor específico está disponível, chame `cdk.isUnresolved(value)` (Python:`is_unresolved`). Para mais detalhes, consulte the section called “Tokens”.

Montagens em nuvem

A chamada para `app.synth()` é o que diz AWS CDK para sintetizar uma montagem em nuvem a partir de um aplicativo. Normalmente, você não interage diretamente com os conjuntos de nuvem. São arquivos que incluem tudo o que é necessário para implantar seu aplicativo em um ambiente de nuvem. Por exemplo, ele inclui um AWS CloudFormation modelo para cada pilha em seu aplicativo. Ele também inclui uma cópia de todos os ativos de arquivo ou imagens do Docker que você referenciar no seu aplicativo.

Consulte a especificação de montagem em nuvem para obter detalhes sobre como as montagens em nuvem são formatadas.

Para interagir com o conjunto de nuvem que seu AWS CDK aplicativo cria, você normalmente usa AWS CDK CLI o. No entanto, qualquer ferramenta que possa ler o formato de montagem em nuvem pode ser usada para implantar seu aplicativo.

Executando seu aplicativo

O CDK CLI precisa saber como executar seu AWS CDK aplicativo. Se você criou o projeto a partir de um modelo usando o `cdk init` comando, o `cdk.json` arquivo do seu aplicativo inclui uma `app` chave. Essa chave especifica o comando necessário para o idioma em que o aplicativo está escrito. Se sua linguagem exigir compilação, a linha de comando executa essa etapa antes de executar o aplicativo, para que você não se esqueça de fazer isso.

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"
}
```

Se você não criou seu projeto usando o CDKCLI, ou se quiser substituir a linha de comando fornecida`cdk.json`, você pode usar a --app opção ao emitir o comando. `cdk`

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

A parte *`executável`* do comando indica o comando que deve ser executado para executar seu aplicativo CDK. Use aspas conforme mostrado, pois esses comandos contêm espaços. O *`comando cdk`* é um subcomando como synth ou deploy que informa ao CDK o CLI que você quer fazer com seu aplicativo. Siga isso com todas as opções adicionais necessárias para esse subcomando.

Eles também AWS CDK CLI podem interagir diretamente com um conjunto de nuvem já sintetizado. Para fazer isso, passe o diretório no qual o conjunto de nuvem está armazenado--app. O exemplo a seguir lista as pilhas definidas no conjunto de nuvem armazenado em`./my-cloud-assembly`.

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

Pilhas

Uma AWS Cloud Development Kit (AWS CDK) pilha é uma coleção de uma ou mais construções, que definem AWS recursos. Cada pilha CDK representa uma AWS CloudFormation pilha em seu aplicativo CDK. Na implantação, as construções dentro de uma pilha são provisionadas como uma única unidade, chamada de pilha. AWS CloudFormation Para saber mais sobre AWS CloudFormation pilhas, consulte Como trabalhar com pilhas no Guia do AWS CloudFormation usuário.

Como as pilhas de CDK são implementadas por meio de AWS CloudFormation pilhas, AWS CloudFormation cotas e limitações se aplicam. Para saber mais, consulte AWS CloudFormation cotas.

Tópicos

Definindo pilhas

As pilhas são definidas dentro do contexto de um aplicativo. Você define uma pilha usando a `Stack` classe da AWS Construct Library. As pilhas podem ser definidas de qualquer uma das seguintes formas:

- Diretamente dentro do escopo do aplicativo.
- Indiretamente por qualquer construção dentro da árvore.

O exemplo a seguir define um aplicativo CDK que contém duas pilhas:

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();
```

O exemplo a seguir é um padrão comum para definir uma pilha em um arquivo separado. Aqui, estendemos ou herdamos a `Stack` classe e definimos um construtor que aceita `scopeid`, e. `props` Em seguida, invocamos o construtor `Stack` da classe base usando `super` com o recebido `scopeid`, e. `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
}
```

O exemplo a seguir declara uma classe de pilha chamada `MyFirstStack` que inclui um único bucket do 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
```

```
}
```

No entanto, esse código declarou apenas uma pilha. Para que a pilha seja realmente sintetizada em um AWS CloudFormation modelo e implantada, ela deve ser instanciada. E, como todas as construções de CDK, ela deve ser instanciada em algum contexto. Esse `App` é esse contexto.

Se você estiver usando o modelo de AWS CDK desenvolvimento padrão, suas pilhas serão instanciadas no mesmo arquivo em que você instancia o objeto. `App`

TypeScript

O arquivo com o nome do seu projeto (por exemplo,`hello-cdk.ts`) na `bin` pasta do seu projeto.

JavaScript

O arquivo com o nome do seu projeto (por exemplo,`hello-cdk.js`) na `bin` pasta do seu projeto.

Python

O arquivo `app.py` no diretório principal do seu projeto.

Java

O arquivo chamado*`ProjectName`*`App.java`, por exemplo`HelloCdkApp.java`, está aninhado nas profundezas do `src/main` diretório.

C#

O arquivo nomeado `Program.cs` abaixo`src\`*`ProjectName`*, por exemplo`src\HelloCdk\Program.cs`.

A API de pilha

O objeto Stack fornece uma API avançada, incluindo o seguinte:

- `Stack.of(construct)`— Um método estático que retorna a pilha na qual uma construção é definida. Isso é útil se você precisar interagir com uma pilha de dentro de uma construção reutilizável. A chamada falhará se uma pilha não puder ser encontrada no escopo.

- `stack.stackName`(Python:`stack_name`) — Retorna o nome físico da pilha. Conforme mencionado anteriormente, todas as AWS CDK pilhas têm um nome físico que AWS CDK pode ser resolvido durante a síntese.
- `stack.region`e `stack.account` — Retorne a AWS região e a conta, respectivamente, nas quais essa pilha será implantada. Essas propriedades retornam uma das seguintes:
  - A conta ou região especificada explicitamente quando a pilha foi definida
  - Um token codificado em string que segue os AWS CloudFormation pseudoparâmetros da conta e da região para indicar que essa pilha é independente do ambiente

  Para obter informações sobre como os ambientes são determinados para pilhas, consultethe section called “Ambientes do”.
- `stack.addDependency(stack)`(Python: `stack.add_dependency(stack)` — Pode ser usado para definir explicitamente a ordem de dependência entre duas pilhas. Essa ordem é respeitada pelo cdk deploy comando ao implantar várias pilhas ao mesmo tempo.
- `stack.tags`— Retorna um TagManagerque você pode usar para adicionar ou remover tags no nível da pilha. Esse gerenciador de tags marca todos os recursos da pilha e também marca a própria pilha quando ela é criada por meio dela. AWS CloudFormation
- `stack.partition`, `stack.urlSuffix` (Python:`url_suffix`), (`stack.stackId`Python:) e (`stack.notificationArn`Python: `stack_idnotification_arn`) — Retorna tokens que se resolvem nos respectivos AWS CloudFormation pseudoparâmetros, como. `{ "Ref": "AWS::Partition" }` Esses tokens são associados ao objeto de pilha específico para que a AWS CDK estrutura possa identificar referências entre pilhas.
- `stack.availabilityZones`(Python:`availability_zones`) — Retorna o conjunto de zonas de disponibilidade disponíveis no ambiente em que essa pilha é implantada. Para pilhas independentes do ambiente, isso sempre retorna uma matriz com duas zonas de disponibilidade. Para pilhas específicas do ambiente, ele AWS CDK consulta o ambiente e retorna o conjunto exato de zonas de disponibilidade disponíveis na região que você especificou.
- `stack.parseArn(arn)`e `stack.formatArn(comps)` (Python:`parse_arn`,`format_arn`) — Pode ser usado para trabalhar com Amazon Resource Names (ARNs).
- `stack.toJsonString(obj)`(Python:`to_json_string`) — Pode ser usado para formatar um objeto arbitrário como uma string JSON que pode ser incorporada em um modelo. AWS CloudFormation O objeto pode incluir tokens, atributos e referências, que só são resolvidos durante a implantação.
- `stack.templateOptions`(Python:`template_options`) — Use para especificar opções de AWS CloudFormation modelo, como Transformação, Descrição e Metadados, para sua pilha.

Trabalhar com pilhas do

Para listar todas as pilhas em um aplicativo CDK, use o cdk ls comando. O exemplo anterior produziria o seguinte:

```
stack1
stack2
```

As pilhas são implantadas como parte de uma AWS CloudFormation pilha em um ambiente. AWS O ambiente abrange uma área específica Conta da AWS Região da AWS e.

Quando você executa o cdk synth comando para um aplicativo com várias pilhas, o conjunto de nuvem inclui um modelo separado para cada instância da pilha. Mesmo que as duas pilhas sejam instâncias da mesma classe, elas são AWS CDK emitidas como dois modelos individuais.

Você pode sintetizar cada modelo especificando o nome da pilha no comando. cdk synth O exemplo a seguir sintetiza o modelo para stack1.

```
$ cdk synth stack1
```

Essa abordagem é conceitualmente diferente de como os AWS CloudFormation modelos são normalmente usados, em que um modelo pode ser implantado várias vezes e parametrizado por meio de parâmetros.AWS CloudFormation Embora AWS CloudFormation os parâmetros possam ser definidos no AWS CDK, eles geralmente são desencorajados porque AWS CloudFormation os parâmetros são resolvidos somente durante a implantação. Isso significa que você não pode determinar o valor deles em seu código.

Por exemplo, para incluir condicionalmente um recurso em seu aplicativo com base em um valor de parâmetro, você deve configurar uma AWS CloudFormation condição e marcar o recurso com ela. AWS CDK Adota uma abordagem em que modelos concretos são resolvidos no momento da síntese. Portanto, você pode usar uma instrução if para verificar o valor e determinar se um recurso deve ser definido ou se algum comportamento deve ser aplicado.

> **Note**
>
> Isso AWS CDK fornece o máximo de resolução possível durante o tempo de síntese para permitir o uso idiomático e natural de sua linguagem de programação.

Como qualquer outra construção, as pilhas podem ser compostas em grupos. O código a seguir mostra um exemplo de um serviço que consiste em três pilhas: um plano de controle, um plano de dados e pilhas de monitoramento. A construção do serviço é definida duas vezes: uma para o ambiente beta e outra para o ambiente de produção.

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();
    }
}
```

Eventualmente, esse AWS CDK aplicativo consiste em seis pilhas, três para cada ambiente:

```
$ cdk ls

betacpDA8372D3
betadataE23DB2BA
betamon632BD457
prodcp187264CE
proddataF7378CE5
prodmon631A1083
```

Os nomes físicos das AWS CloudFormation pilhas são determinados automaticamente AWS CDK com base no caminho de construção da pilha na árvore. Por padrão, o nome de uma pilha é derivado do ID de construção do `Stack` objeto. No entanto, você pode especificar um nome explícito usando a `stackName` prop (em Python,`stack_name`), da seguinte maneira.

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"
});
```

Pilhas aninhadas

A NestedStackconstrução oferece uma maneira de contornar o limite de AWS CloudFormation 500 recursos para pilhas. Uma pilha aninhada conta como apenas um recurso na pilha que a contém. No entanto, ele pode conter até 500 recursos, incluindo pilhas aninhadas adicionais.

O escopo de uma pilha aninhada deve ser uma construção `Stack` ou`NestedStack`. A pilha aninhada não precisa ser declarada lexicamente dentro de sua pilha principal. É necessário apenas passar a pilha principal como o primeiro parâmetro (`scope`) ao instanciar a pilha aninhada. Além dessa restrição, definir construções em uma pilha aninhada funciona exatamente da mesma forma que em uma pilha comum.

No momento da síntese, a pilha aninhada é sintetizada em seu próprio AWS CloudFormation modelo, que é carregado no bucket de teste na implantação AWS CDK . As pilhas aninhadas são vinculadas à pilha principal e não são tratadas como artefatos de implantação independentes. Eles não estão listados por `cdk list` e não podem ser implantados por`cdk deploy`.

As referências entre pilhas principais e pilhas aninhadas são traduzidas automaticamente em parâmetros de pilha e saídas nos AWS CloudFormation modelos gerados, como acontece com qualquer referência de pilha cruzada.

> **Warning**
>
> As alterações na postura de segurança não são exibidas antes da implantação das pilhas aninhadas. Essas informações são exibidas somente para pilhas de nível superior.

Construções

As construções são os blocos de construção básicos dos AWS Cloud Development Kit (AWS CDK) aplicativos. Uma construção é um componente em seu aplicativo que representa um ou mais AWS CloudFormation recursos e sua configuração. Você constrói seu aplicativo, peça por peça, importando e configurando construções.

Construções são classes que você importa para seus aplicativos CDK. As construções estão disponíveis na AWS Construct Library. Você também pode criar e distribuir suas próprias construções ou usar construções criadas por desenvolvedores terceirizados.

As construções fazem parte do Construct Programming Model (CPM). Eles estão disponíveis para uso com outras ferramentas, como CDK para Terraform (CDKtf), CDK para Kubernetes (CDK8s) e. Projen

Tópicos

AWS Construir biblioteca

A AWS Construct Library contém uma coleção de construções que são desenvolvidas e mantidas por AWS. Ele é organizado em vários módulos que contêm construções que representam todos os

recursos disponíveis AWS. Para obter informações de referência, consulte a Referência AWS CDK da API.

O pacote principal do CDK é chamado `aws-cdk-lib` e contém a maior parte da AWS Construct Library. Ele também contém classes básicas, como `Stack` `App` e.

O nome real do pacote CDK principal varia de acordo com o idioma.

TypeScript

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

JavaScript

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

Python

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

Java

Em`pom.xml`, adicione	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Import	`import software.amazon.awscdk.App;`

C#

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

Go

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

> **Note**
>
> Se você criou um projeto CDK usandocdk init, não precisa instalar `aws-cdk-lib` manualmente.

A AWS Construct Library também contém o `constructs` pacote com a classe `Construct` base. Ele está em seu próprio pacote porque é usado por outras ferramentas baseadas em construção, além do AWS CDK, incluindo o CDK para Terraform e o CDK para Kubernetes.

Vários terceiros também publicaram construções compatíveis com o. AWS CDK Visite o Construct Hub para explorar o ecossistema do AWS CDK Construct Partner.

Construa níveis

As construções da AWS Construct Library são categorizadas em três níveis. Cada nível oferece um nível crescente de abstração. Quanto maior a abstração, mais fácil de configurar, exigindo menos experiência. Quanto menor a abstração, mais personalização está disponível, exigindo mais experiência.

Construções de nível 1 (L1)

As construções L1, também conhecidas como recursos CFN, são as construções de nível mais baixo e não oferecem abstração. Cada construção L1 é mapeada diretamente para um único AWS CloudFormation recurso. Com construções L1, você importa uma construção que representa um recurso específico AWS CloudFormation . Em seguida, você define as propriedades do recurso em sua instância de construção.

As construções L1 são ótimas para usar quando você está familiarizado AWS CloudFormation e precisa de controle total sobre a definição das propriedades do AWS recurso.

Na AWS Construct Library, as construções L1 são nomeadas começando com`Cfn`, seguidas por um identificador para o AWS CloudFormation recurso que elas representam. Por exemplo, a `CfnBucket` construção é uma construção L1 que representa um `AWS::S3::Bucket` AWS CloudFormation recurso.

As construções L1 são geradas a partir da especificação do AWS CloudFormation recurso. Se um recurso existir em AWS CloudFormation, ele estará disponível no AWS CDK como uma construção L1. Novos recursos ou propriedades podem levar até uma semana para serem disponibilizados na AWS Construct Library. Para obter mais informações, consulte a referência de tipos de AWS recursos e propriedades no Guia AWS CloudFormation do usuário.

Construções de nível 2 (L2)

As construções L2, também conhecidas como construções curadas, são cuidadosamente desenvolvidas pela equipe do CDK e geralmente são o tipo de construção mais amplamente usado. As construções L2 são mapeadas diretamente para AWS CloudFormation recursos individuais, semelhantes às construções L1. Em comparação com as construções L1, as construções L2 fornecem uma abstração de alto nível por meio de uma API intuitiva baseada em intenção. As construções L2 incluem configurações de propriedades padrão sensatas, políticas de segurança de melhores práticas e geram muito código padronizado e lógica de cola para você.

As construções L2 também fornecem métodos auxiliares para a maioria dos recursos que tornam mais simples e rápido definir propriedades, permissões, interações baseadas em eventos entre recursos e muito mais.

A `s3.Bucket` classe é um exemplo de uma construção L2 para um recurso de bucket do Amazon Simple Storage Service (Amazon S3).

A AWS Construct Library contém construções L2 que são designadas como estáveis e prontas para uso em produção. Para construções L2 em desenvolvimento, elas são designadas como experimentais e oferecidas em um módulo separado.

Construções de nível 3 (L3)

As construções L3, também conhecidas como padrões, são o nível mais alto de abstração. Cada construção L3 pode conter uma coleção de recursos que são configurados para trabalhar juntos para realizar uma tarefa ou serviço específico em seu aplicativo. As construções L3 são usadas para criar AWS arquiteturas inteiras para casos de uso específicos em seu aplicativo.

Para fornecer projetos de sistema completos ou partes substanciais de um sistema maior, as construções L3 oferecem configurações de propriedades padrão opinativas. Eles são construídos em torno de uma abordagem específica para resolver um problema e fornecer uma solução. Com construções L3, você pode criar e configurar vários recursos rapidamente, com a menor quantidade de entrada e código.

A `ecsPatterns.ApplicationLoadBalancedFargateService` classe é um exemplo de uma construção L3 que representa um AWS Fargate serviço executado em um cluster do Amazon Elastic Container Service (Amazon ECS) e liderado por um balanceador de carga de aplicativos.

Semelhantes às construções L2, as construções L3 que estão prontas para uso em produção estão incluídas na Construct Library. AWS Os que estão em desenvolvimento são oferecidos em módulos separados.

Definindo construções

Composição

A composição é o padrão chave para definir abstrações de alto nível por meio de construções. Uma construção de alto nível pode ser composta a partir de qualquer número de construções de nível inferior. De uma perspectiva de baixo para cima, você usa construções para organizar os AWS recursos individuais que deseja implantar. Você usa quaisquer abstrações que sejam convenientes para seu propósito, com quantos níveis você precisar.

Com a composição, você define componentes reutilizáveis e os compartilha como qualquer outro código. Por exemplo, uma equipe pode definir uma construção que implemente as melhores práticas da empresa para uma tabela do Amazon DynamoDB, incluindo backup, replicação global,

escalabilidade automática e monitoramento. A equipe pode compartilhar a construção internamente com outras equipes ou publicamente.

As equipes podem usar construções como qualquer outro pacote de biblioteca. Quando a biblioteca é atualizada, os desenvolvedores têm acesso às melhorias e correções de erros da nova versão, semelhante a qualquer outra biblioteca de código.

Inicialização

As estruturas são implementadas em classes que estendem a classe base `Construct`. Você define uma construção instanciando a classe. Todas as estruturas usam três parâmetros ao serem inicializadas:

- escopo — O pai ou proprietário da construção. Isso pode ser uma pilha ou outra construção. O escopo determina o lugar da construção na árvore de construção. Normalmente, você deve passar `this` (`self`inPython), que representa o objeto atual, para o escopo.
- id — Um identificador que deve ser exclusivo dentro do escopo. O identificador serve como um namespace para tudo o que está definido na construção. Ele é usado para gerar identificadores exclusivos, como nomes de recursos e IDs AWS CloudFormation lógicos.

  Os identificadores só precisam ser exclusivos dentro de um escopo. Isso permite que você instancie e reutilize construções sem se preocupar com as construções e identificadores que elas possam conter, além de permitir a composição de construções em abstrações de alto nível. Além disso, os escopos possibilitam a referência a grupos de construções de uma só vez. Os exemplos incluem a marcação ou a especificação de onde as construções serão implantadas.
- props — Um conjunto de propriedades ou argumentos de palavras-chave, dependendo da linguagem, que definem a configuração inicial da construção. Construções de nível superior fornecem mais padrões e, se todos os elementos prop forem opcionais, você poderá omitir completamente o parâmetro props.

Configuração

A maioria das construções aceita `props` como terceiro argumento (ou, em Python, argumentos de palavra-chave), uma coleção de nome/valor que define a configuração da construção. O exemplo a seguir define um bucket com criptografia AWS Key Management Service (AWS KMS) e hospedagem estática de sites ativadas. Como não especifica explicitamente uma chave de criptografia, a `Bucket` construção define uma nova `kms.Key` e a associa ao 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"),
 })
```

Interagindo com construções

Construções são classes que estendem a classe Construct base. Depois de instanciar uma construção, o objeto de construção expõe um conjunto de métodos e propriedades que permitem que você interaja com a construção e a transmita como referência para outras partes do sistema.

A AWS CDK estrutura não impõe nenhuma restrição às APIs das construções. Os autores podem definir qualquer API que quiserem. No entanto, as AWS construções incluídas na AWS Construct Library, por exemplos`s3.Bucket`, seguem diretrizes e padrões comuns. Isso proporciona uma experiência consistente em todos os AWS recursos.

A maioria das AWS construções tem um conjunto de métodos de concessão que você pode usar para conceder permissões AWS Identity and Access Management (IAM) sobre essa construção a um diretor. O exemplo a seguir concede ao grupo do IAM `data-science` permissão para ler do bucket do 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)
```

Outro padrão comum é que AWS as construções definam um dos atributos do recurso a partir de dados fornecidos em outro lugar. Os atributos podem incluir nomes de recursos da Amazon (ARNs), nomes ou URLs.

O código a seguir define uma AWS Lambda função e a associa a uma fila do Amazon Simple Queue Service (Amazon SQS) por meio da URL da fila em uma variável de ambiente.

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()),
  },
 })
```

Para obter informações sobre os padrões de API mais comuns na AWS Construct Library, consultethe section called “Recursos”.

A construção do aplicativo e da pilha

As `Stack` classes `App` e da AWS Construct Library são construções exclusivas. Em comparação com outras construções, elas não configuram AWS recursos sozinhas. Em vez disso, eles são usados para fornecer contexto para suas outras construções. Todas as construções que representam AWS recursos devem ser definidas, direta ou indiretamente, dentro do escopo de uma `Stack` construção. `Stack`construções são definidas dentro do escopo de uma `App` construção.

Para saber mais sobre os aplicativos CDK, consulteAWS CDK aplicativos. Para saber mais sobre as pilhas de CDK, consulte. Pilhas

O exemplo a seguir define um aplicativo com uma única pilha. Dentro da pilha, uma construção L2 é usada para configurar um recurso de bucket do 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

Pilha definida no `HelloCdkStack.java` arquivo:

```
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();
    }
}
```

Aplicativo definido no `HelloCdkApp.java` arquivo:

```
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
}
```

Trabalhando com construções

Trabalhando com construções L1

L1 constrói mapas diretamente para recursos individuais AWS CloudFormation . Você deve fornecer a configuração necessária do recurso.

Neste exemplo, criamos um `bucket` objeto usando a construção `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"),
 })
```

Propriedades de construção que não são simples booleanos, cadeias de caracteres, números ou contêineres são tratadas de forma diferente nas linguagens suportadas.

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

Em Python, essas propriedades são representadas por tipos definidos como classes internas da construção L1. Por exemplo, a propriedade opcional `cors_configuration` de a `CfnBucket` requer um invólucro do tipo`CfnBucket.CorsConfigurationProperty`. Aqui estamos definindo `cors_configuration` em uma `CfnBucket` instância.

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

Java

Em Java, essas propriedades são representadas por tipos definidos como classes internas da construção L1. Por exemplo, a propriedade opcional `corsConfiguration` de a `CfnBucket`

requer um invólucro do tipo`CfnBucket.CorsConfigurationProperty`. Aqui estamos definindo `corsConfiguration` em uma `CfnBucket` instância.

```
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#

Em C#, essas propriedades são representadas por tipos definidos como classes internas da construção L1. Por exemplo, a propriedade opcional `CorsConfiguration` de a `CfnBucket` requer um invólucro do tipo`CfnBucket.CorsConfigurationProperty`. Aqui estamos definindo `CorsConfiguration` em uma `CfnBucket` instância.

```
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

Em Go, esses tipos são nomeados usando o nome da construção L1, um sublinhado e o nome da propriedade. Por exemplo, a propriedade opcional `CorsConfiguration` de a `CfnBucket`

requer um invólucro do tipo`CfnBucket_CorsConfigurationProperty`. Aqui estamos definindo `CorsConfiguration` em uma `CfnBucket` instância.

```
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

Você não pode usar tipos de propriedade L2 com construções L1 ou vice-versa. Ao trabalhar com construções L1, sempre use os tipos definidos para a construção L1 que você está usando. Não use tipos de outras construções L1 (algumas podem ter o mesmo nome, mas não são do mesmo tipo).

Algumas de nossas referências de API específicas da linguagem atualmente têm erros nos caminhos para os tipos de propriedades L1 ou não documentam essas classes. Esperamos corrigir isso em breve. Enquanto isso, lembre-se de que esses tipos são sempre classes internas da construção L1 com a qual são usados.

Trabalhando com construções L2

No exemplo a seguir, definimos um bucket do Amazon S3 criando um objeto a partir da construção `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`não é o nome do bucket que AWS CloudFormation cria. É um identificador lógico fornecido à nova construção dentro do contexto do seu aplicativo CDK. O valor PhysicalName será usado para nomear o AWS CloudFormation recurso.

Trabalhando com construções de terceiros

O Construct Hub é um recurso para ajudá-lo a descobrir construções adicionais de AWS terceiros e da comunidade CDK de código aberto.

Escrevendo suas próprias construções

Além de usar construções existentes, você também pode escrever suas próprias construções e permitir que qualquer pessoa as use em seus aplicativos. Todas as construções são iguais no AWS CDK. As construções da AWS Construct Library são tratadas da mesma forma que uma construção de uma biblioteca de terceiros publicada viaNPM, Maven ou. PyPI As construções publicadas no repositório interno de pacotes da sua empresa também são tratadas da mesma forma.

Para declarar uma nova construção, crie uma classe que estenda a classe base Construct no `constructs` pacote e siga o padrão dos argumentos do inicializador.

O exemplo a seguir mostra como declarar uma construção que representa um bucket do Amazon S3. O bucket do S3 envia uma notificação do Amazon Simple Notification Service (Amazon SNS) toda vez que alguém carrega um arquivo nele.

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**
>
> Nossa `NotifyingBucket` construção herda não de`Bucket`, mas sim de`Construct`. Estamos usando composição, não herança, para agrupar um bucket do Amazon S3 e um

tópico do Amazon SNS. Em geral, a composição é preferida à herança ao desenvolver AWS CDK construções.

O `NotifyingBucket` construtor tem uma assinatura de construção típica: `scope`id, e. `props` O último argumento,`props`, é opcional (obtém o valor padrão`{}`) porque todos os adereços são opcionais. (A `Construct` classe base não aceita `props` argumentos.) Você pode definir uma instância dessa construção em seu aplicativo sem`props`, por exemplo:

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)
```

Ou você pode usar `props` (em Java, um parâmetro adicional) para especificar o prefixo do caminho a ser filtrado, por exemplo:

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/"),
})
```

Normalmente, você também gostaria de expor algumas propriedades ou métodos em suas construções. Não é muito útil ter um tópico escondido atrás de sua construção, porque os usuários de sua construção não conseguem se inscrever nela. Adicionar uma `topic` propriedade permite que os consumidores acessem o tópico interno, conforme mostrado no exemplo a seguir:

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

Para fazer isso em Go, precisaremos de um pouco mais de encanamento. Nossa `NewNotifyingBucket` função original retornou uma`awss3.Bucket`. Precisaremos estender `Bucket` para incluir um `topic` membro criando uma `NotifyingBucket` estrutura. Nossa função então retornará esse tipo.

```
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
}
```

Agora, os consumidores podem se inscrever no tópico, por exemplo:

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))
```

Saiba mais

O vídeo a seguir fornece uma visão geral abrangente das construções de CDK e explica como você pode usá-las em seus aplicativos de CDK.

Explicação das construções do CDK

Ambientes do

Um ambiente é o alvo Conta da AWS e no Região da AWS qual as pilhas são implantadas. Todas as pilhas em seu aplicativo CDK estão explícita ou implicitamente associadas a um ambiente (). `env`

Tópicos

- Configurar ambientes
- Ambientes de bootstrap

Configurar ambientes

Para pilhas de produção, recomendamos que você especifique explicitamente o ambiente para cada pilha em seu aplicativo usando a propriedade. `env` O exemplo a seguir especifica ambientes diferentes para suas duas pilhas diferentes.

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 });
```

Quando você codifica permanentemente a conta e a região de destino, conforme mostrado no exemplo anterior, a pilha é sempre implantada nessa conta e região específicas. Para tornar a pilha implantável em um destino diferente, mas para determinar o destino no momento da síntese, sua pilha pode usar duas variáveis de ambiente fornecidas pela CLI: e. AWS CDK `CDK_DEFAULT_ACCOUNT` `CDK_DEFAULT_REGION` Essas variáveis são definidas com base no AWS perfil especificado usando a --profile opção ou no AWS perfil padrão, se você não especificar um.

O fragmento de código a seguir mostra como acessar a conta e a região passadas da AWS CDK CLI em sua pilha.

TypeScript

Acesse variáveis de ambiente por meio do `process` objeto do Node.

> **Note**
>
> Você precisa do `DefinitelyTyped` módulo para usar `process` em TypeScript. `cdk init`instala esse módulo para você. No entanto, você deve instalar esse módulo manualmente se estiver trabalhando com um projeto criado antes de ser adicionado ou se você não configurou seu projeto usando`cdk init`.
>
> ```
> npm install @types/node
> ```

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

JavaScript

Acesse variáveis de ambiente por meio do `process` objeto do Node.

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

Python

Use o `environ` dicionário do `os` módulo para acessar as variáveis de ambiente.

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

Java

Use `System.getenv()` para obter o valor de uma variável de ambiente.

```
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#

Use `System.Environment.GetEnvironmentVariable()` para obter o valor de uma variável de ambiente.

```
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() });
```

Especifique o Região da AWS uso de um código de região. Para obter uma lista, consulte Endpoints regionais.

A AWS CDK distinção entre não especificar a `env` propriedade e especificá-la usando e. `CDK_DEFAULT_ACCOUNT` `CDK_DEFAULT_REGION` O primeiro implica que a pilha deve sintetizar um modelo independente do ambiente. As construções definidas nessa pilha não podem usar nenhuma informação sobre seu ambiente. Por exemplo, você não pode escrever código como `if (stack.region === 'us-east-1')` nem usar recursos de estrutura como vpc.fromLookup (Python`from_lookup`:), que precisam consultar sua conta. AWS Esses recursos não funcionam até que você especifique um ambiente explícito; para usá-los, você deve especificar`env`.

Quando você passa em seu ambiente usando `CDK_DEFAULT_ACCOUNT` e`CDK_DEFAULT_REGION`, a pilha será implantada na conta e na região determinadas pela AWS CDK CLI no momento da

síntese. Isso permite que o código dependente do ambiente funcione, mas também significa que o modelo sintetizado pode ser diferente com base na máquina, no usuário ou na sessão em que ele é sintetizado. Esse comportamento geralmente é aceitável ou até desejável durante o desenvolvimento, mas provavelmente seria um antipadrão para uso em produção.

Você pode definir como `env` quiser, usando qualquer expressão válida. Por exemplo, você pode escrever sua pilha para suportar duas variáveis de ambiente adicionais para permitir que você substitua a conta e a região no momento da síntese. Vamos chamá-los `CDK_DEPLOY_ACCOUNT` e `CDK_DEPLOY_REGION` aqui, mas você pode nomeá-los como quiser, pois eles não são definidos pelo AWS CDK. No ambiente da pilha a seguir, variáveis de ambiente alternativas são usadas se estiverem definidas. Se não estiverem definidos, eles retornarão ao ambiente padrão fornecido pelo 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() });
```

Com o ambiente da sua pilha declarado dessa forma, você pode escrever um script curto ou um arquivo em lote, como o seguinte, para definir as variáveis a partir dos argumentos da linha de comando e, em seguida, chamar`cdk deploy`. Todos os argumentos além dos dois primeiros são passados para `cdk deploy` e podem ser usados para especificar opções de linha de comando ou pilhas.

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
```

Salve o script como `cdk-deploy-to.sh` e execute `chmod +x cdk-deploy-to.sh` para torná-lo executável.

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
}
```

A versão Windows do script é usada PowerShell para fornecer a mesma funcionalidade da versão macOS/Linux. Ele também contém instruções para permitir que ele seja executado como um

arquivo em lotes para que possa ser facilmente invocado a partir de uma linha de comando. Ele deve ser salvo como`cdk-deploy-to.bat`. O arquivo `cdk-deploy-to.ps1` será criado quando o arquivo em lotes for invocado.

Em seguida, você pode escrever scripts adicionais que chamem o script “deploy-to” para implantar em ambientes específicos (até mesmo vários ambientes por script):

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 %*
```

Ao implantar em vários ambientes, considere se você deseja continuar implantando em outros ambientes após uma falha na implantação. O exemplo a seguir evita a implantação no segundo ambiente de produção se o primeiro não for bem-sucedido.

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 %*
```

Os desenvolvedores ainda podem usar o `cdk deploy` comando normal para implantar em seus próprios AWS ambientes para desenvolvimento.

Se você não especificar um ambiente ao instanciar uma pilha, diz-se que a pilha é independente do ambiente. AWS CloudFormation os modelos sintetizados a partir dessa pilha tentarão usar a resolução do tempo de implantação em atributos relacionados ao ambiente, como`stack.account`, `stack.region` e (Python:). `stack.availabilityZones` `availability_zones`

Ao usar cdk deploy para implantar pilhas independentes do ambiente, eles AWS CDK CLI usarão o AWS CLI perfil especificado para determinar onde implantar. Se nenhum perfil for especificado, o perfil padrão será usado. AWS CDK CLISegue um protocolo semelhante ao AWS CLI para determinar quais AWS credenciais usar ao realizar operações em sua AWS conta. Para mais detalhes, consulte the section called “AWS CDK Kit de ferramentas”.

Em uma pilha independente do ambiente, qualquer construção que use zonas de disponibilidade verá duas zonas de disponibilidade, permitindo que a pilha seja implantada em qualquer região.

Ambientes de bootstrap

Você deve inicializar cada ambiente no qual implantará pilhas de CDK. O bootstrapping prepara o ambiente para implantação. Para saber mais, consulte Bootstrapping.

Bootstrapping

O bootstrapping é o processo de preparar um ambiente para implantação. O bootstrapping é uma ação única que você deve executar em cada ambiente em que você implanta recursos.

Tópicos

Ambientes de bootstrap

> **Important**
>
> Você pode incorrer em AWS cobranças pelos dados armazenados nos recursos inicializados.

O bootstrap provisiona recursos em seu ambiente, como um bucket do Amazon Simple Storage Service (Amazon S3) para armazenar arquivos AWS Identity and Access Management e funções (IAM) que concedem as permissões necessárias para realizar implantações. Esses recursos são provisionados em uma AWS CloudFormation pilha, chamada pilha de bootstrap. Geralmente é nomeado`CDKToolkit`. Como qualquer AWS CloudFormation pilha, ela aparecerá no AWS CloudFormation console do seu ambiente depois de implantada.

> **Note**
>
> O CDK v2 usa um modelo de bootstrap moderno. O modelo legado do CDK v1 não é suportado na v2.

Os ambientes são independentes. Se você quiser implantar em vários ambientes, cada ambiente deve ser inicializado separadamente.

Se você tentar implantar um aplicativo CDK em um ambiente que não tenha sido inicializado, você receberá uma mensagem de erro lembrando-o de inicializar o ambiente.

Inicialização com CDK Pipelines

Se você estiver usando o CDK Pipelines para implantar no ambiente de outra conta e receber uma mensagem como a seguinte:

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

Essa mensagem de erro significa que as funções apropriadas do IAM não existem no outro ambiente. A causa mais provável é que o ambiente não tenha sido inicializado. Inicialize o ambiente e tente novamente.

> **Note**
>
> Se o ambiente for inicializado, não exclua e recrie a pilha de bootstrap do ambiente. A exclusão da pilha de bootstrap excluirá os AWS recursos que foram originalmente provisionados no ambiente para oferecer suporte às implantações de CDK. Isso fará com que o pipeline pare de funcionar. Em vez disso, tente atualizar a pilha de bootstrap para uma nova versão executando o comando CDK CLI `cdk bootstrap` novamente.

Como inicializar

Quando você inicializa um ambiente, um AWS CloudFormation modelo é implantado no ambiente específico. Esse modelo provisiona recursos em sua conta para preparar seu ambiente para implantação.

O modelo de inicialização aceita parâmetros que personalizam alguns aspectos dos recursos inicializados. Para ter mais informações, consulte the section called “Personalizando o bootstrapping”.

Você pode inicializar de qualquer uma das seguintes formas:

- Use o comando AWS CDK CLI.cdk bootstrap Esse é o método mais simples e funciona bem se você tiver apenas alguns ambientes para inicializar.
- Implante o modelo fornecido pelo AWS CDK CLI usando outra ferramenta AWS CloudFormation de implantação. Isso permite que você use AWS CloudFormation StackSets ou AWS Control Tower e também o AWS CloudFormation console ou AWS CLI o. Você pode fazer pequenas modificações no modelo antes da implantação. Essa abordagem é mais flexível e adequada para implantações em grande escala.

Não é um erro inicializar um ambiente mais de uma vez. Se um ambiente que você inicializou já tiver sido inicializado, sua pilha de bootstrap será atualizada, se necessário. Caso contrário, nada acontecerá.

Inicializando com o AWS CDKCLI

Use o `cdk bootstrap` comando para inicializar um ou mais AWS ambientes.

O exemplo a seguir inicializa dois ambientes:

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

Os exemplos a seguir mostram várias maneiras de inicializar ambientes. Conforme mostrado no segundo exemplo, o `aws://` prefixo é opcional ao especificar um ambiente.

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

Quando você executacdk bootstrap, o CDK CLI sempre sintetiza o aplicativo CDK no diretório atual. Se você não especificar pelo menos um ambiente, o CDK CLI inicializará todos os ambientes referenciados no aplicativo.

Para pilhas independentes do ambiente, o CDK CLI tentará determinar um ambiente a partir de fontes padrão. Isso pode ser um ambiente especificado usando a --profile opção, de variáveis de ambiente ou AWS CLI fontes padrão. Se encontrado, o ambiente é então inicializado.

Por exemplo, o comando a seguir sintetiza o AWS CDK aplicativo atual usando o `prod` AWS perfil e, em seguida, inicializa seus ambientes.

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

Inicialização a partir do modelo AWS CloudFormation

Você pode inicializar um ambiente obtendo e implantando o modelo de bootstrap AWS CloudFormation .

Para obter uma cópia desse modelo no arquivo`bootstrap-template.yaml`, execute o seguinte comando:

macOS/Linux

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

Windows

No Windows, PowerShell deve ser usado para preservar a codificação do modelo.

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

O modelo também está disponível no AWS CDK GitHub repositório.

Implante esse modelo usando a CLI do CDK ou seu mecanismo AWS CloudFormation de implantação preferido para modelos. Para implantar usando a CLI do CDK, execute. cdk bootstrap --template ***TEMPLATE_FILENAME*** Você também pode implantá-lo usando o AWS CLI executando o comando abaixo ou implantá-lo em uma ou mais contas ao mesmo tempo usando 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
```

Personalizando o bootstrapping

Há duas maneiras de personalizar a inicialização de recursos em seu ambiente:

- Use os parâmetros da linha de comando com o `cdk bootstrap` comando. Isso permite que você modifique alguns aspectos do modelo.
- Modifique o modelo de bootstrap padrão e implante-o você mesmo. Isso lhe dá um controle mais completo sobre os recursos de bootstrap.

As seguintes opções de linha de comando, quando usadas com o CDK CLIcdk bootstrap, fornecem ajustes comumente usados no modelo de inicialização:

- - -bootstrap-bucket-name substitui o nome do bucket do Amazon S3. Pode exigir alterações em seu aplicativo CDK (consultethe section called “Sintetizadores Stack”).

- --bootstrap-kms-key-idsubstitui a AWS KMS chave usada para criptografar o bucket do S3.
- --cloudformation-execution-policiesespecifica os ARNs das políticas gerenciadas que devem ser anexadas à função de implantação assumida AWS CloudFormation durante a implantação de suas pilhas. Por padrão, as pilhas são implantadas com permissões totais de administrador usando a `AdministratorAccess` política.

  Os ARNs da política devem ser passados como um único argumento de string, com os ARNs individuais separados por vírgulas. Por exemplo: .

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

  > **Important**
  >
  > Para evitar falhas na implantação, certifique-se de que as políticas especificadas sejam suficientes para qualquer implantação que você executará no ambiente que está sendo inicializado.

- --qualifieré uma string adicionada aos nomes de todos os recursos na pilha de bootstrap. Um qualificador permite evitar conflitos de nomes de recursos ao provisionar várias pilhas de bootstrap no mesmo ambiente. O padrão é `hnb659fds` (esse valor não tem significado).

  A alteração do qualificador também exige que seu aplicativo CDK transmita o valor alterado para o sintetizador de pilha. Para ter mais informações, consulte the section called “Sintetizadores Stack”.
- --tagsadiciona uma ou mais AWS CloudFormation tags à pilha de bootstrap.
- --trustlista as AWS contas que podem ser implantadas no ambiente que está sendo inicializado.

  Use esse sinalizador ao inicializar um ambiente no qual um CDK Pipeline em outro ambiente será implantado. A conta que faz o bootstrapping é sempre confiável.
- --trust-for-lookuplista as AWS contas que podem pesquisar informações de contexto do ambiente que está sendo inicializado.

  Use esse sinalizador para dar permissão às contas para sintetizar pilhas que serão implantadas no ambiente, sem realmente dar a elas permissão para implantar essas pilhas diretamente.
- --termination-protectionimpede que a pilha de bootstrap seja excluída. Para obter mais informações, consulte Protegendo uma pilha de ser excluída no Guia do AWS CloudFormation usuário.

> **Important**
>
> O modelo de bootstrap moderno concede efetivamente as permissões implícitas no `--cloudformation-execution-policies` para qualquer AWS conta na `--trust` lista. Por padrão, isso estende as permissões de leitura e gravação em qualquer recurso na conta inicializada. Certifique-se de configurar a pilha de inicialização com políticas e contas confiáveis com as quais você se sinta confortável.

Personalizar o modelo

Quando precisar de mais personalização do que o CDK CLI pode oferecer, você pode modificar o modelo de bootstrap para atender às suas necessidades. Primeiro, você obtém o modelo usando a --show-template opção. Veja um exemplo a seguir:

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

Todas as modificações que você fizer devem seguir o modelo de contrato de bootstrapping. Para garantir que suas personalizações não sejam substituídas acidentalmente posteriormente por alguém executando cdk bootstrap o modelo padrão, altere o valor padrão do parâmetro do modelo. `BootstrapVariant` A CLI do CDK só permitirá sobrescrever a pilha de bootstrap por modelos que tenham a `BootstrapVariant` mesma versão e uma versão igual ou superior ao modelo atualmente implantado.

Em seguida, você pode implantar seu modelo modificado conforme descrito emthe section called “Inicialização a partir do modelo AWS CloudFormation”, ou usandocdk bootstrap --template.

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

Diferenças de modelos de bootstrap

Conforme mencionado anteriormente, a AWS CDK v1 suportava dois modelos de bootstrap, antigos e modernos. O CDK v2 suporta somente o modelo moderno. Para referência, aqui estão as diferenças de alto nível entre esses dois modelos.

Atributo	Legacy (somente v1)	Moderno (v1 e v2)
Implantações entre contas	Não permitido	Permitido

Atributo	Legacy (somente v1)	Moderno (v1 e v2)
AWS CloudFormation Permissões	Implanta usando as permissões atuais do usuário (determinadas pelo AWS perfil, variáveis de ambiente etc.)	Implanta usando as permissões especificadas quando a pilha de bootstrap foi provisionada (por exemplo, usando) `--trust`
Versionamento	Apenas uma versão da pilha de bootstrap está disponível	A pilha do Bootstrap tem versão; novos recursos podem ser adicionados em versões futuras e os AWS CDK aplicativos podem exigir uma versão mínima
Recursos *	Bucket do Amazon S3	Bucket do Amazon S3
		AWS KMS key
		Perfis do IAM
		Repositório Amazon ECR
		Parâmetro SSM para controle de versão
Nomeação de recursos	Gerado automaticamente	Deterministic
Criptografia de bucket	Tecla padrão	Chave gerenciada pelo cliente

* Adicionaremos recursos adicionais ao modelo de bootstrap conforme necessário.

Um ambiente que foi inicializado usando o modelo legado deve ser atualizado para usar o modelo moderno do CDK v2 por meio da reinicialização. Reimplante todos os AWS CDK aplicativos no ambiente pelo menos uma vez antes de excluir o bucket legado.

Sintetizadores Stack

Seu AWS CDK aplicativo precisa conhecer os recursos de inicialização disponíveis para sintetizar com êxito uma pilha que possa ser implantada. O sintetizador de pilha é uma AWS CDK classe que controla como o modelo da pilha é sintetizado. Isso inclui como ele usa recursos de bootstrap (por exemplo, como se refere aos ativos armazenados no bucket de bootstrap).

Os AWS CDK sintetizadores de pilha integrados são chamados. `DefaultStackSynthesizer` Ele inclui recursos para implantações entre contas e implantações do CDK Pipelines.

Você pode passar um sintetizador de pilha para uma pilha ao instanciá-lo usando a propriedade. `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
    })
});
```

Se você não fornecer a `synthesizer` propriedade, `DefaultStackSynthesizer` é usada.

Personalizando a síntese

Dependendo das alterações feitas no modelo de bootstrap, talvez você também precise personalizar a síntese. O `DefaultStackSynthesizer` pode ser personalizado usando as propriedades descritas a seguir.

Se nenhuma dessas propriedades fornecer as personalizações necessárias, você poderá escrever seu sintetizador como uma classe que implementa `IStackSynthesizer` (talvez derivada de). `DefaultStackSynthesizer`

Alterando o qualificador

O qualificador é adicionado ao nome dos recursos de bootstrap para distinguir os recursos em pilhas de bootstrap separadas. Para implantar duas versões diferentes da pilha de bootstrap no mesmo ambiente (AWS conta e região), as pilhas devem ter qualificadores diferentes.

Esse recurso é destinado ao isolamento de nomes entre testes automatizados do próprio CDK. A menos que você possa definir com precisão as permissões do IAM concedidas à função de AWS CloudFormation execução, não há benefícios de isolamento de permissão em ter duas pilhas de

bootstrap diferentes em uma única conta. Portanto, geralmente não há necessidade de alterar esse valor.

Para alterar o qualificador, configure o `DefaultStackSynthesizer` either instanciando o sintetizador com a propriedade:

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"
    })
});
```

Ou configurando o qualificador como uma chave de contexto em. `cdk.json`

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

Alterando os nomes dos recursos

Todas as outras `DefaultStackSynthesizer` propriedades estão relacionadas aos nomes dos recursos no modelo de inicialização. Você só precisa fornecer qualquer uma dessas propriedades se tiver modificado o modelo de bootstrap e alterado os nomes dos recursos ou o esquema de nomenclatura.

Todas as propriedades aceitam os espaços reservados especiais `${Qualifier}${AWS::Partition}`,`${AWS::AccountId}`, e. `${AWS::Region}` Esses espaços reservados são substituídos pelos valores do `qualifier` parâmetro e pelos valores da AWS partição, ID da conta e região do ambiente da pilha, respectivamente.

O exemplo a seguir mostra as propriedades mais usadas `DefaultStackSynthesizer` junto com seus valores padrão, como se você estivesse instanciando o sintetizador. Para obter uma lista completa, consulte 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,
})
```

O modelo de contrato de bootstrapping

Os requisitos da pilha de inicialização dependem do sintetizador de pilha em uso. Se você escrever seu próprio sintetizador de pilha, terá controle total dos recursos de bootstrap que seu sintetizador requer e como o sintetizador os encontra.

Esta seção descreve as expectativas que eles `DefaultStackSynthesizer` têm do modelo de bootstrapping.

Versionamento

O modelo deve conter um recurso para criar um parâmetro SSM com um nome conhecido e uma saída que reflita a versão do modelo.

```
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]
```

Funções

`DefaultStackSynthesizer`Isso requer cinco funções do IAM para cinco finalidades diferentes. Se você não estiver usando as funções padrão, deverá informar ao sintetizador os ARNs das funções que deseja usar.

As funções são as seguintes:

- A função de implantação é assumida pelo AWS CDK kit de ferramentas e pela AWS CodePipeline implantação em um ambiente. Ele `AssumeRolePolicy` controla quem pode ser implantado no ambiente. No modelo, você pode ver as permissões que essa função precisa.
- A função de pesquisa é assumida pelo AWS CDK Toolkit para realizar pesquisas de contexto em um ambiente. Ele `AssumeRolePolicy` controla quem pode ser implantado no ambiente. As permissões que essa função precisa podem ser vistas no modelo.

- A função de publicação de arquivos e a função de publicação de imagens são assumidas pelo AWS CDK kit de ferramentas e pelos AWS CodeBuild projetos para publicar ativos em um ambiente. Eles são usados para gravar no bucket do S3 e no repositório ECR, respectivamente. Essas funções exigem acesso de gravação a esses recursos.
- A função de AWS CloudFormation execução é passada AWS CloudFormation para realizar a implantação real. Suas permissões são as permissões sob as quais a implantação será executada. As permissões são passadas para a pilha como um parâmetro que lista os ARNs de políticas gerenciadas.

Outputs

O AWS CDK kit de ferramentas exige que as seguintes CloudFormation saídas existam na pilha de bootstrap.

- `BucketName`: o nome do repositório de ativos do arquivo
- `BucketDomainName`: o bucket de ativos de arquivo em formato de nome de domínio
- `BootstrapVersion`: a versão atual da pilha de bootstrap

Histórico do modelo

O modelo de bootstrap é versionado e evolui ao longo do tempo com o próprio. AWS CDK Se você fornecer seu próprio modelo de bootstrap, mantenha-o atualizado com o modelo padrão canônico. Você quer garantir que seu modelo continue funcionando com todos os recursos do CDK.

> **Note**
>
> As versões anteriores do modelo de bootstrap criavam um AWS KMS key em cada ambiente inicializado por padrão. Para evitar cobranças pela chave KMS, reinicialize esses ambientes usando o. `--no-bootstrap-customer-key` O padrão atual é a ausência de chave KMS, o que ajuda a evitar essas cobranças.

Esta seção contém uma lista das alterações feitas em cada versão.

Versão do modelo	AWS CDK versão	Alterações
1	1.40.0	Versão inicial do modelo com Bucket, Key, Repository e Roles.
2	1.45,0	Divida a função de publicaçã o de ativos em funções separadas de publicação de arquivos e imagens.
3	1.46.0	Adicione `FileAsset KeyArn` exportação para poder adicionar permissões de descriptografia aos consumido res de ativos.
4	1.61.0	AWS KMS as permissões agora estão implícitas via Amazon S3 e não são mais necessárias. `FileAsetK eyArn` Adicione o parâmetro `CdkBootstrapVersion` SSM para que a versão da pilha de bootstrap possa ser verificada sem saber o nome da pilha.
5	1.87.0	A função de implantação pode ler o parâmetro SSM.
6	1.108.0	Adicione uma função de pesquisa separada da função de implantação.
6	1.109.0	Anexe a `aws-cdk:b ootstrap-role` tag às funções de implantação,

Versão do modelo	AWS CDK versão	Alterações
		publicação de arquivos e publicação de imagens.
7	1.10.0	A função de implantação não pode mais ler buckets diretamente na conta de destino. (No entanto, essa função é efetivamente de administrador e, de qualquer forma, sempre pode usar suas AWS CloudFormation permissões para tornar o bucket legível).
8	1.14.0	A função de pesquisa tem permissões completas de somente leitura para o ambiente de destino e também tem uma `aws-cdk:bootstrap-role` tag.
9	2.1.0	Impede que os uploads de ativos do Amazon S3 sejam rejeitados pela criptografia SCP comumente referenciada.
10	2.4.0	O Amazon ECR agora ScanOnPush está habilitado por padrão.
11	2.18.0	Adiciona uma política que permite que o Lambda extraia dos repositórios do Amazon ECR para que ele sobreviva à reinicialização.

Versão do modelo	AWS CDK versão	Alterações
12	2.20.0	Adiciona suporte para testes experimentaiscdk import.
13	2.25.0	Torna imutáveis as imagens de contêiner nos repositórios Amazon ECR criados pelo bootstrap.
14	2.34.0	Por padrão, desativa a digitalização de imagens do Amazon ECR no nível do repositório para permitir a inicialização de regiões que não oferecem suporte à digitalização de imagens.
15	2.60,0	As chaves KMS não podem ser marcadas.
16	2.69,0	Endereça o Security Hub que encontra o KMS.2.
17	2.72.0	Endereça a descoberta do Security Hub pela ECR.3.
18	2.80,0	Alterações revertidas feitas para a versão 16, pois elas não funcionam em todas as partições e não são recomendadas.
19	2.106.1	Alterações revertidas feitas na versão 18, em que a AccessControl propriedade foi removida do modelo. (#27964)

Versão do modelo	AWS CDK versão	Alterações
20	2.119.0	Adicione `ssm:GetParameters` ação à função de AWS CloudFormation implantação do IAM. Para obter mais informações, consulte #28336.

Conclusões do Security Hub

Se você estiver usando AWS Security Hub, poderá ver descobertas relatadas sobre alguns dos recursos criados pelo processo de AWS CDK Bootstrapping. As descobertas do Security Hub ajudam você a encontrar configurações de recursos que você deve verificar novamente quanto à precisão e segurança. Analisamos essas configurações específicas de recursos com a AWS Security e temos certeza de que elas não constituem um problema de segurança.

As entidades principais do IAM não devem ter políticas incorporadas do IAM que permitam ações de descriptografia em todas as chaves do KMS

A função Deploy (nome padrão`cdk-hnb659fds-deploy-role-ACCOUNT-REGION`) tem permissões para ler dados criptografados armazenados no Amazon S3. A política não dá permissão a nenhum dado por si só: somente dados lidos do Amazon S3 podem ser descriptografados e somente de buckets que permitem explicitamente que a Deploy Role os leia por meio de sua Política de Bucket, e chaves que permitem explicitamente que a Deploy Role decodifique usando-os usando sua Política de Chaves. Essa declaração é usada para permitir que os AWS CDK Pipelines realizem implantações em várias contas.

Por que o Security Hub sinaliza isso? A política contém uma `Condition` cláusula `Resource: *` combinada com uma; o Security Hub está sinalizando o. `*` Isso `*` é necessário porque, no momento em que a conta é inicializada, a AWS KMS chave criada pelo AWS CDK Pipelines para o CodePipeline Artifact Bucket ainda não existe, então não podemos referenciar seu ARN. Além disso, o Security Hub não inclui a `Condition` cláusula na declaração de política em seu raciocínio.

E se eu quiser corrigir essa descoberta? Desde que as políticas de recursos em suas AWS KMS chaves não sejam desnecessariamente permissivas, a política de função atual não permite que a função de implantação acesse mais dados do que deveria. Se você ainda quiser se livrar da

descoberta, pode fazer isso personalizando a pilha de bootstrap (usando o processo descrito acima) de uma dessas duas maneiras:

- Se você não estiver usando AWS CDK Pipelines para implantações entre contas: remova a instrução com `Sid: PipelineCrossAccountArtifactsBucket` da função de implantação; ou
- Se você estiver usando AWS CDK Pipelines para implantações entre contas: depois de implantar seu AWS CDK Pipeline, procure o AWS KMS ARN da chave do Artefact Bucket e substitua a instrução pelo ARN `Resource: *` da chave real. `Sid: PipelineCrossAccountArtifactsBucket`

Recursos

Recursos são o que você configura para usar Serviços da AWS em seus aplicativos. Os recursos são uma característica do AWS CloudFormation. Ao configurar recursos e suas propriedades em um AWS CloudFormation modelo, você pode implantar para AWS CloudFormation provisionar seus recursos. Com o AWS Cloud Development Kit (AWS CDK), você pode configurar recursos por meio de construções. Em seguida, você implanta seu aplicativo CDK, o que envolve sintetizar um AWS CloudFormation modelo e implantá-lo para AWS CloudFormation provisionar seus recursos.

Tópicos

Configurando recursos usando construções

Conforme descrito emthe section called “Construções”, AWS CDK fornece uma rica biblioteca de classes de construções, chamadas de AWS construções, que representam todos os AWS recursos.

Para criar uma instância de um recurso usando sua construção correspondente, passe o escopo como o primeiro argumento, a ID lógica da construção e um conjunto de propriedades de configuração (props). Por exemplo, veja como criar uma fila do Amazon SQS com AWS KMS criptografia usando a construção SQS.Queue da 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
});
```

Alguns adereços de configuração são opcionais e, em muitos casos, têm valores padrão. Em alguns casos, todos os adereços são opcionais e o último argumento pode ser totalmente omitido.

Atributos de recursos

A maioria dos recursos na AWS Construct Library expõe atributos, que são resolvidos no momento da implantação pelo AWS CloudFormation. Os atributos são expostos na forma de propriedades nas classes de recursos com o nome do tipo como prefixo. O exemplo a seguir mostra como obter a URL de uma fila do Amazon SQS usando a propriedade (`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
```

Consulte the section called “Tokens” para obter informações sobre como o AWS CDK codifica atributos de tempo de implantação como cadeias de caracteres.

Recursos de referência

Ao configurar recursos, muitas vezes você precisará referenciar propriedades de outro recurso. Veja os exemplos a seguir:

- Um recurso do Amazon Elastic Container Service (Amazon ECS) requer uma referência ao cluster no qual ele é executado.
- Uma CloudFront distribuição da Amazon exige uma referência ao bucket do Amazon Simple Storage Service (Amazon S3) contendo o código-fonte.

Você pode referenciar recursos de qualquer uma das seguintes formas:

- Ao passar um recurso definido em seu aplicativo CDK, na mesma pilha ou em uma diferente
- Ao passar um objeto proxy referenciando um recurso definido em sua AWS conta, criado a partir de um identificador exclusivo do recurso (como um ARN)

Se a propriedade de uma construção representa uma construção para outro recurso, seu tipo é o tipo de interface da construção. Por exemplo, a construção do Amazon ECS assume uma propriedade `cluster` do tipo`ecs.ICluster`. Outro exemplo é a construção CloudFront de distribuição que usa uma propriedade `sourceBucket` (Python:`source_bucket`) do tipo. `s3.IBucket`

Você pode transmitir diretamente qualquer objeto de recurso do tipo adequado definido no mesmo AWS CDK aplicativo. O exemplo a seguir define um cluster do Amazon ECS e o usa para definir um serviço do 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 });
```

Referenciando recursos em uma pilha diferente

Você pode se referir aos recursos em uma pilha diferente, desde que estejam definidos no mesmo aplicativo e no mesmo AWS ambiente. O padrão a seguir é geralmente usado:

- Armazene uma referência à construção como um atributo da pilha que produz o recurso. (Para obter uma referência à pilha da construção atual, use`Stack.of(this)`.)
- Passe essa referência para o construtor da pilha que consome o recurso como parâmetro ou propriedade. A pilha consumidora então a passa como uma propriedade para qualquer construção que precise dela.

O exemplo a seguir define uma pilha`stack1`. Essa pilha define um bucket do Amazon S3 e armazena uma referência à construção do bucket como um atributo da pilha. Em seguida, o

aplicativo define uma segunda pilha,`stack2`, que aceita um bucket na instanciação. `stack2`pode, por exemplo, definir uma AWS Glue tabela que usa o bucket para armazenamento de dados.

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});
```

Se AWS CDK determinar que o recurso está no mesmo ambiente, mas em uma pilha diferente, ele sintetiza automaticamente AWS CloudFormation as exportações na pilha de produção e um Fn:: ImportValue na pilha consumidora para transferir essas informações de uma pilha para a outra.

Resolvendo impasses de dependência

Fazer referência a um recurso de uma pilha em outra pilha cria uma dependência entre as duas pilhas. Isso garante que eles sejam implantados na ordem correta. Depois que as pilhas são implantadas, essa dependência é concreta. Depois disso, remover o uso do recurso compartilhado da pilha consumidora pode causar uma falha inesperada na implantação. Isso acontece se houver outra dependência entre as duas pilhas que as força a serem implantadas na mesma ordem. Isso

também pode acontecer sem dependência se a pilha de produção for simplesmente escolhida pelo CDK Toolkit para ser implantada primeiro. A AWS CloudFormation exportação é removida da pilha de produção porque não é mais necessária, mas o recurso exportado ainda está sendo usado na pilha de consumo porque sua atualização ainda não foi implantada. Portanto, a implantação da pilha do produtor falha.

Para resolver esse impasse, remova o uso do recurso compartilhado da pilha de consumo. (Isso remove a exportação automática da pilha de produção.) Em seguida, adicione manualmente a mesma exportação à pilha de produção usando exatamente a mesma ID lógica da exportação gerada automaticamente. Remova o uso do recurso compartilhado na pilha de consumo e implante as duas pilhas. Em seguida, remova a exportação manual (e o recurso compartilhado, se não for mais necessário) e implante as duas pilhas novamente. O `exportValue()` método da pilha é uma maneira conveniente de criar a exportação manual para essa finalidade. (Veja o exemplo na referência do método vinculado.)

Referenciando recursos em sua conta AWS

Suponha que você queira usar um recurso já disponível na sua AWS conta no seu AWS CDK aplicativo. Isso pode ser um recurso definido por meio do console, de um AWS SDK, diretamente com AWS CloudFormation ou em um AWS CDK aplicativo diferente. Você pode transformar o ARN do recurso (ou outro atributo de identificação ou grupo de atributos) em um objeto proxy. O objeto proxy serve como referência ao recurso chamando um método estático de fábrica na classe do recurso.

Quando você cria esse proxy, o recurso externo não se torna parte do seu AWS CDK aplicativo. Portanto, as alterações feitas no proxy em seu AWS CDK aplicativo não afetam o recurso implantado. No entanto, o proxy pode ser passado para qualquer AWS CDK método que exija um recurso desse tipo.

O exemplo a seguir mostra como referenciar um bucket com base em um bucket existente com o ARN arn:aws:s3::: e uma my-bucket-name Amazon Virtual Private Cloud com base em uma VPC existente com um ID específico.

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"
});
```

Vamos dar uma olhada mais de perto no `Vpc.fromLookup()`método. Como a `ec2.Vpc` construção é complexa, talvez você queira selecionar a VPC de várias maneiras para ser usada com seu aplicativo CDK. Para resolver isso, a construção da VPC tem um método `fromLookup` estático (Python:`from_lookup`) que permite pesquisar a Amazon VPC desejada consultando sua conta no momento da síntese. AWS

Para ser usado`Vpc.fromLookup()`, o sistema que sintetiza a pilha deve ter acesso à conta proprietária da Amazon VPC. Isso ocorre porque o CDK Toolkit consulta a conta para encontrar a Amazon VPC certa no momento da síntese.

Além disso, `Vpc.fromLookup()` funciona somente em pilhas definidas com uma conta e uma região explícitas (consultethe section called “Ambientes do”). Se AWS CDK tentar pesquisar uma Amazon VPC a partir de uma pilha independente do ambiente, o CDK Toolkit não sabe qual ambiente consultar para encontrar a VPC.

Você deve fornecer `Vpc.fromLookup()` atributos suficientes para identificar de forma exclusiva uma VPC em sua conta. AWS Por exemplo, só pode haver uma VPC padrão, então basta especificar a VPC como padrão.

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 });
```

Você também pode usar a `tags` propriedade para consultar VPCs por tag. Você pode adicionar tags à Amazon VPC no momento de sua criação usando AWS CloudFormation ou o. AWS CDK Você pode editar as tags a qualquer momento após a criação usando o AWS Management Console AWS CLI, o ou um AWS SDK. Além de todas as tags que você mesmo adiciona, o adiciona AWS CDK automaticamente as seguintes tags a todas as VPCs criadas.

- Nome — O nome da VPC.
- aws-cdk:subnet-name — O nome da sub-rede.
- aws-cdk:subnet-type — O tipo da sub-rede: pública, privada ou isolada.

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" });
```

Os resultados de `Vpc.fromLookup()` são armazenados em cache no `cdk.context.json` arquivo do projeto. (Consulte the section called “Contexto”.) Confirme esse arquivo com o controle de versão para que seu aplicativo continue se referindo à mesma Amazon VPC. Isso funciona mesmo se você alterar posteriormente os atributos de suas VPCs de uma forma que resultaria na seleção de uma VPC diferente. Isso é particularmente importante se você estiver implantando a pilha em um ambiente que não tem acesso à AWS conta que define a VPC, como o CDK Pipelines.

Embora você possa usar um recurso externo em qualquer lugar em que usaria um recurso semelhante definido no seu AWS CDK aplicativo, não é possível modificá-lo. Por exemplo, chamar `addToResourcePolicy` (Python:`add_to_resource_policy`) em um externo `s3.Bucket` não faz nada.

Nomes físicos de recursos

Os nomes lógicos dos recursos em AWS CloudFormation são diferentes dos nomes dos recursos que são mostrados AWS Management Console depois de serem implantados pelo AWS CloudFormation. Eles AWS CDK chamam esses nomes finais de nomes físicos.

Por exemplo, AWS CloudFormation pode criar o bucket do Amazon S3 com o ID lógico `Stack2MyBucket4DD88B4F` do exemplo anterior com o nome físico. `stack2mybucket4dd88b4f-iuv1rbv9z3to`

Você pode especificar um nome físico ao criar construções que representam recursos usando a propriedade <resourceType>Name. O exemplo a seguir cria um bucket do Amazon S3 com o nome físico. `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" });
```

A atribuição de nomes físicos aos recursos tem algumas desvantagens em AWS CloudFormation. Mais importante ainda, qualquer alteração nos recursos implantados que exija a substituição de um recurso, como alterações nas propriedades de um recurso que são imutáveis após a criação, falhará se um recurso tiver um nome físico atribuído. Se você acabar nesse estado, a única solução é excluir a AWS CloudFormation pilha e implantar o AWS CDK aplicativo novamente. Consulte a AWS CloudFormation documentação para obter detalhes.

Em alguns casos, como ao criar um AWS CDK aplicativo com referências entre ambientes, nomes físicos são necessários para AWS CDK que o funcione corretamente. Nesses casos, se você não

quiser se preocupar em criar um nome físico, pode deixar o AWS CDK nome para você. Para fazer isso, use o valor especial`PhysicalName.GENERATE_IF_NEEDED`, da seguinte forma.

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 });
```

Passando identificadores de recursos exclusivos

Sempre que possível, você deve passar recursos por referência, conforme descrito na seção anterior. No entanto, há casos em que você não tem outra escolha a não ser se referir a um recurso por meio de um de seus atributos. Exemplos de casos de uso incluem o seguinte:

- Quando você está usando AWS CloudFormation recursos de baixo nível.
- Quando você precisa expor recursos aos componentes de tempo de execução de um AWS CDK aplicativo, como ao se referir às funções do Lambda por meio de variáveis de ambiente.

Esses identificadores estão disponíveis como atributos nos recursos, como os seguintes.

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

A AWS CDK vinculação Java usa métodos getter para atributos.

```
bucket.getBucketName()
lambdaFunc.getFunctionArn()
securityGroup.getGroupArn()
```

C#

```
bucket.BucketName
lambdaFunc.FunctionArn
securityGroup.GroupArn
```

O exemplo a seguir mostra como passar um nome de bucket gerado para uma AWS Lambda função.

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
```

```
    }
});
```

Concedendo permissões entre recursos

Construções de alto nível possibilitam permissões com privilégios mínimos, oferecendo APIs simples e baseadas em intenção para expressar os requisitos de permissão. Por exemplo, muitas construções de L2 oferecem métodos de concessão que você pode usar para conceder permissão a uma entidade (como um papel ou usuário do IAM) para trabalhar com o recurso, sem precisar criar manualmente declarações de permissão do IAM.

O exemplo a seguir cria as permissões para permitir que a função de execução de uma função Lambda leia e grave objetos em um determinado bucket do Amazon S3. Se o bucket do Amazon S3 for criptografado com uma AWS KMS chave, esse método também concederá permissões à função de execução da função Lambda para decodificar com a chave.

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)
{
    // ...
}
```

Os métodos de concessão retornam um `iam.Grant` objeto. Use o `success` atributo do `Grant` objeto para determinar se a concessão foi efetivamente aplicada (por exemplo, ela pode não ter sido aplicada em recursos externos). Você também pode usar o método `assertSuccess` (Python:`assert_success`) do `Grant` objeto para garantir que a concessão tenha sido aplicada com sucesso.

Se um método de concessão específico não estiver disponível para o caso de uso específico, você poderá usar um método de concessão genérico para definir uma nova concessão com uma lista específica de ações.

O exemplo a seguir mostra como conceder a uma função Lambda acesso à ação do 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");
```

Muitos recursos, como as funções Lambda, exigem que uma função seja assumida ao executar o código. Uma propriedade de configuração permite que você especifique um`iam.IRole`. Se nenhuma função for especificada, a função criará automaticamente uma função específica para esse uso. Em seguida, você pode usar métodos de concessão nos recursos para adicionar declarações à função.

Os métodos de concessão são criados usando APIs de nível inferior para lidar com as políticas do IAM. As políticas são modeladas como PolicyDocumentobjetos. Adicione declarações diretamente às funções (ou à função anexada de uma construção) usando o `addToRolePolicy` método (Python:`add_to_role_policy`) ou à política de um recurso (como uma `Bucket` política) usando o método (`addToResourcePolicy`Python:). `add_to_resource_policy`

Métricas e alarmes de recursos

Muitos recursos emitem CloudWatch métricas que podem ser usadas para configurar painéis de monitoramento e alarmes. Construções de nível superior têm métodos métricos que permitem acessar as métricas sem procurar o nome correto a ser usado.

O exemplo a seguir mostra como definir um alarme quando a fila `ApproximateNumberOfMessagesNotVisible` de uma fila do Amazon SQS excede 100.

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,
    // ..
});
```

Se não houver um método para uma métrica específica, você poderá usar o método métrico geral para especificar o nome da métrica manualmente.

As métricas também podem ser adicionadas aos CloudWatch painéis. Consulte CloudWatch.

Tráfego de rede

Em muitos casos, você deve habilitar permissões em uma rede para que um aplicativo funcione, como quando a infraestrutura computacional precisa acessar a camada de persistência. Recursos que estabelecem ou escutam conexões expõem métodos que permitem fluxos de tráfego, incluindo a definição de regras de grupos de segurança ou ACLs de rede.

Os recursos IConnectable têm uma `connections` propriedade que é a porta de entrada para a configuração das regras de tráfego de rede.

Você permite que os dados fluam em um determinado caminho de rede usando `allow` métodos. O exemplo a seguir permite conexões HTTPS com a web e conexões de entrada do grupo Amazon EC2 Auto Scaling. `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());
```

Alguns recursos têm portas padrão associadas a eles. Os exemplos incluem o ouvinte de um balanceador de carga na porta pública e as portas nas quais o mecanismo de banco de dados aceita conexões para instâncias de um banco de dados do Amazon RDS. Nesses casos, você pode aplicar um controle rígido da rede sem precisar especificar manualmente a porta. Para fazer isso, use os `allowToDefaultPort` métodos `allowDefaultPortFrom` e (Python:`allow_default_port_from`,`allow_to_default_port`).

O exemplo a seguir mostra como habilitar conexões de qualquer endereço IPV4 e uma conexão de um grupo de Auto Scaling para acessar um banco de dados.

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");
```

Tratamento de eventos

Alguns recursos podem atuar como fontes de eventos. Use o `addEventNotification` método (Python:`add_event_notification`) para registrar um destino de evento para um determinado tipo de evento emitido pelo recurso. Além disso, `addXxxNotification` os métodos oferecem uma maneira simples de registrar um manipulador para tipos de eventos comuns.

O exemplo a seguir mostra como acionar uma função Lambda quando um objeto é adicionado a um bucket do 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));
```

Políticas de remoção

Recursos que mantêm dados persistentes, como bancos de dados, buckets do Amazon S3 e registros do Amazon ECR, têm uma política de remoção. A política de remoção indica se objetos persistentes devem ser excluídos quando a AWS CDK pilha que os contém for destruída. Os valores que especificam a política de remoção estão disponíveis por meio da `RemovalPolicy` enumeração no módulo. AWS CDK `core`

> **Note**
>
> Recursos além daqueles que armazenam dados de forma persistente também podem ser usados para uma finalidade diferente. `removalPolicy` Por exemplo, uma versão da função Lambda usa um `removalPolicy` atributo para determinar se uma determinada versão é retida quando uma nova versão é implantada. Eles têm significados e padrões diferentes em comparação com a política de remoção em um bucket do Amazon S3 ou tabela do DynamoDB.

Valor	sentido
`RemovalPolicy.RETER`	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.DESTRUIR`	The resource will be destroyed along with the stack.

AWS CloudFormation não remove buckets do Amazon S3 que contêm arquivos, mesmo que sua política de remoção esteja definida como. `DESTROY` Tentar fazer isso é um AWS CloudFormation erro. Para AWS CDK excluir todos os arquivos do bucket antes de destruí-lo, defina a `autoDeleteObjects` propriedade do bucket como`true`.

Veja a seguir um exemplo de criação de um bucket do Amazon S3 com `RemovalPolicy` of `DESTROY` e `autoDeleteOjbects` definido como. `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
    });
}
```

Você também pode aplicar uma política de remoção diretamente ao AWS CloudFormation recurso subjacente por meio do `applyRemovalPolicy()` método. Esse método está disponível em alguns recursos com estado que não têm uma `removalPolicy` propriedade nas propriedades do recurso L2. Os exemplos incluem:

- AWS CloudFormation pilhas
- Grupos de usuários do Amazon Cognito
- Instâncias do banco de dados Amazon DocumentDB
- Volumes do Amazon EC2
- OpenSearch Domínios do Amazon Service
- Sistemas de arquivos Amazon FSx
- Filas do 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**
>
> O “ AWS CDK s” `RemovalPolicy` se traduz em AWS CloudFormation“`DeletionPolicy`s”. No entanto, o padrão AWS CDK é reter os dados, o que é o oposto do AWS CloudFormation padrão.

Identificadores

Ao criar AWS Cloud Development Kit (AWS CDK) aplicativos, você usará vários tipos de identificadores e nomes. Para usá-los de AWS CDK forma eficaz e evitar erros, é importante entender os tipos de identificadores.

Os identificadores devem ser exclusivos dentro do escopo em que foram criados; eles não precisam ser globalmente exclusivos em seu AWS CDK aplicativo.

Se você tentar criar um identificador com o mesmo valor dentro do mesmo escopo, isso AWS CDK gerará uma exceção.

Tópicos

IDs de construção

O identificador mais comum,`id`, é o identificador passado como segundo argumento ao instanciar um objeto de construção. Esse identificador, como todos os identificadores, só precisa ser exclusivo dentro do escopo em que foi criado, que é o primeiro argumento ao instanciar um objeto de construção.

> **Note**
>
> O `id` de uma pilha também é o identificador que você usa para se referir a ela nothe section called “AWS CDK Kit de ferramentas”.

Vejamos um exemplo em que temos duas construções com o identificador `MyBucket` em nosso aplicativo. O primeiro é definido no escopo da pilha com o identificador`Stack1`. O segundo é definido no escopo de uma pilha com o identificador`Stack2`. Como eles são definidos em escopos diferentes, isso não causa nenhum conflito e eles podem coexistir no mesmo aplicativo sem problemas.

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");
    }
}
```

Caminhos

As construções em um AWS CDK aplicativo formam uma hierarquia enraizada na classe. App Nós nos referimos à coleção de IDs de uma determinada construção, sua construção principal, sua avô e assim por diante até a raiz da árvore de construção, como um caminho.

AWS CDK Normalmente, exibe caminhos em seus modelos como uma string. Os IDs dos níveis são separados por barras, começando no nó imediatamente abaixo da App instância raiz, que geralmente é uma pilha. Por exemplo, os caminhos dos dois recursos de bucket do Amazon S3 no exemplo de código anterior são e. `Stack1/MyBucket` `Stack2/MyBucket`

Você pode acessar o caminho de qualquer construção programaticamente, conforme mostrado no exemplo a seguir. Isso segue o caminho de `myConstruct` (ou`my_construct`, como escreveriam os desenvolvedores do Python). Como as IDs devem ser exclusivas dentro do escopo em que são criadas, seus caminhos são sempre exclusivos em um AWS CDK aplicativo.

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;
```

IDs exclusivos

AWS CloudFormation exige que todos os IDs lógicos em um modelo sejam exclusivos. Por isso, eles AWS CDK devem ser capazes de gerar um identificador exclusivo para cada construção em um aplicativo. Os recursos têm caminhos que são globalmente exclusivos (os nomes de todos os escopos da pilha até um recurso específico). Portanto, AWS CDK gera os identificadores exclusivos necessários concatenando os elementos do caminho e adicionando um hash de 8 dígitos. (O hash é necessário para distinguir caminhos distintos, como `A/B/C` e`A/BC`, que resultariam no mesmo AWS CloudFormation identificador. AWS CloudFormation os identificadores são alfanuméricos e não podem conter barras ou outros caracteres separadores.) O AWS CDK chama essa string de ID exclusivo da construção.

Em geral, seu AWS CDK aplicativo não precisa saber sobre IDs exclusivos. No entanto, você pode acessar o ID exclusivo de qualquer construção programaticamente, conforme mostrado no exemplo a seguir.

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);
```

O endereço é outro tipo de identificador exclusivo que distingue de forma exclusiva os recursos do CDK. Derivado do hash SHA-1 do caminho, não é legível por humanos. No entanto, seu comprimento constante e relativamente curto (sempre 42 caracteres hexadecimais) o torna útil em situações em que o ID exclusivo “tradicional” pode ser muito longo. Algumas construções podem usar o endereço no AWS CloudFormation modelo sintetizado em vez do ID exclusivo. Novamente, seu aplicativo geralmente não precisa saber sobre os endereços de suas construções, mas você pode recuperar o endereço de uma construção da seguinte maneira.

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;
```

IDs lógicos

Os IDs exclusivos servem como identificadores lógicos (ou nomes lógicos) dos recursos nos AWS CloudFormation modelos gerados para construções que representam AWS recursos.

Por exemplo, o bucket do Amazon S3 no exemplo anterior, criado dentro, `Stack2` resulta em um `AWS::S3::Bucket` recurso. O ID lógico do recurso está `Stack2MyBucket4DD88B4F` no AWS CloudFormation modelo resultante. (Para obter detalhes sobre como esse identificador é gerado, consultethe section called “IDs exclusivos”.)

Estabilidade lógica de ID

Evite alterar a ID lógica de um recurso após sua criação. AWS CloudFormation identifica os recursos por meio de sua ID lógica. Portanto, se você alterar a ID lógica de um recurso, AWS CloudFormation cria um novo recurso com a nova ID lógica e exclui a existente. Dependendo do tipo de recurso, isso pode causar interrupção do serviço, perda de dados ou ambas.

Tokens

Os tokens representam valores que só podem ser resolvidos posteriormente no ciclo de vida do aplicativo. Por exemplo, o nome de um bucket do Amazon Simple Storage Service (Amazon S3) que você define em seu aplicativo CDK só é alocado quando o modelo é sintetizado. AWS CloudFormation Se você imprimir o `bucket.bucketName` atributo, que é uma string, verá que ele contém algo parecido com o seguinte:

```
${TOKEN[Bucket.Name.1234]}
```

É assim que ele AWS CDK codifica um token cujo valor ainda não é conhecido no momento da construção, mas estará disponível posteriormente. Eles AWS CDK chamam esses espaços reservados de tokens. Nesse caso, é um token codificado como uma string.

Você pode passar essa string como se fosse o nome do bucket. No exemplo a seguir, o nome do bucket é especificado como uma variável de ambiente para uma AWS Lambda função.

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
    }
});
```

Quando o AWS CloudFormation modelo é finalmente sintetizado, o token é renderizado como intrínseco. AWS CloudFormation `{ "Ref": "MyBucket" }` No momento da implantação, AWS CloudFormation substitui esse intrínseco pelo nome real do bucket que foi criado.

Tópicos

Tokens e codificações de tokens

Tokens são objetos que implementam a interface IResolvable, que contém um único método. `resolve` O AWS CDK chama esse método durante a síntese para produzir o valor final para o AWS CloudFormation modelo. Os tokens participam do processo de síntese para produzir valores arbitrários de qualquer tipo.

> **Note**
>
> Você raramente trabalhará diretamente com a `IResolvable` interface. Provavelmente, você verá apenas versões codificadas por string dos tokens.

Outras funções normalmente aceitam apenas argumentos de tipos básicos, como `string` ou`number`. Para usar tokens nesses casos, você pode codificá-los em um dos três tipos usando métodos estáticos na classe cdk.Token.

- `Token.asString`para gerar uma codificação de string (ou chamar `.toString()` o objeto token)
- `Token.asList`para gerar uma codificação de lista
- `Token.asNumber`para gerar uma codificação numérica

Eles pegam um valor arbitrário, que pode ser um`IResolvable`, e os codificam em um valor primitivo do tipo indicado.

> **Important**
>
> Como qualquer um dos tipos anteriores pode ser potencialmente um token codificado, tenha cuidado ao analisar ou tentar ler seu conteúdo. Por exemplo, se você tentar analisar uma string para extrair um valor dela, e a string for um token codificado, sua análise falhará. Da mesma forma, se você tentar consultar o comprimento de uma matriz ou realizar operações matemáticas com um número, primeiro verifique se eles não são tokens codificados.

Para verificar se um valor tem um token não resolvido, chame o método `Token.isUnresolved` (Python`is_unresolved`:).

O exemplo a seguir confirma que um valor de string, que pode ser um token, não tem mais do que 10 caracteres.

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");
```

Se o nome for um token, a validação não será executada e um erro ainda poderá ocorrer em um estágio posterior do ciclo de vida, como durante a implantação.

> **Note**
>
> Você pode usar codificações de token para escapar do sistema de tipos. Por exemplo, você pode codificar em string um token que produz um valor numérico no momento da síntese. Se você usar essas funções, é sua responsabilidade garantir que seu modelo seja resolvido para um estado utilizável após a síntese.

Tokens codificados por string

Os tokens codificados por string têm a seguinte aparência.

```
${TOKEN[Bucket.Name.1234]}
```

Eles podem ser transmitidos como cadeias de caracteres regulares e podem ser concatenados, conforme mostrado no exemplo a seguir.

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";
```

Você também pode usar a interpolação de strings, se sua linguagem for compatível, conforme mostrado no exemplo a seguir.

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";
```

Evite manipular a string de outras formas. Por exemplo, pegar uma substring de uma string provavelmente quebrará o token da string.

Tokens codificados em lista

Os tokens codificados em lista têm a seguinte aparência:

```
["#{TOKEN[Stack.NotificationArns.1234]}"]
```

A única coisa segura a fazer com essas listas é passá-las diretamente para outras construções. Os tokens no formato de lista de strings não podem ser concatenados, nem um elemento pode ser

retirado do token. A única maneira segura de manipulá-los é usando funções AWS CloudFormation intrínsecas como fn.Select.

Tokens codificados por números

Os tokens codificados por números são um conjunto de pequenos números negativos de ponto flutuante que se parecem com os seguintes.

```
-1.8881545897087626e+289
```

Assim como acontece com os tokens da lista, você não pode modificar o valor do número, pois isso provavelmente quebrará o token numérico. A única operação permitida é passar o valor para outra construção.

Valores preguiçosos

Além de representar valores de tempo de implantação, como AWS CloudFormation parâmetros, os tokens também são comumente usados para representar valores preguiçosos de tempo de síntese. Esses são valores para os quais o valor final será determinado antes da conclusão da síntese, mas não no ponto em que o valor é construído. Use tokens para passar uma string literal ou um valor numérico para outra construção, enquanto o valor real no momento da síntese pode depender de algum cálculo que ainda não foi feito.

Você pode criar tokens representando valores preguiçosos de tempo de sintetização usando métodos estáticos na `Lazy` classe, como lazy.String e lazy.Number. Esses métodos aceitam um objeto cuja `produce` propriedade é uma função que aceita um argumento de contexto e retorna o valor final quando chamada.

O exemplo a seguir cria um grupo de Auto Scaling cuja capacidade é determinada após sua criação.

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;
```

Convertendo para JSON

Às vezes, você deseja produzir uma string JSON de dados arbitrários e talvez não saiba se os dados contêm tokens. Para codificar adequadamente qualquer estrutura de dados em JSON,

independentemente de ela conter tokens, use a pilha de métodos. toJsonString, conforme mostrado no exemplo a seguir.

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
});
```

Parâmetros

Os parâmetros são valores personalizados fornecidos no momento da implantação. Os parâmetros são uma característica do AWS CloudFormation. Como AWS Cloud Development Kit (AWS CDK) sintetiza AWS CloudFormation modelos, ele também oferece suporte para parâmetros de tempo de implantação.

Tópicos

Sobre os parâmetros

Usando o AWS CDK, você pode definir parâmetros, que podem então ser usados nas propriedades das construções que você cria. Você também pode implantar pilhas que contenham parâmetros.

Ao implantar o AWS CloudFormation modelo usando o AWS CDK Toolkit, você fornece os valores dos parâmetros na linha de comando. Se você implantar o modelo por meio do AWS CloudFormation console, você será solicitado a fornecer os valores dos parâmetros.

Em geral, não recomendamos o uso de AWS CloudFormation parâmetros com AWS CDK o. As formas usuais de passar valores para AWS CDK aplicativos são valores de contexto e variáveis de ambiente. Como não estão disponíveis no momento da síntese, os valores dos parâmetros não podem ser facilmente usados para controle de fluxo e outras finalidades em seu aplicativo CDK.

> **Note**
>
> Para controlar o fluxo com parâmetros, você pode usar `CfnCondition`construções, embora isso seja estranho em comparação com declarações nativas. `if`

O uso de parâmetros exige que você esteja ciente de como o código que você está escrevendo se comporta no momento da implantação e também no momento da síntese. Isso torna mais difícil entender e raciocinar sobre sua AWS CDK inscrição, em muitos casos com poucos benefícios.

Geralmente, é melhor que seu aplicativo CDK aceite as informações necessárias de uma forma bem definida e as use diretamente para declarar construções em seu aplicativo CDK. Um AWS CloudFormation modelo ideal AWS CDK gerado é concreto, sem valores a serem especificados no momento da implantação.

No entanto, existem casos de uso para AWS CloudFormation os quais os parâmetros são exclusivamente adequados. Se você tiver equipes separadas definindo e implantando a infraestrutura, por exemplo, você pode usar parâmetros para tornar os modelos gerados mais amplamente úteis. Além disso, como o AWS CDK suporta AWS CloudFormation parâmetros, você pode usá-lo AWS CDK com AWS serviços que usam AWS CloudFormation modelos (como Service Catalog). Esses AWS serviços usam parâmetros para configurar o modelo que está sendo implantado.

Definindo parâmetros

Use a `CfnParameter`classe para definir um parâmetro. Você deve especificar pelo menos um tipo e uma descrição para a maioria dos parâmetros, embora ambos sejam tecnicamente opcionais. A descrição aparece quando o usuário é solicitado a inserir o valor do parâmetro no AWS CloudFormation console. Para obter mais informações sobre os tipos disponíveis, consulte Tipos.

> **Note**
>
> Você pode definir parâmetros em qualquer escopo. No entanto, recomendamos definir parâmetros no nível da pilha para que seu ID lógico não mude quando você refatorar seu código.

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"
});
```

Uso de parâmetros

Uma `CfnParameter` instância expõe seu valor ao seu AWS CDK aplicativo por meio de um token. Como todos os tokens, o token do parâmetro é resolvido no momento da síntese. Mas isso se resume a uma referência ao parâmetro definido no AWS CloudFormation modelo (que será resolvido no momento da implantação), em vez de um valor concreto.

Você pode recuperar o token como uma instância da `Token` classe ou em string, lista de strings ou codificação numérica. Sua escolha depende do tipo de valor exigido pela classe ou método com o qual você deseja usar o parâmetro.

TypeScript

Property	kind of value
`valor`	`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
`valor`	`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

Python

Property	kind of value
`valor`	`Token` class instance
`valor_como_lista`	The token represented as a string list
`valor_como_número`	The token represented as a number
`valor_como_string`	The token represented as a string

Java

Property	kind of value
`getValue ()`	`Token` class instance
`getValueAsLista ()`	The token represented as a string list
`getValueAsNúmero ()`	The token represented as a number
`getValueAsCadeia de caracteres ()`	The token represented as a string

C#

Property	kind of value
`Valor`	`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

Por exemplo, para usar um parâmetro em uma `Bucket` definição:

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
};
```

Implantação com parâmetros

Um modelo gerado contendo parâmetros pode ser implantado da maneira usual por meio do AWS CloudFormation console. Você será solicitado a fornecer os valores de cada parâmetro.

O AWS CDK Toolkit (ferramenta de linha de `cdk` comando) também suporta a especificação de parâmetros na implantação. Você os fornece na linha de comando, seguindo o `--parameters` sinalizador. Você pode implantar uma pilha que usa o `uploadBucketName` parâmetro, como no exemplo a seguir.

```
cdk deploy MyStack --parameters uploadBucketName=uploadbucket
```

Para definir vários parâmetros, use vários `--parameters` sinalizadores.

```
cdk deploy MyStack --parameters uploadBucketName=upbucket --parameters
 downloadBucketName=downbucket
```

Se você estiver implantando várias pilhas, poderá especificar um valor diferente de cada parâmetro para cada pilha. Para fazer isso, prefixe o nome do parâmetro com o nome da pilha e dois pontos.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=uploadbucket --
parameters YourStack:uploadBucketName=upbucket
```

Por padrão, o AWS CDK retém os valores dos parâmetros de implantações anteriores e os usa em implantações subsequentes, caso não sejam especificados explicitamente. Use o `--no-previous-parameters` sinalizador para exigir que todos os parâmetros sejam especificados.

Tags

As tags são elementos informativos de valor-chave que você pode adicionar às construções em seu aplicativo. AWS CDK Uma tag aplicada a uma determinada construção também se aplica a todos os seus filhos marcáveis. As tags são incluídas no AWS CloudFormation modelo sintetizado a partir do seu aplicativo e aplicadas aos AWS recursos que ele implanta. Você pode usar tags para identificar e categorizar recursos para as seguintes finalidades:

- Simplificando o gerenciamento
- Alocação de custos
- Controle de acesso
- Quaisquer outros propósitos que você conceba

> **Tip**
>
> Para obter mais informações sobre como você pode usar tags com seus AWS recursos, consulte Melhores práticas para marcar AWS recursos no AWS whitepaper.

Tópicos

Usar tags

A `Tags` classe inclui o método estático `of()`, por meio do qual você pode adicionar ou remover tags da construção especificada.

- `Tags.of(SCOPE).add()` aplica uma nova tag à construção fornecida e a todos os seus filhos.
- `Tags.of(SCOPE).remove()` remove uma tag de uma determinada construção e de qualquer um de seus filhos, incluindo tags que uma construção secundária possa ter aplicado a si mesma.

> **Note**
>
> A marcação é implementada usando the section called “Aspectos”. Aspectos são uma forma de aplicar uma operação (como marcação) a todas as construções em um determinado escopo.

O exemplo a seguir aplica a chave de tag com o valor do valor a uma construção.

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");
```

O exemplo a seguir exclui a chave da tag de uma construção.

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");
```

Se você estiver usando `Stage` construções, aplique a tag no `Stage` nível ou abaixo. As etiquetas não são aplicadas além `Stage` dos limites.

Prioridades de tags

O AWS CDK aplica e remove as tags recursivamente. Se houver conflitos, a operação de marcação com a prioridade mais alta vence. (As prioridades são definidas usando a `priority` propriedade opcional.) Se as prioridades de duas operações forem as mesmas, a operação de marcação mais próxima da parte inferior da árvore de construção vence. Por padrão, a aplicação de uma tag tem uma prioridade de 100 (exceto as tags adicionadas diretamente a um AWS CloudFormation recurso, que tem uma prioridade de 50). A prioridade padrão para remover uma tag é 200.

O seguinte aplica uma tag com prioridade de 300 a uma construção.

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 });
```

Propriedades opcionais

As tags `properties`oferecem suporte para ajustar a forma como as tags são aplicadas ou removidas dos recursos. Todas as propriedades são opcionais.

`applyToLaunchedInstances`(Python:) `apply_to_launched_instances`

Disponível somente para add (). Por padrão, as tags são aplicadas às instâncias executadas em um grupo de Auto Scaling. Defina essa propriedade como false para ignorar instâncias iniciadas em um grupo de Auto Scaling.

`includeResourceTypes`/`excludeResourceTypes`(Python:`include_resource_types`/) `exclude_resource_types`

Use-os para manipular tags somente em um subconjunto de recursos, com base nos tipos de AWS CloudFormation recursos. Por padrão, a operação é aplicada a todos os recursos na subárvore de construção, mas isso pode ser alterado incluindo ou excluindo determinados tipos de recursos. Excluir tem precedência sobre incluir, se ambas forem especificadas.

`priority`

Use isso para definir a prioridade dessa operação em relação a outras `Tags.remove()` operações `Tags.add()` e. Valores mais altos têm precedência sobre valores mais baixos. O padrão é 100 para operações de adição (50 para tags aplicadas diretamente aos AWS CloudFormation recursos) e 200 para operações de remoção.

O exemplo a seguir aplica a tag tagname com o valor do valor e a prioridade 100 aos recursos do tipo AWS::Xxx::Yyyna construção. Ela não aplica a tag a instâncias lançadas em um grupo do Amazon EC2 Auto Scaling ou a recursos desse tipo. AWS::Xxx::Zzz (Esses são espaços reservados para dois tipos de AWS CloudFormation recursos arbitrários, mas diferentes.)

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
});
```

O exemplo a seguir remove a tag tagname com prioridade 200 dos recursos do tipo AWS::Xxx::Yyyna construção, mas não dos recursos do tipo 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
});
```

Exemplo

O exemplo a seguir adiciona a chave de tag StackTypecom valor TheBesta qualquer recurso criado dentro do `Stack` nome`MarketingSystem`. Em seguida, ele o remove novamente de todos os recursos, exceto das sub-redes VPC do Amazon EC2. O resultado é que somente as sub-redes têm a tag aplicada.

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"]
});
```

O código a seguir obtém o mesmo resultado. Considere qual abordagem (inclusão ou exclusão) torna sua intenção mais clara.

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"]
});
```

Marcação de construções únicas

`Tags.of(scope).add(key, value)`é a forma padrão de adicionar tags às construções no AWS CDK. Seu comportamento de caminhar em árvores, que marca recursivamente todos os recursos marcáveis sob um determinado escopo, é quase sempre o que você deseja. Às vezes, no entanto, você precisa marcar uma construção (ou construções) específica e arbitrária.

Um desses casos envolve a aplicação de tags cujo valor é derivado de alguma propriedade da construção que está sendo marcada. A abordagem de marcação padrão aplica recursivamente a mesma chave e valor a todos os recursos correspondentes no escopo. No entanto, aqui o valor pode ser diferente para cada construção marcada.

As tags são implementadas usando aspectos, e o CDK chama o `visit()` método da tag para cada construção sob o escopo que você especificou usando`Tags.of(scope)`. Podemos chamar `Tag.visit()` diretamente para aplicar uma tag a uma única construção.

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);
```

Você pode marcar todas as construções em um escopo, mas deixar que os valores das tags sejam derivados das propriedades de cada construção. Para fazer isso, escreva um aspecto e aplique a tag no `visit()` método do aspecto, conforme mostrado no exemplo anterior. Em seguida, adicione o aspecto ao escopo desejado usando`Aspects.of(scope).add(aspect)`.

O exemplo a seguir aplica uma tag a cada recurso em uma pilha contendo o caminho do recurso.

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**
>
> A lógica da marcação condicional, incluindo prioridades, tipos de recursos e assim por diante, é incorporada à `Tag` classe. Você pode usar esses recursos ao aplicar tags a recursos arbitrários; a tag não é aplicada se as condições não forem atendidas. Além disso, a `Tag` classe só marca recursos marcáveis, então você não precisa testar se uma construção pode ser marcada antes de aplicar uma tag.

Ativos

Os ativos são arquivos, diretórios ou imagens do Docker locais que podem ser agrupados em AWS CDK bibliotecas e aplicativos. Por exemplo, um ativo pode ser um diretório que contém o código do manipulador de uma AWS Lambda função. Os ativos podem representar qualquer artefato que o aplicativo precise para operar.

O vídeo tutorial a seguir fornece uma visão geral abrangente dos ativos do CDK e explica como você pode usá-los em sua infraestrutura como código (IaC).

Explicação dos ativos da CDK

Você adiciona ativos por meio de APIs que são expostas por AWS construções específicas. Por exemplo, quando você define uma construção Lambda.function, a propriedade code permite que você passe um ativo (diretório). `Function`usa ativos para agrupar o conteúdo do diretório e usá-lo para o código da função. Da mesma forma, ecs. ContainerImage.fromAsset usa uma imagem do Docker criada a partir de um diretório local ao definir uma definição de tarefa do Amazon ECS.

Ativos em detalhes

Quando você se refere a um ativo em seu aplicativo, o conjunto de nuvem que é sintetizado a partir do seu aplicativo inclui informações de metadados com instruções para a CLI. AWS CDK As instruções incluem onde encontrar o ativo no disco local e que tipo de agrupamento executar com base no tipo de ativo, como um diretório para compactar (zip) ou uma imagem do Docker para criar.

Isso AWS CDK gera um hash de origem para ativos. Isso pode ser usado no momento da construção para determinar se o conteúdo de um ativo foi alterado.

Por padrão, o AWS CDK cria uma cópia do ativo no diretório de montagem da nuvem, cujo padrão é `cdk.out`, sob o hash de origem. Dessa forma, a montagem da nuvem é independente, portanto, se for transferida para um host diferente para implantação, ela ainda poderá ser implantada. Para mais detalhes, consulte the section called “Montagens em nuvem”.

Quando AWS CDK implanta um aplicativo que faz referência a ativos (diretamente pelo código do aplicativo ou por meio de uma biblioteca), a AWS CDK CLI primeiro prepara e publica os ativos em um bucket do Amazon S3 ou repositório do Amazon ECR. (O bucket ou repositório do S3 é criado durante a inicialização.) Somente então os recursos definidos na pilha são implantados.

Esta seção descreve as APIs de baixo nível disponíveis na estrutura.

Tipos de ativo

O AWS CDK suporta os seguintes tipos de ativos:

Ativos do Amazon S3

Esses são arquivos e diretórios locais que são AWS CDK carregados para o Amazon S3.

Imagem de docker

Essas são imagens do Docker que são AWS CDK enviadas para o Amazon ECR.

Esses tipos de ativos são explicados nas seções a seguir.

Ativos do Amazon S3

Você pode definir arquivos e diretórios locais como ativos e os AWS CDK pacotes e enviá-los para o Amazon S3 por meio do módulo aws-s3-assets.

O exemplo a seguir define um ativo de diretório local e um ativo de arquivo.

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")),
})
```

Na maioria dos casos, você não precisa usar diretamente as APIs no `aws-s3-assets` módulo. Módulos que oferecem suporte a ativos`aws-lambda`, como, têm métodos convenientes para que você possa usar ativos. Para funções Lambda, o método estático fromAsset () permite que você especifique um diretório ou um arquivo.zip no sistema de arquivos local.

Exemplo de função Lambda

Um caso de uso comum é criar funções Lambda com o código do manipulador como um ativo do Amazon S3.

O exemplo a seguir usa um ativo do Amazon S3 para definir um manipulador do Python no diretório local. `handler` Ele também cria uma função Lambda com o ativo do diretório local como propriedade. `code` A seguir está o código Python para o manipulador.

```
def lambda_handler(event, context):
```

```
  message = 'Hello World!'
  return {
    'message': message
  }
```

O código do AWS CDK aplicativo principal deve ter a seguinte aparência.

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
}
```

O `Function` método usa ativos para agrupar o conteúdo do diretório e usá-lo para o código da função.

> **Tip**
>
> Os `.jar` arquivos Java são arquivos ZIP com uma extensão diferente. Eles são enviados no estado em que se encontram para o Amazon S3, mas quando implantados como uma função Lambda, os arquivos que eles contêm são extraídos, o que talvez você não queira. Para evitar isso, coloque o `.jar` arquivo em um diretório e especifique esse diretório como o ativo.

Exemplo de atributos de tempo de implantação

Os tipos de ativos do Amazon S3 também expõem atributos de tempo de implantação que podem ser referenciados em bibliotecas e aplicativos. AWS CDK O comando AWS CDK CLI cdk synth exibe as propriedades do ativo como AWS CloudFormation parâmetros.

O exemplo a seguir usa atributos de tempo de implantação para passar a localização de um ativo de imagem para uma função Lambda como variáveis de ambiente. (O tipo de arquivo não importa; a imagem PNG usada aqui é apenas um exemplo.)

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(),
  },
})
```

Permissões

Se você usa ativos do Amazon S3 diretamente por meio do módulo aws-s3-assets, funções, usuários ou grupos do IAM e precisa ler ativos em tempo de execução, conceda a esses ativos permissões do IAM por meio do método asset.grantRead.

O exemplo a seguir concede a um grupo do IAM permissões de leitura em um ativo de arquivo.

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)
```

Ativos de imagem do Docker

O AWS CDK suporta o agrupamento de imagens locais do Docker como ativos por meio do aws-ecr-assetsmódulo.

O exemplo a seguir define uma imagem do Docker que é criada localmente e enviada para o Amazon ECR. As imagens são criadas a partir de um diretório de contexto local do Docker (com um Dockerfile) e enviadas para o Amazon ECR pela AWS CDK CLI ou pelo pipeline de CI/CD do seu aplicativo. As imagens podem ser referenciadas naturalmente em seu AWS CDK aplicativo.

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")),
})
```

O `my-image` diretório deve incluir um Dockerfile. A AWS CDK CLI cria uma imagem do Docker`my-image`, a envia para um repositório Amazon ECR e especifica o nome do repositório como um parâmetro para sua pilha. AWS CloudFormation Os tipos de ativos de imagem do Docker expõem atributos de tempo de implantação que podem ser referenciados em AWS CDK bibliotecas e aplicativos. O comando AWS CDK CLI cdk synth exibe as propriedades do ativo como AWS CloudFormation parâmetros.

Exemplo de definição de tarefa do Amazon ECS

Um caso de uso comum é criar um Amazon ECS TaskDefinitionpara executar contêineres Docker. O exemplo a seguir especifica a localização de um ativo de imagem do Docker que ele AWS CDK cria localmente e envia para o 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),
})
```

Exemplo de atributos de tempo de implantação

O exemplo a seguir mostra como usar os atributos de tempo de implantação `repository` e `imageUri` criar uma definição de tarefa do Amazon ECS com o tipo de AWS Fargate execução. Observe que a pesquisa do repositório Amazon ECR exige a tag da imagem, não seu URI, então nós a recortamos do final do URI do ativo.

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()),
})
```

Exemplo de construção de argumentos

Você pode fornecer argumentos de construção personalizados para a etapa de construção do Docker por meio da opção de propriedade `buildArgs` (Python`build_args`:) quando a AWS CDK CLI cria a imagem durante a implantação.

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"),
  },
})
```

Permissões

Se você usa um módulo que oferece suporte a ativos de imagem do Docker, como aws-ecs, AWS CDK ele gerencia as permissões para você quando você usa ativos diretamente ou por meio dele. ContainerImage fromEcrRepository(Python:`from_ecr_repository`). Se você usar ativos de imagem do Docker diretamente, certifique-se de que o principal consumidor tenha permissão para extrair a imagem.

Na maioria dos casos, você deve usar o método Asset.repository.grantPull (Python:. `grant_pull` Isso modifica a política do IAM do principal para permitir que ele extraia imagens desse repositório. Se o diretor que está extraindo a imagem não estiver na mesma conta ou se for um AWS serviço que não assume uma função na sua conta (como AWS CodeBuild), você deve conceder permissões de extração na política de recursos e não na política do diretor. Use o asset.repository. addToResourceMétodo de política (Python:`add_to_resource_policy`) para conceder as permissões principais apropriadas.

AWS CloudFormation metadados de recursos

> **Note**
>
> Esta seção é relevante somente para autores de construções. Em determinadas situações, as ferramentas precisam saber que um determinado recurso CFN está usando um ativo local. Por exemplo, você pode usar a AWS SAM CLI para invocar funções do Lambda localmente para fins de depuração. Para mais detalhes, consulte the section called “AWS SAM integração”.

Para habilitar esses casos de uso, ferramentas externas consultam um conjunto de entradas de metadados sobre AWS CloudFormation recursos:

- `aws:asset:path`— Aponta para o caminho local do ativo.
- `aws:asset:property`— O nome da propriedade do recurso em que o ativo é usado.

Usando essas duas entradas de metadados, as ferramentas podem identificar que os ativos são usados por um determinado recurso e permitir experiências locais avançadas.

Para adicionar essas entradas de metadados a um recurso, use o método `asset.addResourceMetadata` (Python`add_resource_metadata`:).

Permissões

A AWS Construct Library usa alguns idiomas comuns e amplamente implementados para gerenciar o acesso e as permissões. O módulo IAM fornece as ferramentas de que você precisa para usar esses idiomas.

AWS CDK usa AWS CloudFormation para implantar mudanças. Cada implantação envolve um ator (um desenvolvedor ou um sistema automatizado) que inicia uma AWS CloudFormation implantação. Ao fazer isso, o ator assumirá uma ou mais identidades do IAM (usuário ou funções) e, opcionalmente, passará uma função para. AWS CloudFormation

Se você usa AWS IAM Identity Center para se autenticar como usuário, o provedor de login único fornece credenciais de sessão de curta duração que autorizam você a atuar como uma função predefinida do IAM. Para saber como o AWS CDK obtém AWS credenciais da autenticação do IAM

Identity Center, consulte Compreender a autenticação do IAM Identity Center no Guia de referência de AWS SDKs e ferramentas.

Entidades principais

Um principal do IAM é uma AWS entidade autenticada que representa um usuário, serviço ou aplicativo que pode chamar AWS APIs. A AWS Construct Library suporta a especificação de diretores de várias maneiras flexíveis para permitir que eles acessem seus AWS recursos.

Em contextos de segurança, o termo “principal” se refere especificamente a entidades autenticadas, como usuários. Objetos como grupos e funções não representam usuários (e outras entidades autenticadas), mas os identificam indiretamente com o objetivo de conceder permissões.

Por exemplo, se você criar um grupo do IAM, poderá conceder ao grupo (e, portanto, a seus membros) acesso de gravação a uma tabela do Amazon RDS. No entanto, o grupo em si não é um principal porque não representa uma única entidade (além disso, você não pode fazer login em um grupo).

Na biblioteca IAM do CDK, classes que identificam direta ou indiretamente os principais implementam a `IPrincipal`interface, permitindo que esses objetos sejam usados de forma intercambiável nas políticas de acesso. No entanto, nem todos são diretores no sentido de segurança. Esses objetos incluem:

1. Recursos do IAM`Role`, como`User`, e `Group`
2. Diretores de serviço () `new iam.ServicePrincipal('service.amazonaws.com')`
3. Diretores federados () `new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`
4. Diretores de contas (`new iam.AccountPrincipal('0123456789012')`)
5. Princípios de usuário canônicos () `new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`
6. AWS Organizations diretores () `new iam.OrganizationPrincipal('org-id')`
7. Diretores arbitrários do ARN () `new iam.ArnPrincipal(res.arn)`
8. E `iam.CompositePrincipal(principal1, principal2, ...)` confiar em vários diretores

Concessões

Cada construção que representa um recurso que pode ser acessado, como um bucket do Amazon S3 ou uma tabela do Amazon DynamoDB, tem métodos que concedem acesso a outra entidade. Todos esses métodos têm nomes que começam com grant.

Por exemplo, os buckets do Amazon S3 têm os métodos e `grantRead` (`grantReadWrite`Python:`grant_read`,`grant_read_write`) para permitir o acesso de leitura e leitura/gravação, respectivamente, de uma entidade ao bucket. A entidade não precisa saber exatamente quais permissões do Amazon S3 IAM são necessárias para realizar essas operações.

O primeiro argumento de um método de concessão é sempre do tipo iGrantable. Essa interface representa entidades que podem receber permissões. Ou seja, ele representa recursos com funções, como os objetos do IAM `RoleUser`, `Group` e.

Outras entidades também podem receber permissões. Por exemplo, mais adiante neste tópico, mostraremos como conceder a um CodeBuild projeto acesso a um bucket do Amazon S3. Geralmente, a função associada é obtida por meio de uma `role` propriedade na entidade que está recebendo acesso.

Recursos que usam funções de execução, como`lambda.Function`, também são implementados`IGrantable`, para que você possa conceder acesso direto a eles em vez de conceder acesso à função deles. Por exemplo, se `bucket` for um bucket do Amazon S3 e `function` for uma função Lambda, o código a seguir concede à função acesso de leitura ao bucket.

TypeScript

```
bucket.grantRead(function);
```

JavaScript

```
bucket.grantRead(function);
```

Python

```
bucket.grant_read(function)
```

Java

```
bucket.grantRead(function);
```

C#

```
bucket.GrantRead(function);
```

Às vezes, as permissões precisam ser aplicadas enquanto sua pilha está sendo implantada. Um desses casos é quando você concede a um recurso AWS CloudFormation personalizado acesso a algum outro recurso. O recurso personalizado será invocado durante a implantação, portanto, ele deve ter as permissões especificadas no momento da implantação.

Outro caso é quando um serviço verifica se a função que você passa para ele tem as políticas corretas aplicadas. (Vários AWS serviços fazem isso para garantir que você não se esqueça de definir as políticas.) Nesses casos, a implantação pode falhar se as permissões forem aplicadas tarde demais.

Para forçar a aplicação das permissões da concessão antes da criação de outro recurso, você pode adicionar uma dependência da concessão em si, conforme mostrado aqui. Embora o valor de retorno dos métodos de concessão seja geralmente descartado, todo método de concessão na verdade retorna um `iam.Grant` objeto.

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);
```

Funções

O pacote do IAM contém uma `Role` construção que representa as funções do IAM. O código a seguir cria uma nova função, confiando no serviço 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
});
```

Você pode adicionar permissões a uma função chamando o `addToPolicy` método da função (Python:`add_to_policy`), passando um `PolicyStatement` que define a regra a ser adicionada. A declaração é adicionada à política padrão da função; se não tiver nenhuma, uma será criada.

O exemplo a seguir adiciona uma declaração de `Deny` política à função das ações `ec2:SomeAction` e `s3:AnotherAction` dos recursos `bucket` e `otherRole` (Python:`other_role`), sob a condição de que o serviço autorizado seja. 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"
        }
    }
}));
```

No exemplo anterior, criamos uma nova `PolicyStatement` linha com a chamada (`addToPolicy`Python:). `add_to_policy` Você também pode transmitir uma declaração de política existente ou uma que você tenha modificado. O PolicyStatementobjeto tem vários métodos para adicionar princípios, recursos, condições e ações.

Se você estiver usando uma construção que exige uma função para funcionar corretamente, você pode fazer o seguinte:

- Passe uma função existente ao instanciar o objeto de construção.
- Deixe que a construção crie uma nova função para você, confiando no diretor de serviço apropriado. O exemplo a seguir usa essa construção: um CodeBuild projeto.

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
});
```

Depois que o objeto é criado, a função (seja a função passada ou a padrão criada pela construção) fica disponível como propriedade`role`. No entanto, essa propriedade não está disponível em recursos externos. Portanto, essas construções têm um método `addToRolePolicy` (Python`add_to_role_policy`:).

O método não faz nada se a construção for um recurso externo e, caso contrário, chama o método `addToPolicy` (Python:`add_to_policy`) da `role` propriedade. Isso evita o trabalho de lidar explicitamente com o caso indefinido.

O exemplo a seguir demonstra:

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
}));
```

Políticas de recursos

Alguns recursos AWS, como buckets do Amazon S3 e funções do IAM, também têm uma política de recursos. Essas construções têm um `addToResourcePolicy` método (Python`add_to_resource_policy`:), que usa a `PolicyStatement` como argumento. Cada declaração de política adicionada a uma política de recursos deve especificar pelo menos um principal.

No exemplo a seguir, o bucket do Amazon S3 `bucket` concede uma função com a `s3:SomeAction` permissão para si mesmo.

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 }
}));
```

Usando objetos externos do IAM

Se você definiu um usuário, diretor, grupo ou função do IAM fora do seu AWS CDK aplicativo, você pode usar esse objeto do IAM em seu AWS CDK aplicativo. Para fazer isso, crie uma referência a ele usando seu ARN ou seu nome. (Use o nome para usuários, grupos e funções.) A referência retornada pode então ser usada para conceder permissões ou criar declarações de política, conforme explicado anteriormente.

- Para usuários, ligue `User.fromUserArn()` ou `User.fromUserName()`. `User.fromUserAttributes()` também está disponível, mas atualmente fornece a mesma funcionalidade que `User.fromUserArn()`.
- Para os principais, instancie um objeto. `ArnPrincipal`
- Para grupos, ligue `Group.fromGroupArn()` ou `Group.fromGroupName()`.
- Para funções, ligue para `Role.fromRoleArn()` ou `Role.fromRoleName()`.

As políticas (incluindo políticas gerenciadas) podem ser usadas de forma semelhante usando os métodos a seguir. Você pode usar referências a esses objetos em qualquer lugar em que uma política do IAM seja necessária.

- `Policy.fromPolicyName`
- `ManagedPolicy.fromManagedPolicyArn`
- `ManagedPolicy.fromManagedPolicyName`
- `ManagedPolicy.fromAwsManagedPolicyName`

> **Note**
>
> Como acontece com todas as referências a AWS recursos externos, você não pode modificar objetos externos do IAM em seu aplicativo CDK.

Contexto de runtime

Os valores de contexto são pares de chave-valor que podem ser associados a um aplicativo, pilha ou estrutura. Eles podem ser fornecidos ao seu aplicativo a partir de um arquivo (geralmente `cdk.json` ou `cdk.context.json` no diretório do projeto) ou na linha de comando.

O CDK Toolkit usa contexto para armazenar em cache os valores recuperados da sua AWS conta durante a síntese. Os valores incluem as zonas de disponibilidade em sua conta ou os IDs da Amazon Machine Image (AMI) atualmente disponíveis para instâncias do Amazon EC2. Como esses valores são fornecidos pela sua AWS conta, eles podem mudar entre as execuções do seu aplicativo CDK. Isso os torna uma fonte potencial de mudanças não intencionais. O comportamento de armazenamento em cache do CDK Toolkit “congela” esses valores para seu aplicativo CDK até que você decida aceitar os novos valores.

Imagine o cenário a seguir sem o cache de contexto. Digamos que você especificou “Amazon Linux mais recente” como a AMI para suas instâncias do Amazon EC2 e uma nova versão dessa AMI foi lançada. Então, na próxima vez que você implantasse sua pilha de CDK, suas instâncias já implantadas usariam a AMI desatualizada (“errada”) e precisariam ser atualizadas. A atualização resultaria na substituição de todas as suas instâncias existentes por novas, o que provavelmente seria inesperado e indesejado.

Em vez disso, o CDK registra as AMIs disponíveis da sua conta no `cdk.context.json` arquivo do seu projeto e usa o valor armazenado para futuras operações de síntese. Dessa forma, a lista de AMIs não é mais uma fonte potencial de mudança. Você também pode ter certeza de que suas pilhas sempre serão sintetizadas nos mesmos modelos. AWS CloudFormation

Nem todos os valores de contexto são valores armazenados em cache do seu AWS ambiente. the section called “Sinalizadores de destaque”também são valores de contexto. Você também pode criar seus próprios valores de contexto para uso por seus aplicativos ou construções.

As chaves de contexto são cadeias de caracteres. Os valores podem ser de qualquer tipo suportado pelo JSON: números, cadeias de caracteres, matrizes ou objetos.

> **Tip**
>
> Se suas construções criarem seus próprios valores de contexto, incorpore o nome do pacote da sua biblioteca em suas chaves para que não entrem em conflito com os valores de contexto de outros pacotes.

Muitos valores de contexto estão associados a um AWS ambiente específico, e um determinado aplicativo CDK pode ser implantado em mais de um ambiente. A chave para esses valores inclui a AWS conta e a região, para que os valores de diferentes ambientes não entrem em conflito.

A chave de contexto a seguir ilustra o formato usado pelo AWS CDK, incluindo a conta e a região.

```
availability-zones:account=123456789012:region=eu-central-1
```

> **Important**
>
> Os valores de contexto em cache são gerenciados pelo AWS CDK e suas construções, incluindo construções que você pode escrever. Não adicione nem altere valores de

contexto em cache editando arquivos manualmente. No entanto, pode ser útil revisar `cdk.context.json` ocasionalmente para ver quais valores estão sendo armazenados em cache. Os valores de contexto que não representam valores em cache devem ser armazenados sob a `context` chave de`cdk.json`. Dessa forma, eles não serão apagados quando os valores em cache forem limpos.

Fontes de valores de contexto

Os valores de contexto podem ser fornecidos ao seu AWS CDK aplicativo de seis maneiras diferentes:

- Automaticamente da AWS conta corrente.
- Através da --context opção para o cdk comando. (Esses valores são sempre cadeias de caracteres.)
- No `cdk.context.json` arquivo do projeto.
- Na `context` chave do `cdk.json` arquivo do projeto.
- Na `context` chave do seu `~/.cdk.json` arquivo.
- Em seu AWS CDK aplicativo usando o `construct.node.setContext()` método.

O arquivo do projeto `cdk.context.json` é onde os valores de contexto são armazenados em AWS CDK cache recuperados da sua AWS conta. Essa prática evita mudanças inesperadas em suas implantações quando, por exemplo, uma nova zona de disponibilidade é introduzida. O AWS CDK não grava dados de contexto em nenhum dos outros arquivos listados.

> **Important**
>
> Porque eles fazem parte do estado do seu aplicativo `cdk.json` e `cdk.context.json` devem estar comprometidos com o controle da fonte junto com o restante do código-fonte do seu aplicativo. Caso contrário, implantações em outros ambientes (por exemplo, um pipeline de CI) podem produzir resultados inconsistentes.

Os valores de contexto têm como escopo a construção que os criou; eles são visíveis para construções infantis, mas não para pais ou irmãos. Os valores de contexto definidos pelo AWS

CDK Toolkit (o cdk comando) podem ser definidos automaticamente, a partir de um arquivo ou da --context opção. Os valores de contexto dessas fontes são definidos implicitamente na App construção. Portanto, eles são visíveis para todas as construções em todas as pilhas do aplicativo.

Seu aplicativo pode ler um valor de contexto usando o `construct.node.tryGetContext` método. Se a entrada solicitada não for encontrada na construção atual ou em qualquer um de seus pais, o resultado seráundefined. (Como alternativa, o resultado pode ser equivalente à sua linguagem, como None em Python.)

Métodos de contexto

O AWS CDK suporta vários métodos de contexto que permitem que AWS CDK os aplicativos obtenham informações contextuais do AWS ambiente. Por exemplo, você pode obter uma lista de zonas de disponibilidade que estão disponíveis em uma determinada AWS conta e região usando o método stack.availabilityZones.

A seguir estão os métodos de contexto:

HostedZone. De Lookup

Obtém as zonas hospedadas em sua conta.

Zonas de disponibilidade do Stack.

Obtém as zonas de disponibilidade suportadas.

StringParameter.valueFromLookup

Obtém um valor do Amazon EC2 Systems Manager Parameter Store da região atual.

VPC.FromLookup

Obtém as Amazon Virtual Private Clouds existentes em suas contas.

LookupMachineImage

Pesquisa uma imagem de máquina para uso com uma instância NAT em uma Amazon Virtual Private Cloud.

Se um valor de contexto necessário não estiver disponível, o AWS CDK aplicativo notifica o CDK Toolkit de que as informações de contexto estão ausentes. Em seguida, a CLI consulta as informações na AWS conta atual e armazena as informações de contexto resultantes no arquivo.

`cdk.context.json` Em seguida, ele executa o AWS CDK aplicativo novamente com os valores de contexto.

Visualizando e gerenciando o contexto

Use o cdk context comando para visualizar e gerenciar as informações em seu `cdk.context.json` arquivo. Para ver essas informações, use o cdk context comando sem nenhuma opção. A saída deve ser algo parecido com o seguinte.

```
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.
```

Para remover um valor de contexto, executecdk context --reset, especificando a chave ou o número correspondente do valor. O exemplo a seguir remove o valor que corresponde à segunda chave no exemplo anterior. Esse valor representa a lista de zonas de disponibilidade na região da Europa (Irlanda).

```
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.
```

Portanto, se você quiser atualizar para a versão mais recente do Amazon Linux AMI, use o exemplo anterior para fazer uma atualização controlada do valor do contexto e redefini-lo. Em seguida, sintetize e implante seu aplicativo novamente.

```
cdk synth
```

Para limpar todos os valores de contexto armazenados para seu aplicativocdk context --clear, execute da seguinte maneira.

```
cdk context --clear
```

Somente os valores de contexto armazenados `cdk.context.json` podem ser redefinidos ou apagados. AWS CDK Isso não afeta outros valores de contexto. Portanto, para evitar que um valor de contexto seja redefinido usando esses comandos, você pode copiar o valor para`cdk.json`.

AWS CDK Sinalizador do kit de ferramentas **`--context`**

Use a opção `--context` (abreviada) `-c` para passar valores de contexto de tempo de execução para seu aplicativo CDK durante a síntese ou a implantação.

```
cdk synth --context key=value MyStack
```

Para especificar vários valores de contexto, repita a --context opção quantas vezes quiser, fornecendo um par de valores-chave a cada vez.

```
cdk synth --context key1=value1 --context key2=value2 MyStack
```

Ao sintetizar várias pilhas, os valores de contexto especificados são passados para todas as pilhas. Para fornecer valores de contexto diferentes para pilhas individuais, use chaves diferentes para os valores ou use vários cdk synth cdk deploy comandos.

Os valores de contexto passados da linha de comando são sempre cadeias de caracteres. Se um valor geralmente é de algum outro tipo, seu código deve estar preparado para converter ou analisar o valor. Você pode ter valores de contexto que não sejam de string fornecidos de outras formas (por exemplo, em`cdk.context.json`). Para garantir que esse tipo de valor funcione conforme o esperado, confirme se o valor é uma string antes de convertê-lo.

Exemplo

Veja a seguir um exemplo do uso de uma Amazon VPC existente usando AWS CDK contexto.

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()
        });
    }
```

```
}
```

Você pode usar cdk diff para ver os efeitos da transmissão de um valor de contexto na linha de comando:

```
cdk diff -c vpcid=vpc-0cb9c31031d0d3e22
```

```
Stack ExistsvpcStack
Outputs
[+] Output publicsubnets publicsubnets:
 {"Value":"subnet-06e0ea7dd302d3e8f,subnet-01fc0acfb58f3128f"}
```

Os valores de contexto resultantes podem ser visualizados conforme mostrado aqui.

```
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"
    ]
  }
}
```

Sinalizadores de destaque

AWS CDK Ele usa sinalizadores de recursos para permitir comportamentos potencialmente incorretos em uma versão. Os sinalizadores são armazenados como the section called “Contexto” valores em `cdk.json` (ou`~/.cdk.json`). Eles não são removidos pelos cdk context --clear comandos cdk context --reset ou.

Os sinalizadores de recursos estão desativados por padrão. Os projetos existentes que não especificam o sinalizador continuarão funcionando como antes nas AWS CDK versões posteriores. Novos projetos criados usando sinalizadores de cdk init inclusão que habilitam todos os recursos disponíveis na versão que criou o projeto. Edite `cdk.json` para desativar todos os sinalizadores para os quais você prefere o comportamento anterior. Você também pode adicionar sinalizadores para ativar novos comportamentos após a atualização do. AWS CDK

Uma lista de todos os sinalizadores de recursos atuais pode ser encontrada no AWS CDK GitHub repositório em. `FEATURE_FLAGS.md` Consulte `CHANGELOG` em uma determinada versão para obter uma descrição de quaisquer novos sinalizadores de recursos adicionados nessa versão.

Revertendo para o comportamento v1

No CDK v2, os padrões de alguns sinalizadores de recursos foram alterados em relação à v1. Você pode configurá-los novamente para reverter `false` para um comportamento AWS CDK v1 específico. Use o `cdk diff` comando para inspecionar as alterações em seu modelo sintetizado e ver se algum desses sinalizadores é necessário.

`@aws-cdk/core:newStyleStackSynthesis`

> Use o novo método de síntese de pilha, que pressupõe recursos de bootstrap com nomes conhecidos. Requer uma inicialização moderna, mas, por sua vez, permite CI/CD via CDK Pipelines e implantações entre contas prontas para uso.

`@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId`

Se seu aplicativo usa várias chaves de API do Amazon API Gateway e as associa aos planos de uso.

`@aws-cdk/aws-rds:lowercaseDbIdentifier`

Se seu aplicativo usa uma instância de banco de dados ou clusters de banco de dados do Amazon RDS e especifica explicitamente o identificador para eles.

`@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021`

Se seu aplicativo usa a política de segurança TLS_V1_2_2019 com distribuições. Amazon CloudFront O CDK v2 usa a política de segurança TLSv1.2_2021 por padrão.

`@aws-cdk/core:stackRelativeExports`

Se seu aplicativo usa várias pilhas e você se refere aos recursos de uma pilha em outra, isso determina se o caminho absoluto ou relativo é usado para criar AWS CloudFormation exportações.

`@aws-cdk/aws-lambda:recognizeVersionProps`

Se definido como`false`, o CDK inclui metadados ao detectar se uma função Lambda foi alterada. Isso pode causar falhas na implantação quando somente os metadados são alterados, já que versões duplicadas não são permitidas. Não há necessidade de reverter esse sinalizador se você tiver feito pelo menos uma alteração em todas as funções Lambda em seu aplicativo.

A sintaxe para reverter esses sinalizadores `cdk.json` é mostrada aqui.

```
{
  "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
  }
}
```

Aspectos

Aspectos são uma forma de aplicar uma operação a todas as construções em um determinado escopo. O aspecto pode modificar as construções, por exemplo, adicionando tags. Ou pode verificar algo sobre o estado das construções, como garantir que todos os buckets estejam criptografados.

Para aplicar um aspecto a uma construção e a todas as construções no mesmo escopo, chame `Aspects.of(SCOPE).add()`com um novo aspecto, conforme mostrado no exemplo a seguir.

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 Ele usa aspectos para marcar recursos, mas a estrutura também pode ser usada para outros fins. Por exemplo, você pode usá-lo para validar ou alterar os AWS CloudFormation recursos definidos para você por construções de nível superior.

Aspectos em detalhes

Os aspectos empregam o padrão do visitante. Um aspecto é uma classe que implementa a interface a seguir.

TypeScript

```
interface IAspect {
   visit(node: IConstruct): void;}
```

JavaScript

JavaScript não tem interfaces como recurso de linguagem. Portanto, um aspecto é simplesmente uma instância de uma classe com um `visit` método que aceita o nó a ser operado.

Python

O Python não tem interfaces como recurso de linguagem. Portanto, um aspecto é simplesmente uma instância de uma classe com um `visit` método que aceita o nó a ser operado.

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)
}
```

Quando você chama`Aspects.of(`*SCOPE*`).add(...)`, a construção adiciona o aspecto a uma lista interna de aspectos. Você pode obter a lista com`Aspects.of(`*SCOPE*`)`.

Durante a fase de preparação, ele AWS CDK chama o `visit` método do objeto para a construção e cada um de seus filhos em ordem de cima para baixo.

O `visit` método é livre para alterar qualquer coisa na construção. Em linguagens de tipagem forte, converta a construção recebida em um tipo mais específico antes de acessar propriedades ou métodos específicos da construção.

Os aspectos não se propagam além dos limites da `Stage` construção, porque `Stages` são independentes e imutáveis após a definição. Aplique aspectos na `Stage` construção em si (ou inferior) se quiser que eles visitem construções dentro da`Stage`.

Exemplo

O exemplo a seguir confirma que todos os buckets criados na pilha têm o versionamento ativado. O aspecto adiciona uma anotação de erro às construções que falham na validação. Isso resulta na falha da synth operação e impede a implantação do conjunto de nuvem resultante.

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());
```

Começando com o AWS CDK

Comece com o AWS Cloud Development Kit (AWS CDK) instalando AWS CDK CLI e criando seu primeiro aplicativo CDK.

Tópicos

Pré-requisitos

Recursos recomendados

Antes de começar a usar o AWS CDK, recomendamos uma compreensão básica do seguinte:

- Uma introdução ao AWS CDK. Para saber mais, consulte O que é o AWS CDK?
- Conceitos fundamentais por trás do AWS CDK. Para saber mais, consulte AWS CDK conceitos.
- O Serviços da AWS que você deseja gerenciar com AWS CDK o.
- AWS Identity and Access Management. Para obter mais informações, consulte O que é IAM? e o que é o IAM Identity Center?
- AWS CloudFormation já que AWS CDK utiliza o AWS CloudFormation serviço para provisionar recursos criados no CDK. Para saber mais, consulte O que é o AWS CloudFormation?
- A linguagem de programação compatível que você planeja usar com AWS CDK o.

Prepare seu ambiente local

Todos os AWS CDK desenvolvedores, independentemente do seu idioma preferido, precisam da versão Node.js14.15.0 ou posterior. Todas as linguagens de programação suportadas usam o mesmo back-end, que é executado emNode.js. Recomendamos uma versão com suporte ativo de longo prazo. Sua organização pode ter uma recomendação diferente.

> **Important**
>
> As versões 13.0.0 a 13.6.0 do Node.js não são compatíveis com o AWS CDK devido a problemas de compatibilidade com suas dependências.

Outros pré-requisitos dependem da linguagem na qual você desenvolve AWS CDK aplicativos e são os seguintes.

TypeScript

- TypeScript 3.8 ou posterior () `npm -g install typescript`

JavaScript

Sem requisitos adicionais

Python

- Python 3.7 ou posterior, incluindo e `pip` `virtualenv`

Java

- Java Development Kit (JDK) 8 (também conhecido como 1.8) ou posterior
- Apache Maven 3.5 ou posterior

O Java IDE é recomendado (usamos o Eclipse em alguns exemplos neste guia). O IDE deve ser capaz de importar projetos Maven. Verifique se o seu projeto está configurado para usar o Java 1.8. Defina a variável de ambiente JAVA_HOME para o caminho em que você instalou o JDK.

C#

.NET Core 3.1 ou posterior, ou.NET 6.0 ou posterior.

Recomenda-se o Visual Studio 2019 (qualquer edição) ou o Visual Studio Code.

Go

Vá para a versão 1.1.8 ou posterior.

Para obter informações mais detalhadas, consulte a seção Pré-requisitos do seu idioma:

- the section called “Em TypeScript”
- the section called “Em JavaScript”
- the section called “Em Python”
- the section called “Em Java”
- the section called “Em C#”
- the section called “Em Go”

> **Suspensão de uso de linguagem de terceiros**
>
> Cada versão de idioma só é suportada até o fim da vida útil e está sujeita a alterações mediante aviso prévio. EOL

Etapa 1: criar um Conta da AWS

Se você for iniciante AWS, deverá se inscrever em um Conta da AWS e criar um usuário administrativo. Para obter mais informações, consulte Como configurar com o IAM no Guia do usuário do IAM.

Ao interagir com AWS, você especifica suas credenciais de AWS segurança para verificar quem você é e se tem permissão para acessar os recursos que está solicitando. AWS usa as credenciais de segurança para autenticar e autorizar suas solicitações. Para saber mais, consulte as credenciais AWS de segurança no Guia do usuário do IAM.

Etapa 2: Configurar o acesso programático

Ao desenvolver com o AWS CDK em seu ambiente local, você confiará no AWS CDK CLI para interagir Serviços da AWS e gerenciar seus AWS recursos. Para usar o AWS CDK CLI, você deve configurar o acesso programático. Para saber mais sobre as diferentes formas de configurar o acesso programático, consulte Autenticação e acesso no Guia de referência de AWS SDKs e ferramentas.

Para novos usuários que não recebem um método de autenticação do empregador, recomendamos o uso AWS IAM Identity Center. Esse método inclui instalar o AWS Command Line Interface (AWS CLI) e usá-lo para configuração e login no portal de AWS acesso. Para configurar o acesso

programático usando o IAM Identity Center, consulte a autenticação do IAM Identity Center no Guia de referência de AWS SDKs e ferramentas. Após a conclusão, seu ambiente deve conter os seguintes elementos:

- O AWS CLI, que você usa para iniciar uma sessão do portal de AWS acesso antes de executar seu aplicativo.
- Um AWS`config`arquivo compartilhado com um `[default]` perfil com um conjunto de valores de configuração que podem ser referenciados a AWS CDK partir do. Para encontrar a localização desse arquivo, consulte Localização dos arquivos compartilhados no Guia de referência de ferramentas e SDKs da AWS .
- O arquivo `config` compartilhado define a configuração da `region`. Isso define como padrão Região da AWS os AWS CDK usos das AWS solicitações.
- O AWS CDK usa a configuração do provedor de token SSO do perfil para adquirir credenciais antes de enviar solicitações para. AWS O `sso_role_name` valor, que é uma função do IAM conectada a um conjunto de permissões do IAM Identity Center, deve permitir o acesso ao Serviços da AWS usado em seu aplicativo.

  O arquivo de exemplo `config` a seguir mostra um perfil padrão configurado com a configuração do provedor de token de SSO. A configuração `sso_session` do perfil se refere à seção chamada `sso-session`. A `sso-session` seção contém configurações para iniciar uma sessão do portal de AWS acesso.

```
[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
```

Iniciar uma sessão do portal de AWS acesso

Antes de acessar Serviços da AWS, você precisa de uma sessão ativa do portal de AWS acesso para usar AWS CDK a autenticação do IAM Identity Center para resolver as credenciais.

Dependendo da duração da sessão configurada, seu acesso acabará expirando e AWS CDK ocorrerá um erro de autenticação. Execute o seguinte comando no AWS CLI para entrar no portal de AWS acesso.

```
aws sso login
```

Se a configuração do seu provedor de token SSO estiver usando um perfil nomeado em vez do perfil padrão, o comando seráaws sso login --profile *NAME*. Especifique também esse perfil ao emitir cdk comandos usando a --profile opção ou a variável de AWS_PROFILE ambiente.

Para testar se você já tem uma sessão ativa, execute o AWS CLI comando a seguir.

```
aws sts get-caller-identity
```

A resposta a esse comando deve relatar a conta do Centro de Identidade do IAM e o conjunto de permissões configurados no arquivo compartilhado config.

> **Note**
>
> Se você já tiver uma sessão ativa do portal de AWS acesso e executá-laaws sso login, não será necessário fornecer credenciais.
> O processo de login pode solicitar que você permita o AWS CLI acesso aos seus dados. Como o AWS CLI é construído sobre o SDK para Python, as mensagens de permissão podem conter variações do botocore nome.

Etapa 3: instalar o AWS CDKCLI

Instale o AWS CDK CLI globalmente usando o seguinte comando Node do Package Manager.

```
npm install -g aws-cdk
```

> **Note**
>
> Se você receber um erro de permissão e tiver acesso de administrador em seu sistema, tentesudo npm install -g aws-cdk.

Execute o comando a seguir para verificar se a instalação foi bem-sucedida. O AWS CDK CLI deve gerar o número da versão:

```
cdk --version
```

Se você receber uma mensagem de erro, tente desinstalar o AWS CDK CLI executando o seguinte:

```
npm uninstall -g aws-cdk
```

Em seguida, repita as etapas para reinstalar o. AWS CDK CLI

Se você ainda receber um erro, exclua a `node-modules` pasta do projeto atual e também da `node-modules` pasta global. Para localizar essa pasta, execute`npm config get prefix`.

Eles AWS CDK CLI obterão credenciais de segurança de fontes que você configurou nas etapas anteriores.

> **Note**
>
> O CDK Toolkit v2 funciona com projetos CDK v1 existentes. No entanto, ele não pode inicializar novos projetos CDK v1. Veja the section called “Novos pré-requisitos” se você precisa ser capaz de fazer isso.

Etapa 4: inicialize seu ambiente

Cada AWS ambiente no qual você planeja implantar recursos deve ser inicializado.

Para inicializar, execute o seguinte:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

> **Tip**
>
> Se você não tiver o número AWS da sua conta em mãos, você pode obtê-lo no AWS Management Console. Ou, se você tiver o AWS CLI instalado, o comando a seguir exibirá as informações padrão da sua conta, incluindo o número da conta.
>
> ```
> aws sts get-caller-identity
> ```

Se você criou perfis nomeados em sua AWS configuração local, você pode usar a `--profile` opção para exibir as informações da conta para um perfil específico. O exemplo a seguir mostra como exibir as informações da conta para o perfil prod.

```
aws sts get-caller-identity --profile prod
```

Para exibir a região padrão, use`aws configure get`.

```
aws configure get region
aws configure get region --profile prod
```

AWS CDK Ferramentas opcionais

AWS Toolkit for Visual Studio CodeÉ um plug-in de código aberto para o Visual Studio Code que ajuda você a criar, depurar e implantar aplicativos no. AWS O kit de ferramentas fornece uma experiência integrada para o desenvolvimento de AWS CDK aplicativos. Ele inclui o recurso AWS CDK Explorer para listar seus AWS CDK projetos e navegar pelos vários componentes do aplicativo CDK. Instale o plug-in e saiba mais sobre como usar o AWS CDK Explorer.

Próximas etapas

Agora que você instalou o AWS CDK CLI, use-o para criar seu primeiro AWS CDK aplicativo.

Para saber mais sobre como usar o AWS CDK em sua linguagem de programação preferida, consulteTrabalhando com o AWS CDK em linguagens de programação suportadas.

AWS CDK É um projeto de código aberto. Para contribuir, consulte Contribuindo para AWS Cloud Development Kit (AWS CDK) o.

Saiba mais

Para saber mais sobre o AWS CDK, consulte o seguinte:

- Workshop CDK — Workshop prático aprofundado.
- Referência de API — Explore as construções disponíveis para o Serviços da AWS que você usará.

- Construct Hub — Encontre construções da comunidade CDK.
- AWS CDK exemplos — Explore exemplos de código de AWS CDK projetos.

Seu primeiro AWS CDK aplicativo

Comece a usar o AWS Cloud Development Kit (AWS CDK) criando seu primeiro aplicativo CDK.

Antes de iniciar este tutorial, recomendamos que você conclua o seguinte:

- Consulte O que é o AWS CDK? para obter uma introdução ao AWS CDK.
- Veja AWS CDK conceitos para aprender os principais conceitos do AWS CDK.
- Siga os pré-requisitos e as etapas de AWS CDK configuração em. Começando com o AWS CDK

Tópicos

Sobre este tutorial

Neste tutorial, você criará e implantará um AWS CDK aplicativo simples. Esse aplicativo contém uma pilha com um único recurso de bucket do Amazon Simple Storage Service (Amazon S3). Por meio deste tutorial, você aprenderá o seguinte:

- A estrutura de um AWS CDK projeto.
- Como criar um AWS CDK aplicativo.

- Como usar a AWS Construct Library para definir aplicativos, pilhas e AWS recursos.
- Como usar o CDK CLI para sintetizar, diferenciar, implantar e excluir seu aplicativo CDK.
- Como modificar e reimplantar seu aplicativo CDK para atualizar seus recursos implantados.

O fluxo de trabalho de AWS CDK desenvolvimento padrão consiste nas seguintes etapas:

1. Crie seu AWS CDK aplicativo - Aqui, você usará um modelo fornecido pelo AWS CDK CLI.
2. Defina suas pilhas e recursos — Use construções para definir suas pilhas e AWS recursos em seu aplicativo.
3. Crie seu aplicativo — Essa etapa é opcional. O executa AWS CDK CLI automaticamente essa etapa, se necessário. É recomendável realizar essa etapa para identificar erros de sintaxe e tipo.
4. Sintetize suas pilhas — Essa etapa cria um AWS CloudFormation modelo para cada pilha em seu aplicativo. Essa etapa é útil para identificar erros lógicos nos AWS recursos definidos.
5. Implante seu aplicativo — implante em seu AWS ambiente usando AWS CloudFormation para provisionar seus recursos. Durante a implantação, você identificará quaisquer problemas de permissão com seu aplicativo.

Por meio de um fluxo de trabalho típico, você voltará e repetirá as etapas anteriores para modificar ou depurar seu aplicativo.

Recomendamos que você use o controle de versão para seus AWS CDK projetos.

Etapa 1: criar o aplicativo

Um aplicativo CDK deve estar em seu próprio diretório, com suas próprias dependências de módulos locais. Em sua máquina de desenvolvimento, crie um novo diretório. Veja a seguir um exemplo que cria um novo `hello-cdk` diretório:

```
$ mkdir hello-cdk
$ cd hello-cdk
```

> **Important**
>
> Certifique-se de nomear o diretório do seu projeto`hello-cdk`, exatamente como mostrado aqui. O modelo do AWS CDK projeto usa o nome do diretório para nomear coisas no código gerado. Se você usar um nome diferente, o código deste tutorial não funcionará.

Em seguida, no seu novo diretório, inicialize o aplicativo usando o cdk init comando. Especifique o app modelo e sua linguagem de programação preferida com a `--language` opção. Veja um exemplo a seguir:

TypeScript

```
cdk init app --language typescript
```

JavaScript

```
cdk init app --language javascript
```

Python

```
cdk init app --language python
```

Depois que o aplicativo for criado, insira também os dois comandos a seguir. Eles ativam o ambiente virtual Python do aplicativo e instalam as dependências AWS CDK principais.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
cdk init app --language java
```

Se você estiver usando um IDE, agora você pode abrir ou importar o projeto. No Eclipse, por exemplo, escolha Arquivo > Importar > Maven > Projetos Maven existentes. Certifique-se de que as configurações do projeto estejam definidas para usar o Java 8 (1.8).

C#

```
cdk init app --language csharp
```

Se você estiver usando o Visual Studio, abra o arquivo da solução no `src` diretório.

Go

```
cdk init app --language go
```

Depois que o aplicativo for criado, insira também o comando a seguir para instalar os módulos do AWS Construct Library que o aplicativo exige.

```
go get
```

O cdk init comando cria vários arquivos e pastas dentro do `hello-cdk` diretório para ajudar você a organizar o código-fonte do seu AWS CDK aplicativo. Coletivamente, isso é chamado de seu AWS CDK projeto. Reserve um momento para explorar o projeto CDK.

Se você Git instalou, cada projeto que você cria usando também cdk init é inicializado como um Git repositório.

Etapa 2: criar o aplicativo

Na maioria dos ambientes de programação, você cria ou compila o código depois de fazer alterações. Isso não é necessário com o, AWS CDK pois o CDK CLI executará essa etapa automaticamente. No entanto, você ainda pode criar manualmente quando quiser detectar erros de sintaxe e tipo. Veja um exemplo a seguir:

TypeScript

```
npm run build
```

JavaScript

Nenhuma etapa de construção é necessária.

Python

Nenhuma etapa de construção é necessária.

Java

```
mvn compile -q
```

Ou pressione Control-B no Eclipse (outros IDEs Java podem variar)

C#

```
dotnet build src
```

Ou pressione F6 no Visual Studio

Go

```
go build
```

Etapa 3: liste as pilhas no aplicativo

Verifique se seu aplicativo foi criado corretamente listando as pilhas em seu aplicativo. Execute o seguinte:

```
cdk ls
```

A saída deve ser exibida`HelloCdkStack`. Se você não ver essa saída, verifique se você está no diretório de trabalho correto do seu projeto e tente novamente. Se você ainda não vê sua pilha, repita the section called “Etapa 1: criar o aplicativo” e tente novamente.

Etapa 4: Adicionar um bucket do Amazon S3

Neste momento, seu aplicativo CDK contém uma única pilha. Em seguida, você definirá um recurso de bucket do Amazon Simple Storage Service (Amazon S3) dentro da sua pilha. Para fazer isso, você importará e usará a construção `Bucket` L2 da Construct Library. AWS

Modifique seu aplicativo CDK importando a `Bucket` construção e definindo seu recurso de bucket do Amazon S3. Veja um exemplo a seguir:

TypeScript

Em `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

Em `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

Em `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

Em `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#

Em `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

Em `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
}
```

Vamos dar uma olhada mais de perto na `Bucket` construção. Como todas as construções, a `Bucket` classe usa três parâmetros:

- escopo — Define a `Stack` classe como a mãe da `Bucket` construção. Todas as construções que definem AWS recursos são criadas dentro do escopo de uma pilha. Você pode definir construções dentro de construções, criando uma hierarquia (árvore). Aqui, e na maioria dos casos, o escopo é `this` (`self`emPython).
- Id — O ID lógico do `Bucket` dentro do seu AWS CDK aplicativo. Esse ID, mais um hash baseado na localização do bucket na pilha, identifica exclusivamente o bucket durante a implantação. AWS CDK Também faz referência a esse ID quando você atualiza a construção em seu aplicativo e reimplanta para atualizar o recurso implantado. Aqui, sua identificação lógica é`MyFirstBucket`. Os buckets também podem ter um nome, especificado com a `bucketName` propriedade. Isso é diferente do ID lógico.
- props — Um pacote de valores que define as propriedades do bucket. Aqui você definiu a `versioned` propriedade como`true`, que permite o controle de versão dos arquivos no bucket.

  Os adereços são representados de forma diferente nos idiomas suportados pelo AWS CDK.

  - In TypeScript and JavaScript, `props` é um único argumento e você passa um objeto contendo as propriedades desejadas.
  - Em Python, os adereços são passados como argumentos de palavras-chave.
  - Em Java, um Builder é fornecido para passar os adereços. Existem dois: um para `BucketProps` e um segundo para permitir que você `Bucket` construa a construção e seu objeto de adereços em uma única etapa. Esse código usa o último.
  - Em C#, você instancia um `BucketProps` objeto usando um inicializador de objetos e o passa como o terceiro parâmetro.

  Se os adereços de uma construção forem opcionais, você poderá omitir totalmente o `props` parâmetro.

Todas as construções usam esses mesmos três argumentos, então é fácil se manter orientado à medida que você aprende sobre novos. E, como era de se esperar, você pode subclassificar qualquer construção para estendê-la de acordo com suas necessidades ou se quiser alterar seus padrões.

Etapa 5: sintetizar um modelo AWS CloudFormation

Sintetize um AWS CloudFormation modelo para o aplicativo, da seguinte forma:

```
cdk synth
```

Se seu aplicativo contiver mais de uma pilha, você deverá especificar quais pilhas sintetizar. Como seu aplicativo contém uma única pilha, o CDK detecta CLI automaticamente a pilha para sintetizar.

Se você não executar`cdk synth`, o CDK CLI executará automaticamente essa etapa quando você implantar. No entanto, recomendamos que você execute essa etapa antes de cada implantação.

> **Tip**
>
> Se você receber um erro como`--app is required ...`, verifique o diretório no qual você está executando os CLI comandos do CDK. Você deve estar no diretório principal do seu aplicativo.

O `cdk synth` comando executa seu aplicativo. Isso cria um AWS CloudFormation modelo para cada pilha no seu aplicativo. O CDK CLI exibirá uma versão em formato YAML do seu modelo na linha de comando e salvará uma versão em formato JSON do seu modelo no diretório. `cdk.out` Veja a seguir um trecho da saída da linha de comando que mostra o bucket sendo definido no AWS CloudFormation modelo:

```
Resources:
  MyFirstBucketB8884501:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
    Metadata:...
```

> **Note**
>
> Cada modelo gerado contém um `AWS::CDK::Metadata` recurso por padrão. A AWS CDK equipe usa esses metadados para obter informações sobre o AWS CDK uso e encontrar

maneiras de melhorá-lo. Para obter detalhes, incluindo como optar por não receber relatórios de versão, consulteRelatórios de versão.

O modelo gerado pode ser implantado por meio do AWS CloudFormation console ou de qualquer ferramenta AWS CloudFormation de implantação. Você também pode usar o CDK CLI para implantar. Na próxima etapa, você usa o CDK CLI para implantar.

Etapa 6: implantar sua pilha

Para implantar sua pilha de CDK para AWS CloudFormation usar o CDKCLI, execute o seguinte:

```
cdk deploy
```

Important

Você deve executar uma única inicialização do seu AWS ambiente antes da implantação. Para obter instruções, consulte Bootstrap em seu ambiente.

Da mesma formacdk synth, você não precisa especificar a AWS CDK pilha, pois o aplicativo contém uma única pilha.

Se seu código tiver implicações de segurança, o CDK CLI produzirá um resumo. Você precisará confirmá-los para continuar com a implantação. O aplicativo deste tutorial não tem essas implicações.

Após a execuçãocdk deploy, o CDK CLI exibe informações de progresso à medida que sua pilha é implantada. Quando terminar, você pode acessar o AWS CloudFormation console para ver sua HelloCdkStack pilha. Você também pode acessar o console do Amazon S3 para visualizar seu MyFirstBucket recurso.

Parabéns! Você implantou sua primeira pilha usando o. AWS CDK Em seguida, você modificará seu aplicativo e o reimplantará para atualizar seu recurso.

Etapa 7: modifique seu aplicativo

Nesta etapa, você modificará seu bucket do Amazon S3 configurando-o para ser excluído automaticamente quando sua pilha for excluída. Essa modificação envolve a alteração da

`RemovalPolicy` propriedade do bucket. Você também configurará a `autoDeleteObjects` propriedade para configurar o CDK CLI para excluir objetos do bucket antes de destruí-lo. Por padrão, AWS CloudFormation não exclui buckets do Amazon S3 que contêm objetos.

Use o exemplo a seguir para modificar seu recurso:

TypeScript

Atualizar `lib/hello-cdk-stack.ts`.

```
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

JavaScript

Atualizar `lib/hello-cdk-stack.js`.

```
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

Python

Atualizar `hello_cdk/hello_cdk_stack.py`.

```
bucket = s3.Bucket(self, "MyFirstBucket",
    versioned=True,
    removal_policy=cdk.RemovalPolicy.DESTROY,
    auto_delete_objects=True)
```

Java

Atualizar `src/main/java/com/myorg/HelloCdkStack.java`.

```
Bucket.Builder.create(this, "MyFirstBucket")
        .versioned(true)
        .removalPolicy(RemovalPolicy.DESTROY)
        .autoDeleteObjects(true)
```

```
        .build();
```

C#

Atualizar `src/HelloCdk/HelloCdkStack.cs`.

```
new Bucket(this, "MyFirstBucket", new BucketProps
{
    Versioned = true,
    RemovalPolicy = RemovalPolicy.DESTROY,
    AutoDeleteObjects = true
});
```

Go

Atualizar `hello-cdk.go`.

```
   awss3.NewBucket(stack, jsii.String("MyFirstBucket"), &awss3.BucketProps{
  Versioned:         jsii.Bool(true),
  RemovalPolicy:     awscdk.RemovalPolicy_DESTROY,
  AutoDeleteObjects: jsii.Bool(true),
 })
```

Atualmente, suas alterações de código não fizeram nenhuma atualização direta em seu recurso de bucket do Amazon S3 implantado. Seu código define o estado desejado do seu recurso. Para modificar seu recurso implantado, você usará o CDK CLI para sintetizar o estado desejado em um novo modelo. AWS CloudFormation Em seguida, você implantará seu novo AWS CloudFormation modelo como um conjunto de alterações. Os conjuntos de alterações fazem somente as alterações necessárias para alcançar o novo estado desejado.

Para ver essas alterações, use o `cdk diff` comando. Execute o seguinte:

```
cdk diff
```

O CDK CLI consulta sua Conta da AWS conta para obter o AWS CloudFormation modelo mais recente para a pilha. `HelloCdkStack` Em seguida, ele compara o modelo mais recente com o modelo que acabou de sintetizar do seu aplicativo. A saída deve ser a seguinte.

```
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
```

Essa diferença tem quatro seções:

- Alterações na declaração do IAM e na política do IAM — Essas alterações de permissão existem porque você define a `AutoDeleteObjects` propriedade no seu bucket do Amazon S3. O recurso de exclusão automática usa um recurso personalizado para excluir os objetos no bucket antes que o próprio bucket seja excluído. Os objetos do IAM concedem ao código do recurso personalizado acesso ao bucket.
- Parâmetros — O AWS CDK usa essas entradas para localizar o ativo da AWS Lambda função para o recurso personalizado.
- Recursos — Os recursos novos e alterados nessa pilha. Podemos ver os objetos IAM mencionados anteriormente, o recurso personalizado e a função Lambda associada sendo adicionados. Também podemos ver que o bucket `DeletionPolicy` e os `UpdateReplacePolicy` atributos estão sendo atualizados. Isso permite que o bucket seja excluído junto com a pilha e substituído por um novo.

Você pode notar que especificamos `RemovalPolicy` em nosso AWS CDK aplicativo, mas obtivemos uma `DeletionPolicy` propriedade no AWS CloudFormation modelo resultante. Isso ocorre porque o AWS CDK usa um nome diferente para a propriedade. O AWS CDK padrão é reter o bucket quando a pilha é excluída, enquanto o AWS CloudFormation padrão é excluí-la. Para ter mais informações, consulte the section called “Políticas de remoção”.

Para ver seu novo AWS CloudFormation modelo, você pode executarcdk synth. Ao fazer algumas alterações em seu aplicativo CDK, seu novo AWS CloudFormation modelo agora inclui muitas linhas de código adicionais em comparação com o AWS CloudFormation modelo original.

Em seguida, implante seu aplicativo executando o seguinte:

```
cdk deploy
```

Ele AWS CDK informará você sobre as mudanças na política de segurança que já vimos no diff. Entre y para aprovar as alterações e implantar a pilha atualizada. O CDK CLI implantará sua pilha para fazer as alterações desejadas. Veja a seguir um exemplo de saída:

```
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
```

Etapa 8: Destruindo os recursos do aplicativo

Agora que você concluiu este tutorial, pode excluir a AWS CloudFormation pilha implantada e todos os recursos associados a ela. Essa é uma boa prática para minimizar custos desnecessários e manter seu ambiente limpo. Execute o seguinte:

```
cdk destroy
```

Entre y para aprovar as alterações e excluir sua pilha.

> **Note**
>
> Se você não alterasse o intervalo`RemovalPolicy`, a exclusão da pilha seria concluída com êxito, mas o intervalo ficaria órfão (não mais associado à pilha).

Próximas etapas

Parabéns! Você concluiu este tutorial e usou o AWS CDK para criar, modificar e excluir recursos no Nuvem AWS. Agora você está pronto para começar a usar AWS CDK o.

Para saber mais sobre como usar o AWS CDK em sua linguagem de programação preferida, consulteTrabalhando com o AWS CDK em linguagens de programação suportadas.

Para obter recursos adicionais, consulte o seguinte:

- Experimente o CDK Workshop para um tour mais aprofundado envolvendo um projeto mais complexo.
- Mergulhe mais fundo em conceitos como the section called “Ambientes do” the section called “Ativos” the section called “Permissões”the section called “Contexto”,,the section called “Parâmetros”,, the section called “Personalizando construções” e.
- Consulte a referência da API para começar a explorar as construções de CDK disponíveis para seus serviços favoritos AWS .
- Visite o Construct Hub para descobrir construções criadas por AWS e outros.
- Explore exemplos de uso do AWS CDK.

AWS CDK É um projeto de código aberto. Para contribuir, consulte Contribuindo para AWS Cloud Development Kit (AWS CDK) o.

Migrando da AWS CDK v1 para a v2 AWS CDK

A versão 2 do foi AWS Cloud Development Kit (AWS CDK) projetada para facilitar a escrita de infraestrutura como código em sua linguagem de programação preferida. Este tópico descreve as mudanças entre a v1 e a v2 do. AWS CDK

> **Tip**
>
> Para identificar pilhas implantadas com a AWS CDK v1, use o utilitário awscdk-v1-stack-finder.

As principais mudanças da AWS CDK v1 para o CDK v2 são as seguintes.

- AWS CDK A v2 consolida as partes estáveis da AWS Construct Library, incluindo a biblioteca principal, em um único pacote,. `aws-cdk-lib` Os desenvolvedores não precisam mais instalar pacotes adicionais para os AWS serviços individuais que usam. Essa abordagem de pacote único também significa que você não precisa sincronizar as versões dos vários pacotes da biblioteca CDK.

  As construções L1 (CFNxxxx), que representam os recursos exatos disponíveis AWS CloudFormation, são sempre consideradas estáveis e, portanto, estão incluídas. `aws-cdk-lib`
- Módulos experimentais, nos quais ainda estamos trabalhando com a comunidade para desenvolver novas construções L2 ou L3, não estão incluídos. `aws-cdk-lib` Em vez disso, eles são distribuídos como pacotes individuais. Os pacotes experimentais são nomeados com um `alpha` sufixo e um número de versão semântico. O número da versão semântica corresponde à primeira versão da AWS Construct Library com a qual eles são compatíveis, também com um `alpha` sufixo. As construções entram `aws-cdk-lib` depois de serem designadas como estáveis, permitindo que a Construct Library principal adote um controle de versão semântico estrito.

  A estabilidade é especificada no nível de serviço. Por exemplo, se começarmos a criar uma ou mais construções L2 para a Amazon AppFlow, que no momento em que escrevemos só construções L1, elas aparecerão primeiro em um módulo chamado. `@aws-cdk/aws-appflow-alpha` Em seguida, eles passam para `aws-cdk-lib` quando sentimos que as novas construções atendem às necessidades fundamentais dos clientes.

Depois que um módulo é designado como estável e incorporado`aws-cdk-lib`, novas APIs são adicionadas usando a convenção “BetAn” descrita no próximo bullet.

Uma nova versão de cada módulo experimental é lançada a cada versão do AWS CDK. Na maioria das vezes, no entanto, eles não precisam ser mantidos em sincronia. Você pode atualizar `aws-cdk-lib` o módulo experimental sempre que quiser. A exceção é que, quando dois ou mais módulos experimentais relacionados dependem um do outro, eles devem ser da mesma versão.

- Para módulos estáveis aos quais novas funcionalidades estão sendo adicionadas, novas APIs (sejam construções totalmente novas ou novos métodos ou propriedades em uma construção existente) recebem um `Beta1` sufixo enquanto o trabalho está em andamento. (Seguido por`Beta2`,`Beta3`, e assim por diante, quando mudanças significativas são necessárias.) Uma versão da API sem o sufixo é adicionada quando a API é designada estável. Todos os métodos, exceto o mais recente (beta ou final), serão então descontinuados.

  Por exemplo, se adicionarmos um novo método `grantPower()` a uma construção, ele aparecerá inicialmente como`grantPowerBeta1()`. Se forem necessárias alterações significativas (por exemplo, um novo parâmetro ou propriedade obrigatória), a próxima versão do método será nomeada `grantPowerBeta2()` e assim por diante. Quando o trabalho é concluído e a API é finalizada, o método `grantPower()` (sem sufixo) é adicionado e os métodos betaN são descontinuados.

  Todas as APIs beta permanecem na Construct Library até o próximo lançamento da versão principal (3.0), e suas assinaturas não serão alteradas. Você verá avisos de descontinuação se os usar. Portanto, você deve migrar para a versão final da API o mais rápido possível. No entanto, nenhuma AWS CDK versão 2.x futura quebrará seu aplicativo.
- A `Construct` classe foi extraída do AWS CDK para uma biblioteca separada, junto com os tipos relacionados. Isso é feito para apoiar os esforços de aplicar o Modelo de Programação Construct a outros domínios. Se você estiver escrevendo suas próprias construções ou usando APIs relacionadas, deverá declarar o `constructs` módulo como uma dependência e fazer pequenas alterações nas importações. Se você estiver usando recursos avançados, como conectar-se ao ciclo de vida do aplicativo CDK, talvez sejam necessárias mais mudanças. Para obter detalhes completos, consulte o RFC.
- Propriedades, métodos e tipos obsoletos na AWS CDK v1.x e em sua Construct Library foram completamente removidos da API CDK v2. Na maioria dos idiomas compatíveis, essas APIs produzem avisos na versão v1.x, então talvez você já tenha migrado para as APIs substitutas. Uma lista completa das APIs obsoletas no CDK v1.x está disponível em. GitHub

- O comportamento que foi controlado por sinalizadores de recursos na AWS CDK v1.x é ativado por padrão no CDK v2. Os sinalizadores de recursos anteriores não são mais necessários e, na maioria dos casos, não são suportados. Alguns ainda estão disponíveis para permitir que você volte ao comportamento do CDK v1 em circunstâncias muito específicas. Para ter mais informações, consulte the section called “Atualizando sinalizadores de recursos”.
- Com o CDK v2, os ambientes nos quais você implanta devem ser inicializados usando a pilha de bootstrap moderna. A pilha de bootstrap legada (o padrão na v1) não é mais suportada. Além disso, o CDK v2 requer uma nova versão da pilha moderna. Para atualizar seus ambientes existentes, reinicialize-os. Não é mais necessário definir nenhum sinalizador de recurso ou variável de ambiente para usar a pilha de bootstrap moderna.

> **Important**
>
> O modelo de bootstrap moderno concede efetivamente as permissões implícitas no `--cloudformation-execution-policies` para qualquer AWS conta na `--trust` lista. Por padrão, isso estende as permissões de leitura e gravação em qualquer recurso na conta inicializada. Certifique-se de configurar a pilha de inicialização com políticas e contas confiáveis com as quais você se sinta confortável.

Novos pré-requisitos

A maioria dos requisitos para a AWS CDK v2 é a mesma da AWS CDK v1.x. Os requisitos adicionais estão listados aqui.

- Para TypeScript desenvolvedores, a TypeScript versão 3.8 ou posterior é necessária.
- É necessária uma nova versão do CDK Toolkit para uso com o CDK v2. Agora que o CDK v2 está disponível ao público em geral, a v2 é a versão padrão ao instalar o CDK Toolkit. Ele é compatível com versões anteriores de projetos CDK v1, então você não precisa manter a versão anterior instalada, a menos que queira criar projetos CDK v1. Para atualizar, emitan`pm install -g aws-cdk`.

Atualização a partir da versão AWS CDK 2 Developer Preview

Se você estiver usando o CDK v2 Developer Preview, você tem dependências em seu projeto em uma versão Release Candidate do AWS CDK, como. `2.0.0-rc1` Atualize-os `2.0.0` e, em seguida, atualize os módulos instalados em seu projeto.

TypeScript

`npm install` ou `yarn install`

JavaScript

`npm install` ou `yarn install`

Python

```
python -m pip install -r requirements.txt
```

Java

```
mvn package
```

C#

```
dotnet restore
```

Go

```
go get
```

Depois de atualizar suas dependências, atualize `npm update -g aws-cdk` o CDK Toolkit para a versão de lançamento.

Migração da AWS CDK v1 para o CDK v2

Para migrar seu aplicativo para a AWS CDK v2, primeiro atualize os sinalizadores de recursos em. `cdk.json` Em seguida, atualize as dependências e importações do seu aplicativo conforme necessário para a linguagem de programação na qual ele está escrito.

Atualizando para uma v1 recente

Estamos vendo vários clientes atualizando de uma versão antiga da AWS CDK v1 para a versão mais recente da v2 em uma única etapa. Embora certamente seja possível fazer isso, você estaria atualizando ao longo de vários anos de mudanças (que, infelizmente, nem todas tiveram a mesma quantidade de testes de evolução que temos hoje), bem como atualizando várias versões com novos padrões e uma organização de código diferente.

Para uma experiência de atualização mais segura e para diagnosticar com mais facilidade as fontes de quaisquer alterações inesperadas, recomendamos que você separe essas duas etapas: primeiro atualize para a versão v1 mais recente e, em seguida, faça a mudança para a v2.

Atualizando sinalizadores de recursos

Remova os seguintes sinalizadores de recursos v1, `cdk.json` se eles existirem, pois todos eles estão ativos por padrão na AWS CDK v2. Se o efeito antigo deles for importante para sua infraestrutura, você precisará fazer alterações no código-fonte. Consulte a lista de bandeiras GitHub para obter mais informações.

- `@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`

Alguns sinalizadores de recursos da v1 podem ser configurados para reverter para `false` comportamentos específicos da AWS CDK v1; consulte the section called “Revertendo para o comportamento v1” ou acesse a lista para obter uma referência completa. GitHub

Para os dois tipos de sinalizadores, use o `cdk diff` comando para inspecionar as alterações em seu modelo sintetizado e ver se as alterações em algum desses sinalizadores afetam sua infraestrutura.

Compatibilidade com o CDK Toolkit

O CDK v2 requer a versão 2 ou posterior do CDK Toolkit. Esta versão é compatível com versões anteriores dos aplicativos CDK v1. Portanto, você pode usar uma única versão instalada globalmente do CDK Toolkit com todos os seus AWS CDK projetos, sejam eles v1 ou v2. Uma exceção é que o CDK Toolkit v2 só cria projetos CDK v2.

Se você precisar criar projetos CDK v1 e v2, não instale o CDK Toolkit v2 globalmente. (Remova-o se você já o tiver instalado:`npm remove -g aws-cdk`.) Para invocar o CDK Toolkit, use npx para executar a v1 ou v2 do CDK Toolkit conforme desejado.

```
npx aws-cdk@1.x init app --language typescript
npx aws-cdk@2.x init app --language typescript
```

Tip

Configure aliases de linha de comando para que você possa usar os cdk1 comandos cdk e para invocar a versão desejada do CDK Toolkit.

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 $*
```

Atualizando dependências e importações

Atualize as dependências do seu aplicativo e, em seguida, instale os novos pacotes. Por fim, atualize as importações em seu código.

TypeScript

Aplicações

Para aplicativos CDK, atualize da `package.json` seguinte forma. Remova as dependências dos módulos estáveis individuais no estilo v1 e estabeleça a versão mais baixa `aws-cdk-lib` que você precisa para seu aplicativo (2.0.0 aqui).

As construções experimentais são fornecidas em pacotes separados, com versões independentes, com nomes que terminam em `alpha` e um número de versão alfa. O número da versão alfa corresponde à primeira versão `aws-cdk-lib` com a qual eles são compatíveis. Aqui, fixamos a versão `aws-codestar` 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"
  }
}
```

Construa bibliotecas

Para construir bibliotecas, estabeleça a versão mais baixa `aws-cdk-lib` que você precisa para seu aplicativo (2.0.0 aqui) e atualize da `package.json` seguinte forma.

Observe que isso `aws-cdk-lib` aparece tanto como uma dependência de pares quanto como uma dependência de desenvolvimento.

```
{
  "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**
>
> Você deve fazer um grande aumento de versão no número da versão da sua biblioteca ao lançar uma biblioteca compatível com a v2, pois essa é uma alteração importante para

os consumidores de bibliotecas. Não é possível oferecer suporte ao CDK v1 e v2 com uma única biblioteca. Para continuar oferecendo suporte aos clientes que ainda usam a v1, você pode manter a versão anterior em paralelo ou criar um novo pacote para a v2. Depende de você por quanto tempo você deseja continuar oferecendo suporte aos clientes AWS CDK v1. Você pode seguir o exemplo do ciclo de vida do próprio CDK v1, que entrou em manutenção em 1º de junho de 2022 e end-of-life chegará em 1º de junho de 2023. Para obter detalhes completos, consulte a Política de AWS CDK manutenção.

Bibliotecas e aplicativos

Instale as novas dependências executando `npm install` ou `yarn install`.

Altere suas importações para importar `Construct` do novo `constructs` módulo, tipos principais, como `App` e `Stack` do nível superior `aws-cdk-lib`, e módulos estáveis da Construct Library para os serviços que você usa dos namespaces abaixo. `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

Atualize da `package.json` seguinte forma. Remova as dependências dos módulos estáveis individuais no estilo v1 e estabeleça a versão mais baixa `aws-cdk-lib` que você precisa para seu aplicativo (2.0.0 aqui).

As construções experimentais são fornecidas em pacotes separados, com versões independentes, com nomes que terminam em `alpha` e um número de versão alfa. O número da versão alfa corresponde à primeira versão `aws-cdk-lib` com a qual eles são compatíveis. Aqui, fixamos a versão `aws-codestar` 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"
  }
```

```
}
```

Instale as novas dependências executando `npm install` ou `yarn install`.

Altere as importações do seu aplicativo para fazer o seguinte:

- Importar `Construct` do novo `constructs` módulo
- Importe tipos principais, como `App` e `Stack`, do nível superior do `aws-cdk-lib`
- Importe módulos da AWS Construct Library de namespaces em `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

`requirements.txt`Atualize a `install_requires` definição da `setup.py` seguinte forma. Remova as dependências dos módulos estáveis individuais no estilo v1.

As construções experimentais são fornecidas em pacotes separados, com versões independentes, com nomes que terminam em `alpha` e um número de versão alfa. O número da versão alfa corresponde à primeira versão `aws-cdk-lib` com a qual eles são compatíveis. Aqui, fixamos a `aws-codestar` v2.0.0alpha1.

```
install_requires=[
     "aws-cdk-lib>=2.0.0",
     "constructs>=10.0.0",
     "aws-cdk.aws-codestar-alpha>=2.0.0alpha1",
     # ...
],
```

> **Tip**
>
> Desinstale todas as outras versões dos AWS CDK módulos já instalados no ambiente virtual do seu aplicativo usando `pip uninstall` o. Em seguida, instale as novas dependências com`python -m pip install -r requirements.txt`.

Altere as importações do seu aplicativo para fazer o seguinte:

- Importar `Construct` do novo `constructs` módulo
- Importe tipos principais, como `App` e`Stack`, do nível superior do `aws_cdk`
- Importe módulos da AWS Construct Library de namespaces em `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

Em`pom.xml`, remova todas as `software.amazon.awscdk` dependências dos módulos estáveis e substitua-as por dependências em `software.constructs` (para`Construct`) e. `software.amazon.awscdk`

As construções experimentais são fornecidas em pacotes separados, com versões independentes, com nomes que terminam em `alpha` e um número de versão alfa. O número da versão alfa corresponde à primeira versão `aws-cdk-lib` com a qual eles são compatíveis. Aqui, fixamos a versão `aws-codestar` 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>
```

Instale as novas dependências executando`mvn package`.

Altere seu código para fazer o seguinte:

- Importar `Construct` da nova `software.constructs` biblioteca
- Importe classes principais, como `Stack` eApp, de `software.amazon.awscdk`
- Importe construções de serviços de `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#

A maneira mais simples de atualizar as dependências de um aplicativo CDK em C# é editar o arquivo manualmente. `.csproj` Remova todas as referências de `Amazon.CDK.*` pacotes estáveis e substitua-as por referências aos `Constructs` pacotes `Amazon.CDK.Lib` e.

As construções experimentais são fornecidas em pacotes separados, com versões independentes, com nomes que terminam em `alpha` e um número de versão alfa. O número da versão alfa corresponde à primeira versão `aws-cdk-lib` com a qual eles são compatíveis. Aqui, fixamos a versão `aws-codestar` 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" />
```

Instale as novas dependências executando`dotnet restore`.

Altere as importações em seus arquivos de origem da seguinte maneira.

```
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

Problema go get para atualizar suas dependências para a versão mais recente e atualizar o `.mod` arquivo do seu projeto.

Testando seu aplicativo migrado antes da implantação

Antes de implantar suas pilhas, use `cdk diff` para verificar se há mudanças inesperadas nos recursos. Não são esperadas alterações nas IDs lógicas (causando a substituição de recursos).

As mudanças esperadas incluem, mas não estão limitadas a:

- Mudanças no `CDKMetadata` recurso.
- Hashes de ativos atualizados.
- Mudanças relacionadas ao novo estilo de síntese de pilhas. Aplica-se se seu aplicativo usou o sintetizador de pilha legado na v1. (O CDK v2 não é compatível com o sintetizador de pilha legado.)
- A adição de uma `CheckBootstrapVersion` regra.

Normalmente, mudanças inesperadas não são causadas pela atualização para a AWS CDK v2 em si. Normalmente, eles são o resultado de um comportamento obsoleto que foi alterado anteriormente por sinalizadores de recursos. Isso é um sintoma da atualização de uma versão do CDK anterior à 1.85.x. Você veria as mesmas mudanças na atualização para a versão v1.x mais recente. Normalmente, você pode resolver isso fazendo o seguinte:

1. Atualize seu aplicativo para a versão v1.x mais recente
2. Remover sinalizadores de recursos
3. Revise seu código conforme necessário
4. Implantar
5. Atualize para v2

> **Note**
>
> Se seu aplicativo atualizado não puder ser implantado após a atualização em dois estágios, relate o problema.

Quando você estiver pronto para implantar as pilhas em seu aplicativo, considere implantar uma cópia primeiro para poder testá-la. A maneira mais fácil de fazer isso é implantá-lo em uma região diferente. No entanto, você também pode alterar os IDs das suas pilhas. Após o teste, certifique-se de destruir a cópia de teste comcdk destroy.

Solução de problemas

TypeScript **`'from' expected`**ou **`';' expected`** erro nas importações

Atualize para TypeScript 3.8 ou posterior.

Execute 'cdk bootstrap'

Se você ver um erro como o seguinte:

```
#  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 A v2 requer uma pilha de bootstrap atualizada e, além disso, todas as implantações da v2 exigem recursos de bootstrap. (Com a v1, você pode implantar pilhas simples sem inicializar.) Para obter detalhes completos, consulte the section called “Bootstrapping”.

Encontrando pilhas v1

Ao migrar seu aplicativo CDK da v1 para a v2, talvez você queira identificar as AWS CloudFormation pilhas implantadas que foram criadas usando a v1. Para fazer isso, execute o seguinte comando:

```
npx awscdk-v1-stack-finder
```

Para obter detalhes de uso, consulte o README awscdk-v1-stack-finder.

Migre recursos e AWS CloudFormation modelos existentes para o AWS CDK

O recurso CDK Migrate está na versão prévia AWS CDK e está sujeito a alterações.

Use a interface de linha de AWS Cloud Development Kit (AWS CDK) comando (AWS CDK CLI) para migrar AWS recursos implantados, AWS CloudFormation pilhas implantadas e modelos locais para o. AWS CloudFormation AWS CDK

Tópicos

Como funciona a migração

Use o AWS CDK CLI `cdk migrate` comando para migrar das seguintes fontes:

- AWS Recursos implantados.
- Pilhas AWS CloudFormation implantadas.
- AWS CloudFormation Modelos locais.

Recursos implantados AWS

Você pode migrar AWS recursos implantados de um ambiente específico (Conta da AWS e Região da AWS) que não estejam associados a uma AWS CloudFormation pilha.

O AWS CDK CLI utiliza o serviço gerador IaC para verificar recursos em seu AWS ambiente e coletar detalhes dos recursos. Para saber mais sobre o gerador IaC, consulte Geração de modelos para recursos existentes no Guia do AWS CloudFormation usuário.

Depois de coletar os detalhes dos recursos, AWS CDK CLI ele cria um novo aplicativo CDK que inclui uma única pilha contendo seus recursos migrados.

Pilhas implantadas AWS CloudFormation

Você pode migrar uma única AWS CloudFormation pilha para um novo AWS CDK aplicativo. Eles AWS CDK CLI recuperarão o AWS CloudFormation modelo da sua pilha e criarão um novo aplicativo CDK. O aplicativo CDK consistirá em uma única pilha que contém sua pilha AWS CloudFormation migrada.

AWS CloudFormation Modelos locais

Você pode migrar de um AWS CloudFormation modelo local. Os modelos locais podem ou não conter recursos implantados. Eles AWS CDK CLI criarão um novo aplicativo CDK que contém uma única pilha com seus recursos.

Depois de migrar, você pode gerenciar, modificar e implantar seu aplicativo CDK para AWS CloudFormation provisionar ou atualizar seus recursos.

Benefícios do CDK Migrate

Historicamente, migrar recursos para lá AWS CDK tem sido um processo manual que requer tempo e experiência AWS CDK para começar. AWS CloudFormation Com o CDK Migrate, isso AWS CDK CLI facilita a maior parte do esforço de migração para você em uma fração do tempo. O CDK Migrate fará com que você comece rapidamente a usar o AWS CDK para desenvolver e gerenciar aplicativos novos e existentes no. AWS

Considerações

Considerações gerais

CDK Migrate versus CDK Import

O `cdk import` comando pode importar recursos implantados em um aplicativo CDK novo ou existente. Ao importar, cada recurso precisará ser definido manualmente como uma construção

L1 em seu aplicativo. Recomendamos usar `cdk import` para importar um ou mais recursos por vez em um aplicativo CDK novo ou existente. Para saber mais, consulte Importar recursos existentes para uma pilha.

O `cdk migrate` comando migra de recursos implantados, AWS CloudFormation pilhas implantadas ou AWS CloudFormation modelos locais para um novo aplicativo CDK. Durante a migração, os AWS CDK CLI usos `cdk import` para importar seus recursos para o novo aplicativo CDK. Isso AWS CDK CLI também gera construções L1 para cada recurso para você. Recomendamos usar `cdk migrate` ao importar de uma fonte de migração compatível para um novo AWS CDK aplicativo.

O CDK Migrate cria somente construções L1

O aplicativo CDK recém-criado incluirá somente construções L1. Você pode adicionar construções de alto nível ao seu aplicativo após a migração.

O CDK Migrate cria aplicativos CDK que contêm uma única pilha

O aplicativo CDK recém-criado conterá uma única pilha.

Ao migrar os recursos implantados, todos os recursos migrados estarão contidos em uma única pilha no novo aplicativo CDK.

Ao migrar AWS CloudFormation pilhas, você só pode migrar uma única pilha para uma única AWS CloudFormation pilha no novo aplicativo CDK.

Migração de ativos

Os ativos do projeto, como AWS Lambda código, não migrarão diretamente para o novo aplicativo CDK. Após a migração, você pode especificar os valores dos ativos para incluí-los no aplicativo CDK.

Migração de recursos com estado

Ao migrar recursos com estado, como bancos de dados e buckets do Amazon Simple Storage Service (Amazon S3), você geralmente deseja migrar o recurso existente em vez de criar um novo recurso.

Para migrar e preservar recursos com estado, faça o seguinte:

- Verifique se seu recurso com estado é compatível com importação. Para obter mais informações, consulte Suporte ao tipo de recurso no Guia AWS CloudFormation do usuário.

- Após a migração, verifique se a ID lógica do recurso migrado no novo aplicativo CDK corresponde à ID lógica do recurso implantado.
- Se estiver migrando de uma AWS CloudFormation pilha, verifique se o nome da pilha no novo aplicativo CDK corresponde à pilha. AWS CloudFormation
- Implante o aplicativo CDK usando a mesma AWS conta e Região da AWS do recurso migrado.

Considerações ao migrar de um modelo AWS CloudFormation

O CDK Migrate oferece suporte à migração de modelo único

Ao migrar AWS CloudFormation modelos, você pode selecionar um único modelo para migrar. Não há suporte para modelos aninhados.

Migração de modelos com funções intrínsecas

Ao migrar de um AWS CloudFormation modelo que usa funções intrínsecas, eles AWS CDK CLI tentarão migrar sua lógica para o aplicativo CDK com a classe. `Fn` Para saber mais, consulte a classe Fn na Referência da AWS Cloud Development Kit (AWS CDK) API.

Considerações ao migrar dos recursos implantados

Limitações de verificação

Ao escanear seu ambiente em busca de recursos, o gerador IaC tem limitações específicas nos dados que ele pode recuperar e limitações de cota durante a digitalização. Para saber mais, consulte Considerações no Guia do AWS CloudFormation usuário.

Pré-requisitos

Antes de usar o `cdk migrate` comando, faça o seguinte:

1. Estabeleça a autenticação com AWS. Para obter instruções, consulte Etapa 2: Configurar o acesso programático.
2. Instalar ou atualizar a AWS CDK CLI. Para obter instruções de instalação, consulte Etapa 3: instalar o AWS CDKCLI.

Comece a usar o CDK Migrate

Para começar, execute o AWS CDK CLI `cdk migrate` comando em um diretório de sua escolha. Forneça as opções necessárias e opcionais, dependendo do tipo de migração que você está realizando.

Para obter uma lista completa e uma descrição das opções com as quais você pode usar`cdk migrate`, consulte referência de comando da cdk migrate.

A seguir estão algumas opções importantes que você talvez queira fornecer.

Nome da stack

A única opção necessária é`--stack-name`. Use essa opção para especificar um nome para a pilha que será criada no AWS CDK aplicativo após a migração. O nome da pilha também será usado como o nome da sua AWS CloudFormation pilha na implantação.

Idioma

Use `--language` para especificar a linguagem de programação do novo aplicativo CDK.

AWS conta e Região da AWS

O AWS CDK CLI recupera a AWS conta e Região da AWS as informações de fontes padrão. Para ter mais informações, consulte Etapa 2: Configurar o acesso programático. Você pode usar `--account` as `--region` opções com `cdk migrate` para fornecer outros valores.

Diretório de saída do seu novo projeto CDK

Por padrão, o AWS CDK CLI criará um novo projeto CDK em seu diretório de trabalho e usará o valor fornecido `--stack-name` para nomear a pasta do projeto. Se uma pasta com o mesmo nome existir atualmente, o AWS CDK CLI substituirá essa pasta.

Você pode especificar um caminho de saída diferente para a nova pasta do projeto CDK com a `--output-path` opção.

Fonte de migração

Forneça uma opção para especificar a origem da qual você está migrando.

- `--from-path`— Migre de um AWS CloudFormation modelo local.
- `--from-scan`— Migre dos recursos implantados em uma AWS conta e. Região da AWS
- `--from-stack`— Migre de uma AWS CloudFormation pilha.

Dependendo da fonte de migração, você pode fornecer opções adicionais para personalizar o `cdk migrate` comando.

Migre de uma pilha AWS CloudFormation

Para migrar de uma AWS CloudFormation pilha implantada, forneça a opção. `--from-stack` Forneça o nome da sua AWS CloudFormation pilha implantada com. `--stack-name` Veja um exemplo a seguir:

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

Eles AWS CDK CLI farão o seguinte:

1. Recupere o AWS CloudFormation modelo da sua pilha implantada.
2. Execute `cdk init` para inicializar um novo aplicativo CDK.
3. Crie uma pilha dentro do aplicativo CDK que contenha sua pilha AWS CloudFormation migrada.

Quando você migra de uma AWS CloudFormation pilha implantada, as AWS CDK CLI tentativas de combinar as IDs lógicas dos recursos implantados e o nome da pilha implantada com os recursos e a AWS CloudFormation pilha migrados no novo aplicativo CDK.

Após a migração, você pode gerenciar e modificar seu aplicativo CDK normalmente. Ao implantar, AWS CloudFormation identificará a implantação como uma atualização da AWS CloudFormation pilha devido ao nome da AWS CloudFormation pilha correspondente. Os recursos com IDs lógicos correspondentes serão atualizados. Para obter mais informações sobre implantação, consulteGerencie e implante seu aplicativo CDK.

Migrar de um modelo AWS CloudFormation

O CDK Migrate suporta a migração de AWS CloudFormation modelos formatados em ou. `JSON` `YAML`

Para migrar de um AWS CloudFormation modelo local, use a `--from-path` opção e forneça um caminho para o modelo local. Você também deve fornecer a `--stack-name` opção necessária. Veja um exemplo a seguir:

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

Eles AWS CDK CLI farão o seguinte:

1. Recupere seu AWS CloudFormation modelo local.
2. Execute `cdk init` para inicializar um novo aplicativo CDK.
3. Crie uma pilha no aplicativo CDK que contenha seu modelo AWS CloudFormation migrado.

Após a migração, você pode gerenciar e modificar seu aplicativo CDK normalmente. Na implantação, você tem as seguintes opções:

- Atualizar uma AWS CloudFormation pilha — Se o AWS CloudFormation modelo local tiver sido implantado anteriormente, você poderá atualizar a pilha AWS CloudFormation implantada.
- Implantar uma nova AWS CloudFormation pilha — Se o AWS CloudFormation modelo local nunca foi implantado, ou se você quiser criar uma nova pilha a partir de um modelo implantado anteriormente, você pode implantar uma nova pilha. AWS CloudFormation

Migrar de um modelo AWS SAM

Para migrar de um modelo AWS Serverless Application Model (AWS SAM), você deve primeiro convertê-lo em um AWS CloudFormation modelo ou implantá-lo para criar uma AWS CloudFormation pilha.

Para converter um AWS SAM modelo em AWS CloudFormation, você pode usar o AWS SAM CLI `sam validate --debug` comando. Talvez seja necessário `lint` definir como `false` em seu `samconfig.toml` arquivo antes de executar esse comando.

Para converter em uma AWS CloudFormation pilha, implante o AWS SAM modelo usando o. AWS SAM CLI Em seguida, migre da pilha implantada.

Migre dos recursos implantados

Para migrar dos AWS recursos implantados, forneça a `--from-scan` opção. Você também deve fornecer a `--stack-name` opção necessária. Veja um exemplo a seguir:

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

Eles AWS CDK CLI farão o seguinte:

1. Analise sua conta para obter detalhes de recursos e propriedades — Ele AWS CDK CLI utiliza o gerador IaC para escanear sua conta e coletar detalhes.
2. Gere um AWS CloudFormation modelo — Após a digitalização, AWS CDK CLI ele utiliza o gerador IaC para criar um AWS CloudFormation modelo.
3. Inicialize um novo aplicativo CDK e migre seu modelo — A AWS CDK CLI execução é executada `cdk init` para inicializar um novo AWS CDK aplicativo e migra seu AWS CloudFormation modelo para o aplicativo CDK como uma única pilha.

Use filtros

Por padrão, AWS CDK CLI ele examinará todo o AWS ambiente e migrará recursos até o limite máximo de cota do gerador IaC. Você pode fornecer filtros com o AWS CDK CLI para especificar um critério para os quais os recursos serão migrados da sua conta para o novo aplicativo CDK. Para saber mais, consulte `--filter`.

Escaneamento de recursos com o gerador IaC

Dependendo do número de recursos em sua conta, a verificação pode levar alguns minutos. Uma barra de progresso será exibida durante o processo de digitalização.

Tipos de recursos compatíveis

> Eles AWS CDK CLI migrarão recursos suportados pelo gerador IaC. Para obter uma lista completa, consulte Suporte ao tipo de recurso no Guia AWS CloudFormation do usuário.

Resolver propriedades somente de gravação

Alguns recursos compatíveis contêm propriedades somente para gravação. Essas propriedades podem ser gravadas para configurar a propriedade, mas não podem ser lidas pelo gerador IaC ou AWS CloudFormation para obter o valor. Por exemplo, uma propriedade usada para especificar uma senha de banco de dados pode ser somente de gravação por motivos de segurança.

Ao escanear recursos durante a migração, o gerador IaC detectará recursos que possam conter propriedades somente de gravação e os categorizará em qualquer um dos seguintes tipos:

- `MUTUALLY_EXCLUSIVE_PROPERTIES`— Essas são propriedades somente de gravação para um recurso específico que são intercambiáveis e têm uma finalidade semelhante. Uma das

propriedades mutuamente exclusivas é necessária para configurar seu recurso. Por exemplo, as propriedades `S3BucketImageUri`, e de um `AWS::Lambda::Function` recurso são `ZipFile` propriedades mutuamente exclusivas somente para gravação. Qualquer um deles pode ser usado para especificar seus ativos de função, mas você deve usar um.

- `MUTUALLY_EXCLUSIVE_TYPES`— Essas são propriedades obrigatórias somente de gravação que aceitam vários tipos de configuração. Por exemplo, a `Body` propriedade de um `AWS::ApiGateway::RestApi` recurso aceita um tipo de objeto ou string.
- `UNSUPPORTED_PROPERTIES`— Essas são propriedades somente para gravação que não se enquadram nas outras duas categorias. Elas são propriedades opcionais ou obrigatórias que aceitam uma matriz de objetos.

Para obter mais informações sobre propriedades somente de gravação e como o gerador de IaC as gerencia ao verificar recursos implantados e criar AWS CloudFormation modelos, consulte Gerador de IaC e propriedades somente de gravação no Guia do usuário.AWS CloudFormation

Após a migração, você deve especificar valores de propriedade somente para gravação no novo aplicativo CDK. Eles AWS CDK CLI anexarão uma seção de Avisos ao `ReadMe` arquivo do projeto CDK para documentar todas as propriedades somente de gravação que foram identificadas pelo gerador IaC. Veja um exemplo a seguir:

```
# 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
```

- Os avisos são organizados em títulos que identificam o ID lógico do recurso ao qual estão associados.
- Os avisos são categorizados por tipo. Esses tipos vêm diretamente do gerador IaC.

Para resolver propriedades somente de gravação

1. Identifique propriedades somente de gravação a serem resolvidas na seção Avisos do arquivo do seu projeto CDK. `ReadMe` Aqui, você pode anotar os recursos em seu aplicativo CDK que podem conter propriedades somente de gravação e identificar os tipos de propriedades somente de gravação que foram descobertos.

   a. Para`MUTUALLY_EXCLUSIVE_PROPERTIES`, determine qual propriedade mutuamente exclusiva configurar em seu AWS CDK aplicativo.

   b. Para`MUTUALLY_EXCLUSIVE_TYPES`, determine qual tipo aceito você usará para configurar a propriedade.

   c. Para`UNSUPPORTED_PROPERTIES`, determine se a propriedade é opcional ou obrigatória. Em seguida, configure conforme necessário.
2. Use a orientação do gerador IaC e das propriedades somente de gravação para fazer referência ao que significam os tipos de aviso.
3. No seu aplicativo CDK, os valores das propriedades somente de gravação a serem resolvidos também serão especificados na `Props` seção do seu aplicativo. Forneça os valores corretos aqui. Para obter descrições e orientações sobre propriedades, você pode consultar a Referência AWS CDK da API.

   Veja a seguir um exemplo da `Props` seção em um aplicativo CDK migrado com duas propriedades somente de gravação a serem resolvidas:

   ```
   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;
}
```

Depois de resolver todos os valores de propriedade somente para gravação, você estará pronto para se preparar para a implantação.

O arquivo migrate.json

AWS CDK CLIIsso cria um `migrate.json` arquivo em seu AWS CDK projeto durante a migração. Esse arquivo contém informações de referência sobre seus recursos implantados. Quando você implanta seu aplicativo CDK pela primeira vez, ele AWS CDK CLI usa esse arquivo para referenciar seus recursos implantados, associa seus recursos à nova AWS CloudFormation pilha e exclui o arquivo.

Gerencie e implante seu aplicativo CDK

Ao migrar para AWS CDK, o novo aplicativo CDK pode não estar pronto para implantação imediatamente. Este tópico descreve itens de ação a serem considerados ao gerenciar e implantar seu novo aplicativo CDK.

Preparar-se para implantação

Antes da implantação, você deve preparar seu aplicativo CDK.

Sintetize seu aplicativo

Use o `cdk synth` comando para sintetizar a pilha em seu aplicativo CDK em um modelo. AWS CloudFormation

Se você migrou de uma AWS CloudFormation pilha ou modelo implantado, você pode comparar o modelo sintetizado com o modelo migrado para verificar os valores de recursos e propriedades.

Para saber mais sobre `cdk synth`, consulte Sintetizando pilhas.

Execute um diff

Se você migrou de uma AWS CloudFormation pilha implantada, pode usar o comando cdk diff para comparar com a pilha em seu novo aplicativo CDK.

Para saber mais sobre cdk diff, consulte. Comparando pilhas

Inicialize seu ambiente

Se você estiver implantando a partir de um AWS ambiente pela primeira vez, use `cdk bootstrap` para preparar seu ambiente. Para saber mais, consulte Bootstrapping.

Implemente seu aplicativo CDK

Quando você implanta um aplicativo CDK, AWS CDK CLI ele utiliza o AWS CloudFormation serviço para provisionar seus recursos. Os recursos são agrupados em uma única pilha no aplicativo CDK e implantados como uma única pilha. AWS CloudFormation

Dependendo de onde você migrou, você pode implantar para criar uma nova AWS CloudFormation pilha ou atualizar uma pilha existente AWS CloudFormation .

Implante para criar uma nova AWS CloudFormation pilha

Se você migrou dos recursos implantados, eles AWS CDK CLI criarão automaticamente uma nova AWS CloudFormation pilha na implantação. Seus recursos implantados serão incluídos na nova AWS CloudFormation pilha.

Se você migrou de um AWS CloudFormation modelo local que nunca foi implantado, eles AWS CDK CLI criarão automaticamente uma nova AWS CloudFormation pilha na implantação.

Se você migrou de uma AWS CloudFormation pilha implantada ou de um AWS CloudFormation modelo local que foi implantado anteriormente, você pode implantar para criar uma nova pilha. AWS CloudFormation Para criar uma nova pilha, faça o seguinte:

- Implemente em um novo AWS ambiente. Isso consiste em usar uma AWS conta diferente ou implantar em outra Região da AWS.
- Se quiser implantar uma nova pilha no mesmo AWS ambiente da pilha ou modelo migrado, você deve modificar o nome da pilha em seu aplicativo CDK para um novo valor. Você também deve modificar todas as IDs lógicas dos recursos em seu aplicativo CDK. Em seguida, você pode implantar no mesmo ambiente para criar uma nova pilha e novos recursos.

Implemente para atualizar uma AWS CloudFormation pilha existente

Se você migrou de uma AWS CloudFormation pilha implantada ou de um AWS CloudFormation modelo local que foi implantado anteriormente, você pode implantar para atualizar a pilha existente. AWS CloudFormation

Verifique se o nome da pilha em seu aplicativo CDK corresponde ao nome da pilha implantada e implante no AWS CloudFormation mesmo ambiente. AWS

Trabalhando com o AWS CDK em linguagens de programação suportadas

Use o AWS Cloud Development Kit (AWS CDK) para definir sua Nuvem AWS infraestrutura com uma linguagem de programação compatível.

Tópicos

Importando a biblioteca AWS Construct

AWS CDK Isso inclui a AWS Construct Library, uma coleção de construções organizadas por AWS serviço. As construções estáveis da biblioteca são oferecidas em um único módulo, chamado pelo nome TypeScript do pacote:`aws-cdk-lib`. O nome real do pacote varia de acordo com o idioma.

TypeScript

Instalar	`instalação do npm aws-cdk-lib`
Importar	`const cdk = require ('aws-cdk-lib');`

JavaScript

Instalar	`instalação do npm aws-cdk-lib`

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

Python

Instalar	`instalação do python -m pip aws-cdk-lib`
Importar	`importar aws_cdk como cdk`

Java

Adicionar a `pom.xml`	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Importar	`importar software.amazon.awscdk.app;` (for example)

C#

Instalar	`dotnet adiciona pacote Amazon.cdk.lib`
Importar	`usando Amazon.cdk;`

A classe `construct` base e o código de suporte estão no `constructs` módulo. As construções experimentais, nas quais a API ainda está sendo refinada, são distribuídas como módulos separados.

A referência AWS CDK da API

A Referência da AWS CDK API fornece documentação detalhada das construções (e outros componentes) na biblioteca. Uma versão da Referência da API é fornecida para cada linguagem de programação compatível.

O material de referência de cada módulo é dividido nas seções a seguir.

- Visão geral: Material introdutório que você precisará conhecer para trabalhar com o serviço no AWS CDK, incluindo conceitos e exemplos.
- Construções: classes de biblioteca que representam um ou mais AWS recursos concretos. Esses são os recursos ou padrões “selecionados” (L2) (recursos L3) que fornecem uma interface de alto nível com padrões sensatos.
- Classes: classes não construtivas que fornecem a funcionalidade usada pelas construções no módulo.
- Estruturas: estruturas de dados (pacotes de atributos) que definem a estrutura de valores compostos, como propriedades (o `props` argumento das construções) e opções.
- Interfaces: as interfaces, cujos nomes começam todos com “I”, definem a funcionalidade mínima absoluta para a construção correspondente ou outra classe. O CDK usa interfaces de construção para representar AWS recursos definidos fora do seu AWS CDK aplicativo e referenciados por métodos como. `Bucket.fromBucketArn()`
- Enums: coleções de valores nomeados para uso na especificação de determinados parâmetros de construção. O uso de um valor enumerado permite que o CDK verifique a validade desses valores durante a síntese.
- CloudFormation Recursos: Essas construções L1, cujos nomes começam com “Cfn”, representam exatamente os recursos definidos na especificação. CloudFormation Eles são gerados automaticamente a partir dessa especificação em cada versão do CDK. Cada construção L2 ou L3 encapsula um ou mais recursos. CloudFormation
- CloudFormation Tipos de propriedade: a coleção de valores nomeados que definem as propriedades de cada CloudFormation recurso.

Interfaces comparadas com classes de construção

AWS CDK Ele usa interfaces de uma forma específica que pode não ser óbvia, mesmo se você estiver familiarizado com interfaces como um conceito de programação.

O AWS CDK suporta o uso de recursos definidos fora dos aplicativos do CDK usando métodos como`Bucket.fromBucketArn()`. Os recursos externos não podem ser modificados e podem não ter todas as funcionalidades disponíveis com os recursos definidos em seu aplicativo CDK usando, por exemplo, a `Bucket` classe. As interfaces, então, representam a funcionalidade mínima disponível no CDK para um determinado tipo de AWS recurso, incluindo recursos externos.

Ao instanciar recursos em seu aplicativo CDK, você deve sempre usar classes concretas, como. `Bucket` Ao especificar o tipo de argumento que você está aceitando em uma de suas próprias construções, use o tipo de interface, como `IBucket` se estivesse preparado para lidar com recursos externos (ou seja, não precisará alterá-los). Se você precisar de uma construção definida pelo CDK, especifique o tipo mais geral que você pode usar.

Algumas interfaces são versões mínimas de propriedades ou pacotes de opções associados a classes específicas, em vez de construções. Essas interfaces podem ser úteis na criação de subclasses para aceitar argumentos que você passará para sua classe principal. Se você precisar de uma ou mais propriedades adicionais, convém implementar ou derivar dessa interface ou de um tipo mais específico.

> **Note**
>
> Algumas linguagens de programação suportadas pelo AWS CDK não têm um recurso de interface. Nessas linguagens, as interfaces são apenas classes comuns. Você pode identificá-los por seus nomes, que seguem o padrão de um “I” inicial seguido pelo nome de alguma outra construção (por exemplo`IBucket`). As mesmas regras se aplicam.

Gerenciando dependências

As dependências do seu AWS CDK aplicativo ou biblioteca são gerenciadas usando ferramentas de gerenciamento de pacotes. Essas ferramentas são comumente usadas com as linguagens de programação.

Normalmente, ele AWS CDK suporta a ferramenta de gerenciamento de pacotes padrão ou oficial do idioma, se houver uma. Caso contrário, o AWS CDK suportará o idioma mais popular ou amplamente suportado. Você também poderá usar outras ferramentas, especialmente se elas funcionarem com as ferramentas suportadas. No entanto, o suporte oficial para outras ferramentas é limitado.

O AWS CDK suporta os seguintes gerenciadores de pacotes:

Idioma	Ferramenta de gerenciamento de pacotes compatível
TypeScript/JavaScript	NPM (Node Package Manager) ou Yarn
Python	PIP (Package Installer para Python)

Idioma	Ferramenta de gerenciamento de pacotes compatível
Java	Maven
C#	NuGet
Go	Módulos Go

Quando você cria um novo projeto usando o AWS CDK CLI `cdk init` comando, as dependências das bibliotecas principais e construções estáveis do CDK são especificadas automaticamente.

Para obter mais informações sobre o gerenciamento de dependências para linguagens de programação compatíveis, consulte o seguinte:

- Gerenciando dependências em TypeScript.
- Gerenciando dependências em JavaScript.
- Gerenciando dependências em Python.
- Gerenciando dependências em Java.
- Gerenciando dependências em C#.
- Gerenciando dependências em Go.

AWS CDK Comparando TypeScript com outros idiomas

TypeScript foi a primeira linguagem suportada para o desenvolvimento de AWS CDK aplicativos. Portanto, uma quantidade substancial de exemplos de código CDK é gravada TypeScript. Se você estiver desenvolvendo em outra linguagem, pode ser útil comparar como o AWS CDK código é implementado em TypeScript comparação com a linguagem de sua escolha. Isso pode ajudá-lo a usar os exemplos em toda a documentação.

Importando um módulo

TypeScript/JavaScript

TypeScript suporta a importação de um namespace inteiro ou de objetos individuais de um namespace. Cada namespace inclui construções e outras classes para uso com um determinado serviço. 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

Por exemplo TypeScript, o Python suporta importações de módulos com namespace e importações seletivas. Os namespaces em Python se parecem com aws_cdk. xxx, onde xxx representa um nome AWS de serviço, como s3 para Amazon S3. (O Amazon S3 é usado nesses exemplos).

```
# 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

As importações do Java funcionam de forma diferente TypeScript das nossas. Cada instrução de importação importa um único nome de classe de um determinado pacote ou todas as classes definidas nesse pacote (usando*). As classes podem ser acessadas usando o nome da classe por si só, se tiver sido importada, ou o nome da classe qualificada, incluindo seu pacote.

As bibliotecas têm o mesmo `software.amazon.awscdk.services.xxx` nome da AWS Construct Library (a biblioteca principal é`software.amazon.awscdk`). O ID do grupo Maven para AWS CDK pacotes é`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#

Em C#, você importa tipos com a `using` diretiva. Há dois estilos. Um fornece acesso a todos os tipos no namespace especificado usando seus nomes simples. Com o outro, você pode se referir ao próprio namespace usando um alias.

Os pacotes são nomeados da mesma forma que `Amazon.CDK.AWS.xxx` os pacotes da AWS Construct Library. (O módulo principal é`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

Cada módulo da AWS Construct Library é fornecido como um pacote 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(...)
```

Instanciando uma construção

AWS CDK as classes de construção têm o mesmo nome em todos os idiomas suportados. A maioria das linguagens usa a `new` palavra-chave para instanciar uma classe (Python e Go não). Além disso, na maioria dos idiomas, a palavra-chave `this` se refere à instância atual. (O Python é usado `self` por convenção.) Você deve passar uma referência à instância atual como `scope` parâmetro para cada construção criada.

O terceiro argumento para uma AWS CDK construção é `props` um objeto contendo os atributos necessários para construir a construção. Esse argumento pode ser opcional, mas quando necessário, as linguagens suportadas o tratam de forma idiomática. Os nomes dos atributos também são adaptados aos padrões de nomenclatura padrão da linguagem.

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

O Python não usa uma `new` palavra-chave ao instanciar uma classe. O argumento de propriedades é representado usando argumentos de palavra-chave e os argumentos são nomeados usando`snake_case`.

Se um valor de props for em si um pacote de atributos, ele será representado por uma classe com o nome da propriedade, que aceita argumentos de palavra-chave para as subpropriedades.

Em Python, a instância atual é passada para os métodos como o primeiro argumento, que é nomeado `self` por convenção.

```
# 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

Em Java, o argumento props é representado por uma classe chamada `XxxxProps` (por exemplo, `BucketProps` para as props da `Bucket` construção). Você constrói o argumento props usando um padrão de construtor.

Cada `XxxxProps` classe tem um construtor. Também existe um construtor conveniente para cada construção que constrói os adereços e a construção em uma única etapa, conforme mostrado no exemplo a seguir.

Os adereços são nomeados da mesma forma que em TypeScript, usando`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#

Em C#, os adereços são especificados usando um inicializador de objeto para uma classe chamada `XxxxProps` (por exemplo, `BucketProps` para os adereços da `Bucket` construção).

Os adereços são nomeados de forma semelhante a TypeScript, exceto usando`PascalCase`.

É conveniente usar a `var` palavra-chave ao instanciar uma construção, para que você não precise digitar o nome da classe duas vezes. No entanto, seu guia de estilo de código local pode variar.

```
// 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

Para criar uma construção em Go, chame a função `NewXxxxxx` onde `Xxxxxxx` está o nome da construção. As propriedades das construções são definidas como uma estrutura.

Em Go, todos os parâmetros de construção são ponteiros, incluindo valores como números, booleanos e cadeias de caracteres. Use as funções de conveniência, como `jsii.String` criar esses ponteiros.

```
// 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"),
 }})
```

Acessando membros

É comum se referir a atributos ou propriedades de construções e outras AWS CDK classes e usar esses valores como, por exemplo, entradas para criar outras construções. As diferenças de nomenclatura descritas anteriormente para os métodos também se aplicam aqui. Além disso, em Java, não é possível acessar os membros diretamente. Em vez disso, um método getter é fornecido.

TypeScript/JavaScript

Os nomes sãocamelCase.

```
bucket.bucketArn
```

Python

Os nomes sãosnake_case.

```
bucket.bucket_arn
```

Java

Um método getter é fornecido para cada propriedade; esses nomes sãocamelCase.

```
bucket.getBucketArn()
```

C#

Os nomes sãoPascalCase.

```
bucket.BucketArn
```

Go

Os nomes sãoPascalCase.

```
bucket.BucketArn
```

Constantes de enumeração

As constantes Enum têm como escopo uma classe e têm nomes em maiúsculas com sublinhados em todos os idiomas (às vezes chamadas de). SCREAMING_SNAKE_CASE Como os nomes das classes também usam a mesma letra maiúscula em todos os idiomas suportados, exceto Go, os nomes de enumeração qualificados também são os mesmos nesses idiomas.

```
s3.BucketEncryption.KMS_MANAGED
```

Em Go, as constantes enum são atributos do namespace do módulo e são escritas da seguinte forma.

```
awss3.BucketEncryption_KMS_MANAGED
```

Interfaces de objetos

O AWS CDK usa interfaces de TypeScript objetos para indicar que uma classe implementa um conjunto esperado de métodos e propriedades. Você pode reconhecer uma interface de objeto porque seu nome começa comI. Uma classe concreta indica as interfaces que ela implementa usando a `implements` palavra-chave.

TypeScript/JavaScript

> **Note**
>
> JavaScript não tem um recurso de interface. Você pode ignorar a `implements` palavra-chave e os nomes das classes que a seguem.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';
```

```
class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

O Python não tem um recurso de interface. No entanto, para o, AWS CDK você pode indicar a implementação da interface decorando sua classe com`@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

As estruturas Go não precisam declarar explicitamente quais interfaces elas implementam. O compilador Go determina a implementação com base nos métodos e propriedades disponíveis na estrutura. Por exemplo, no código a seguir, `MyAspect` implementa a `IAspect` interface porque ela fornece um `Visit` método que usa uma construção.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
 fmt.Println("Visited", *node.Node().Path())
}
```

Trabalhando com a AWS CDK pessoa TypeScript

TypeScript é uma linguagem de cliente totalmente compatível com o AWS Cloud Development Kit (AWS CDK) e é considerada estável. Trabalhar com o AWS CDK in TypeScript usa ferramentas familiares, incluindo o TypeScript compilador (`tsc`) da Microsoft, o Node.js e o Node Package Manager (`npm`). Você também pode usar o Yarn se preferir, embora os exemplos neste Guia usem o NPM. Os módulos que compõem a AWS Construct Library são distribuídos por meio do repositório NPM, npmjs.org.

Você pode usar qualquer editor ou IDE. Muitos AWS CDK desenvolvedores usam o Visual Studio Code (ou seu equivalente de código aberto, o VSCodium), que tem excelente suporte para o. TypeScript

Tópicos

- Comece com TypeScript
- Criação de um projeto
- Usando local tsc e cdk
- Gerenciando módulos da AWS Construct Libr
- Gerenciando dependências em TypeScript
- AWS CDK expressões idiomáticas em TypeScript

- Construindo, sintetizando e implantando

Comece com TypeScript

Para trabalhar com o AWS CDK, você deve ter uma AWS conta e credenciais e ter instalado o Node.js e o AWS CDK Toolkit. Consulte Começando com o AWS CDK.

Você também precisa de TypeScript si mesmo (versão 3.8 ou posterior). Se ainda não o tiver, você pode instalá-lo usandonpm.

```
npm install -g typescript
```

> **Note**
>
> Se você receber um erro de permissão e tiver acesso de administrador em seu sistema, tentesudo npm install -g typescript.

TypeScript Mantenha-se atualizado com um cliente regularnpm update -g typescript.

> **Note**
>
> Suspensão de uso de idioma de terceiros: a versão do idioma só é suportada até seu EOL (End Of Life) compartilhado pelo fornecedor ou pela comunidade e está sujeita a alterações mediante aviso prévio.

Criação de um projeto

Você cria um novo AWS CDK projeto invocando cdk init em um diretório vazio. Use a --language opção e especifiquetypescript:

```
mkdir my-project
cd my-project
cdk init app --language typescript
```

A criação de um projeto também instala o aws-cdk-libmódulo e suas dependências.

`cdk init`usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Os hífens no nome da pasta são convertidos em sublinhados. No entanto, caso contrário, o nome deve seguir a forma de um TypeScript identificador; por exemplo, ele não deve começar com um número ou conter espaços.

Usando local **`tsc`** e **`cdk`**

Na maioria das vezes, este guia pressupõe que você instale TypeScript o CDK Toolkit globalmente (`npm install -g typescript aws-cdk`), e os exemplos de comandos fornecidos (como`cdk synth`) seguem essa suposição. Essa abordagem facilita a atualização dos dois componentes e, como ambos adotam uma abordagem rígida de compatibilidade com versões anteriores, geralmente há pouco risco em sempre usar as versões mais recentes.

Algumas equipes preferem especificar todas as dependências em cada projeto, incluindo ferramentas como o TypeScript compilador e o CDK Toolkit. Essa prática permite que você fixe esses componentes em versões específicas e garanta que todos os desenvolvedores da sua equipe (e do seu ambiente de CI/CD) usem exatamente essas versões. Isso elimina uma possível fonte de mudança, ajudando a tornar as compilações e implantações mais consistentes e reproduzíveis.

O CDK inclui dependências para ambos TypeScript e o CDK Toolkit no modelo do TypeScript projeto. Portanto`package.json`, se você quiser usar essa abordagem, não precisará fazer nenhuma alteração em seu projeto. Tudo o que você precisa fazer é usar comandos ligeiramente diferentes para criar seu aplicativo e emitir `cdk` comandos.

Operation	Use ferramentas globais	Use ferramentas locais
Inicializar projeto	`cdk init --language typescript`	`npx aws-cdk init --language typescript`
Criação	`tsc`	`npm run build`
Execute o comando CDK Toolkit	`cdk ...`	`npm run cdk ...` or `npx aws-cdk ...`

`npx aws-cdk`executa a versão do CDK Toolkit instalada localmente no projeto atual, se houver, retornando à instalação global, se houver. Se não existir uma instalação global, `npx` baixa uma cópia temporária do CDK Toolkit e a executa. Você pode especificar uma versão arbitrária do CDK Toolkit usando a `@` sintaxe: prints. `npx aws-cdk@2.0 --version` `2.0.0`

> **Tip**
>
> Configure um alias para poder usar o `cdk` comando com uma instalação local do CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Gerenciando módulos da AWS Construct Libr

Use o Node Package Manager (`npm`) para instalar e atualizar os módulos da AWS Construct Library para uso por seus aplicativos, bem como por outros pacotes necessários. (Você pode usar `yarn` em vez de, `npm` se preferir.) `npm`também instala as dependências desses módulos automaticamente.

A maioria das AWS CDK construções está no pacote CDK principal, chamado`aws-cdk-lib`, que é uma dependência padrão em novos projetos criados pelo. cdk init Os módulos “experimentais” da AWS Construct Library, nos quais construções de nível superior ainda estão em desenvolvimento, são nomeados como. `@aws-cdk/`*`SERVICE-NAME`*`-alpha` O nome do serviço tem um prefixo aws-. Se você não tiver certeza do nome de um módulo, procure-o no NPM.

> **Note**
>
> A Referência da API CDK também mostra os nomes dos pacotes.

Por exemplo, o comando abaixo instala o módulo experimental para AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

O suporte da Construct Library de alguns serviços está em mais de um namespace. Por exemplo, além disso`aws-route53`, há três namespaces adicionais do Amazon Route 53,, `aws-route53-targets`e `aws-route53-patterns`, e. `aws-route53resolver`

As dependências do seu projeto são mantidas em`package.json`. Você pode editar esse arquivo para bloquear algumas ou todas as suas dependências em uma versão específica ou para permitir que elas sejam atualizadas para versões mais recentes sob determinados critérios. Para atualizar as dependências do NPM do seu projeto para a versão mais recente permitida de acordo com as regras especificadas em: `package.json`

```
npm update
```

Em TypeScript, você importa módulos para o seu código com o mesmo nome usado para instalá-los usando o NPM. Recomendamos as seguintes práticas ao importar AWS CDK classes e módulos da AWS Construct Library em seus aplicativos. Seguir essas diretrizes ajudará a tornar seu código consistente com outros AWS CDK aplicativos, além de ser mais fácil de entender.

- Use `import` diretivas no estilo ES6, não. `require()`
- Geralmente, importe classes individuais de`aws-cdk-lib`.

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

- Se você precisar de muitas classes`aws-cdk-lib`, poderá usar um alias de namespace `cdk` em vez de importar as classes individuais. Evite fazer as duas coisas.

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

- Geralmente, importe construções AWS de serviços usando aliases curtos de namespace.

  ```
  import { aws_s3 as s3 } from 'aws-cdk-lib';
  ```

Gerenciando dependências em TypeScript

Em projetos TypeScript CDK, as dependências são especificadas no `package.json` arquivo no diretório principal do projeto. Os AWS CDK módulos principais estão em um único NPM pacote chamado`aws-cdk-lib`.

Quando você instala um pacote usandonpm install, o NPM grava o pacote `package.json` para você.

Se preferir, você pode usar o Yarn no lugar do NPM. No entanto, o CDK não suporta o modo Yarn, que é o plug-and-play modo padrão no Yarn 2. Adicione o seguinte ao `.yarnrc.yml` arquivo do seu projeto para desativar esse recurso.

```
nodeLinker: node-modules
```

Aplicativos CDK

Veja a seguir um exemplo de `package.json` arquivo gerado pelo `cdk init --language typescript` comando:

```
{
  "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"
  }
}
```

Para aplicativos CDK implantáveis, `aws-cdk-lib` deve ser especificado na `dependencies` seção de. `package.json` Você pode usar um especificador de número de versão circunflexo (^) para

indicar que aceitará versões posteriores à especificada, desde que estejam na mesma versão principal.

Para construções experimentais, especifique as versões exatas dos módulos da biblioteca de construções alfa, que têm APIs que podem mudar. Não use ^ ou ~, pois versões posteriores desses módulos podem trazer alterações na API que podem prejudicar seu aplicativo.

Especifique as versões das bibliotecas e ferramentas necessárias para testar seu aplicativo (por exemplo, a estrutura de `jest` teste) na `devDependencies` seção de`package.json`. Opcionalmente, use ^ para especificar que versões compatíveis posteriores sejam aceitáveis.

Bibliotecas de construção de terceiros

Se você estiver desenvolvendo uma biblioteca de construção, especifique suas dependências usando uma combinação das `devDependencies` seções `peerDependencies` e, conforme mostrado no `package.json` arquivo de exemplo a seguir.

```
{
  "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"
  }
}
```

Em`peerDependencies`, use um acento circunflexo (^) para especificar a versão mais baixa com `aws-cdk-lib` a qual sua biblioteca trabalha. Isso maximiza a compatibilidade da sua biblioteca com uma variedade de versões do CDK. Especifique as versões exatas dos módulos de biblioteca de construção alfa, que têm APIs que podem mudar. `peerDependencies`O uso garante que haja apenas uma cópia de todas as bibliotecas CDK na `node_modules` árvore.

Em`devDependencies`, especifique as ferramentas e bibliotecas que você precisa para testar, opcionalmente com ^ para indicar que versões compatíveis posteriores são aceitáveis. Especifique exatamente (sem ^ ou ~) as versões mais baixas `aws-cdk-lib` e outros pacotes de CDK com os quais você anuncia a compatibilidade da sua biblioteca. Essa prática garante que seus testes sejam executados nessas versões. Dessa forma, se você usar inadvertidamente um recurso encontrado apenas em versões mais recentes, seus testes poderão detectá-lo.

> **Warning**
>
> `peerDependencies`são instalados automaticamente somente pelo NPM 7 e versões posteriores. Se você estiver usando o NPM 6 ou anterior, ou se estiver usando o Yarn, deverá incluir as dependências de suas dependências no. `devDependencies` Caso contrário, eles não serão instalados e você receberá um aviso sobre dependências de pares não resolvidas.

Instalando e atualizando dependências

Execute o comando a seguir para instalar as dependências do seu projeto.

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
```

Para atualizar os módulos instalados, os comandos anteriores npm install e yarn upgrade os comandos podem ser usados. Qualquer um dos comandos atualiza os pacotes `node_modules` para as versões mais recentes que atendem às regras do`package.json`. No entanto, eles não

`package.json` se atualizam sozinhos, o que talvez você queira fazer para definir uma nova versão mínima. Se você hospedar seu pacote em GitHub, poderá configurar as atualizações de versão do Dependabot para serem atualizadas automaticamente. `package.json` Como alternativa, use npm-check-updates.

> **Important**
>
> Por padrão, quando você instala ou atualiza dependências, o NPM e o Yarn escolhem a versão mais recente de cada pacote que atenda aos requisitos especificados em. `package.json` Sempre existe o risco de que essas versões sejam quebradas (acidentalmente ou intencionalmente). Teste minuciosamente depois de atualizar as dependências do seu projeto.

AWS CDK expressões idiomáticas em TypeScript

Adereços

Todas as classes da AWS Construct Library são instanciadas usando três argumentos: o escopo no qual a construção está sendo definida (seu pai na árvore de construção), um id e adereços. Argument props é um pacote de pares de chave/valor que a construção usa para configurar os AWS recursos que cria. Outras classes e métodos também usam o padrão “pacote de atributos” para argumentos.

Em TypeScript, a forma de `props` é definida usando uma interface que informa os argumentos obrigatórios e opcionais e seus tipos. Essa interface é definida para cada tipo de `props` argumento, geralmente específico para uma única construção ou método. Por exemplo, a construção Bucket (no`aws-cdk-lib/aws-s3 module`) especifica um `props` argumento em conformidade com a BucketPropsinterface.

Se uma propriedade for em si um objeto, por exemplo, a propriedade websiteRedirect de`BucketProps`, esse objeto terá sua própria interface à qual sua forma deverá estar em conformidade, nesse caso. RedirectTarget

Se você estiver subclassificando uma classe da AWS Construct Library (ou substituindo um método que usa um argumento semelhante a adereços), você pode herdar da interface existente para criar uma nova que especifique quaisquer novos adereços que seu código exija. Ao chamar a classe principal ou o método base, geralmente você pode passar todo o argumento props recebido, pois quaisquer atributos fornecidos no objeto, mas não especificados na interface, serão ignorados.

Uma versão futura do AWS CDK poderia coincidentemente adicionar uma nova propriedade com um nome que você usou para sua própria propriedade. Passar o valor que você recebe para a cadeia de herança pode causar um comportamento inesperado. É mais seguro passar uma cópia rasa dos adereços que você recebeu com sua propriedade removida ou configurada para. `undefined` Por exemplo: .

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Como alternativa, nomeie suas propriedades para que fique claro que elas pertencem à sua construção. Dessa forma, é improvável que eles colidam com propriedades em AWS CDK versões futuras. Se houver muitos deles, use um único objeto com nome apropriado para mantê-los.

Valores ausentes

Valores ausentes em um objeto (como adereços) têm o valor `undefined` em TypeScript. A versão 3.7 da linguagem introduziu operadores que simplificam o trabalho com esses valores, facilitando a especificação de padrões e o encadeamento em “curto-circuito” quando um valor indefinido é atingido. Para obter mais informações sobre esses recursos, consulte as notas de versão TypeScript 3.7, especificamente os dois primeiros recursos, Encadeamento opcional e coalescência nula.

Construindo, sintetizando e implantando

Geralmente, você deve estar no diretório raiz do projeto ao criar e executar seu aplicativo.

O Node.js não pode ser executado TypeScript diretamente; em vez disso, seu aplicativo é convertido para JavaScript usar o TypeScript compilador,`tsc`. O JavaScript código resultante é então executado.

O faz isso AWS CDK automaticamente sempre que precisa executar seu aplicativo. No entanto, pode ser útil compilar manualmente para verificar erros e executar testes. Para compilar seu TypeScript aplicativo manualmente, emitan`npm run build`. Você também pode `npm run watch` entrar no modo de observação, no qual o TypeScript compilador recria automaticamente seu aplicativo sempre que você salva as alterações em um arquivo de origem.

As pilhas definidas em seu AWS CDK aplicativo podem ser sintetizadas e implantadas individualmente ou em conjunto usando os comandos abaixo. Geralmente, você deve estar no diretório principal do seu projeto ao emiti-los.

- `cdk synth`: sintetiza um AWS CloudFormation modelo a partir de uma ou mais pilhas do seu aplicativo. AWS CDK

- `cdk deploy`: implanta os recursos definidos por uma ou mais pilhas em seu AWS CDK aplicativo para. AWS

Você pode especificar os nomes de várias pilhas a serem sintetizadas ou implantadas em um único comando. Se seu aplicativo definir apenas uma pilha, você não precisará especificá-la.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Você também pode usar os curingas * (qualquer número de caracteres) e? (qualquer caractere único) para identificar pilhas por padrão. Ao usar curingas, coloque o padrão entre aspas. Caso contrário, o shell pode tentar expandi-lo para os nomes dos arquivos no diretório atual antes de serem passados para o AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Você não precisa sintetizar explicitamente as pilhas antes de implantá-las; `cdk deploy` executa essa etapa para garantir que seu código mais recente seja implantado.

Para obter a documentação completa do `cdk` comando, consultethe section called “AWS CDK Kit de ferramentas”.

Trabalhando com a AWS CDK pessoa JavaScript

JavaScriptté uma linguagem de cliente totalmente compatível com o AWS CDK e é considerada estável. Trabalhar com o AWS Cloud Development Kit (AWS CDK) in JavaScript usa ferramentas familiares, incluindo o Node.js e o Node Package Manager (`npm`). Você também pode usar o Yarn se preferir, embora os exemplos neste Guia usem o NPM. Os módulos que compõem a AWS Construct Library são distribuídos por meio do repositório NPM, npmjs.org.

Você pode usar qualquer editor ou IDE. Muitos AWS CDK desenvolvedores usam o Visual Studio Code (ou seu equivalente de código aberto, o VSCodium), que tem um bom suporte para o. JavaScript

Tópicos

Conceitos básicos do JavaScript

Para trabalhar com o AWS CDK, você deve ter uma AWS conta e credenciais e ter instalado o Node.js e o AWS CDK Toolkit. Consulte Começando com o AWS CDK.

JavaScript AWS CDK os aplicativos não exigem pré-requisitos adicionais além desses.

> **Note**
>
> Suspensão de uso de idioma de terceiros: a versão do idioma só é suportada até seu EOL (End Of Life) compartilhado pelo fornecedor ou pela comunidade e está sujeita a alterações mediante aviso prévio.

Criação de um projeto

Você cria um novo AWS CDK projeto invocando `cdk init` em um diretório vazio. Use a `--language` opção e especifique`javascript`:

```
mkdir my-project
cd my-project
cdk init app --language javascript
```

A criação de um projeto também instala o `aws-cdk-lib`módulo e suas dependências.

`cdk init`usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Os hífens no nome da pasta são convertidos em sublinhados. No entanto, caso contrário, o nome deve seguir a forma de um JavaScript identificador; por exemplo, ele não deve começar com um número ou conter espaços.

Usando o local **cdk**

Na maioria das vezes, este guia pressupõe que você instale o CDK Toolkit globalmente (`npm install -g aws-cdk`), e os exemplos de comandos fornecidos (como`cdk synth`) seguem essa suposição. Essa abordagem facilita a atualização do CDK Toolkit e, como o CDK adota uma abordagem estrita em relação à compatibilidade com versões anteriores, geralmente há pouco risco em sempre usar a versão mais recente.

Algumas equipes preferem especificar todas as dependências em cada projeto, incluindo ferramentas como o CDK Toolkit. Essa prática permite que você fixe esses componentes em versões específicas e garanta que todos os desenvolvedores da sua equipe (e do seu ambiente de CI/CD) usem exatamente essas versões. Isso elimina uma possível fonte de mudança, ajudando a tornar as compilações e implantações mais consistentes e reproduzíveis.

O CDK inclui uma dependência para o CDK Toolkit no modelo do JavaScript projeto. Portanto`package.json`, se você quiser usar essa abordagem, não precisará fazer nenhuma alteração em seu projeto. Tudo o que você precisa fazer é usar comandos ligeiramente diferentes para criar seu aplicativo e emitir `cdk` comandos.

Operation	Use o kit de ferramentas global da CDK	Use o kit de ferramentas CDK local
Inicializar projeto	`cdk init --linguagem javascript`	`npx aws-cdk init --language javascript`
Execute o comando CDK Toolkit	`cdk...`	`npm execute cdk...` or `npx aws-cdk...`

`npx aws-cdk`executa a versão do CDK Toolkit instalada localmente no projeto atual, se houver, retornando à instalação global, se houver. Se não existir uma instalação global, `npx` baixa uma cópia temporária do CDK Toolkit e a executa. Você pode especificar uma versão arbitrária do CDK Toolkit usando a `@` sintaxe: prints. `npx aws-cdk@1.120 --version` `1.120.0`

> **Tip**
>
> Configure um alias para que você possa usar o `cdk` comando com uma instalação local do CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Gerenciando módulos da AWS Construct Libr

Use o Node Package Manager (`npm`) para instalar e atualizar os módulos da AWS Construct Library para uso por seus aplicativos, bem como por outros pacotes necessários. (Você pode usar `yarn` em vez de, `npm` se preferir.) `npm`também instala as dependências desses módulos automaticamente.

A maioria das AWS CDK construções está no pacote CDK principal, chamado`aws-cdk-lib`, que é uma dependência padrão em novos projetos criados pelo. cdk init Os módulos “experimentais” da AWS Construct Library, nos quais construções de nível superior ainda estão em desenvolvimento, são nomeados como. `aws-cdk-lib/`*`SERVICE-NAME`*`-alpha` O nome do serviço tem um prefixo aws-. Se você não tiver certeza do nome de um módulo, procure-o no NPM.

> **Note**
>
> A Referência da API CDK também mostra os nomes dos pacotes.

Por exemplo, o comando abaixo instala o módulo experimental para AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

O suporte da Construct Library de alguns serviços está em mais de um namespace. Por exemplo, além disso`aws-route53`, há três namespaces adicionais do Amazon Route 53,, `aws-route53-targets`, `aws-route53-patterns`, e. `aws-route53resolver`

As dependências do seu projeto são mantidas em`package.json`. Você pode editar esse arquivo para bloquear algumas ou todas as suas dependências em uma versão específica ou para permitir que elas sejam atualizadas para versões mais recentes sob determinados critérios. Para atualizar as dependências do NPM do seu projeto para a versão mais recente permitida de acordo com as regras especificadas em: `package.json`

```
npm update
```

Em JavaScript, você importa módulos para o seu código com o mesmo nome usado para instalá-los usando o NPM. Recomendamos as seguintes práticas ao importar AWS CDK classes e módulos da AWS Construct Library em seus aplicativos. Seguir essas diretrizes ajudará a tornar seu código consistente com outros AWS CDK aplicativos, além de ser mais fácil de entender.

- Use`require()`, não diretivas do estilo ES6. `import` As versões mais antigas do Node.js não oferecem suporte às importações do ES6, portanto, usar a sintaxe mais antiga é mais amplamente compatível. (Se você realmente quiser usar as importações do ES6, use o esm para garantir que seu projeto seja compatível com todas as versões suportadas do Node.js.)
- Geralmente, importe classes individuais de`aws-cdk-lib`.

  ```
  const { App, Stack } = require('aws-cdk-lib');
  ```
- Se você precisar de muitas classes`aws-cdk-lib`, poderá usar um alias de namespace `cdk` em vez de importar as classes individuais. Evite fazer as duas coisas.

  ```
  const cdk = require('aws-cdk-lib');
  ```
- Geralmente, importe Bibliotecas do AWS Construct usando aliases curtos de namespace.

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

Gerenciando dependências em JavaScript

Em projetos JavaScript CDK, as dependências são especificadas no `package.json` arquivo no diretório principal do projeto. Os AWS CDK módulos principais estão em um único NPM pacote chamado`aws-cdk-lib`.

Quando você instala um pacote usando`npm install`, o NPM grava o pacote `package.json` para você.

Se preferir, você pode usar o Yarn no lugar do NPM. No entanto, o CDK não suporta o modo Yarn, que é o plug-and-play modo padrão no Yarn 2. Adicione o seguinte ao `.yarnrc.yml` arquivo do seu projeto para desativar esse recurso.

```
nodeLinker: node-modules
```

Aplicativos CDK

Veja a seguir um exemplo de `package.json` arquivo gerado pelo `cdk init --language typescript` comando. O arquivo gerado para JavaScript é semelhante, só que sem as entradas TypeScript relacionadas.

```
{
  "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"
  }
}
```

Para aplicativos CDK implantáveis, `aws-cdk-lib` deve ser especificado na `dependencies` seção de. `package.json` Você pode usar um especificador de número de versão com acento circunflexo

(^) para indicar que aceitará versões posteriores à especificada, desde que estejam na mesma versão principal.

Para construções experimentais, especifique as versões exatas dos módulos da biblioteca de construções alfa, que têm APIs que podem mudar. Não use ^ ou ~, pois versões posteriores desses módulos podem trazer alterações na API que podem prejudicar seu aplicativo.

Especifique as versões das bibliotecas e ferramentas necessárias para testar seu aplicativo (por exemplo, a estrutura de `jest` teste) na `devDependencies` seção de`package.json`. Opcionalmente, use ^ para especificar que versões compatíveis posteriores sejam aceitáveis.

Bibliotecas de construção de terceiros

Se você estiver desenvolvendo uma biblioteca de construção, especifique suas dependências usando uma combinação das `devDependencies` seções `peerDependencies` e, conforme mostrado no `package.json` arquivo de exemplo a seguir.

```
{
  "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"
  }
}
```

Em`peerDependencies`, use um acento circunflexo (^) para especificar a versão mais baixa com `aws-cdk-lib` a qual sua biblioteca trabalha. Isso maximiza a compatibilidade da sua biblioteca com uma variedade de versões do CDK. Especifique as versões exatas dos módulos de biblioteca de construção alfa, que têm APIs que podem mudar. `peerDependencies`O uso garante que haja apenas uma cópia de todas as bibliotecas CDK na `node_modules` árvore.

Em`devDependencies`, especifique as ferramentas e bibliotecas que você precisa para testar, opcionalmente com ^ para indicar que versões compatíveis posteriores são aceitáveis. Especifique exatamente (sem ^ ou ~) as versões mais baixas `aws-cdk-lib` e outros pacotes de CDK com os quais você anuncia a compatibilidade da sua biblioteca. Essa prática garante que seus testes sejam executados nessas versões. Dessa forma, se você usar inadvertidamente um recurso encontrado apenas em versões mais recentes, seus testes poderão detectá-lo.

> **Warning**
>
> `peerDependencies`são instalados automaticamente somente pelo NPM 7 e versões posteriores. Se você estiver usando o NPM 6 ou anterior, ou se estiver usando o Yarn, deverá incluir as dependências de suas dependências no. `devDependencies` Caso contrário, eles não serão instalados e você receberá um aviso sobre dependências de pares não resolvidas.

Instalando e atualizando dependências

Execute o comando a seguir para instalar as dependências do seu projeto.

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
```

Para atualizar os módulos instalados, os comandos anteriores npm install e yarn upgrade os comandos podem ser usados. Qualquer um dos comandos atualiza os pacotes `node_modules` para as versões mais recentes que atendem às regras do`package.json`. No entanto, eles não

`package.json` se atualizam sozinhos, o que talvez você queira fazer para definir uma nova versão mínima. Se você hospedar seu pacote em GitHub, você pode configurar as atualizações de versão do Dependabot para serem atualizadas automaticamente. `package.json` Como alternativa, use npm-check-updates.

> **Important**
>
> Por padrão, quando você instala ou atualiza dependências, o NPM e o Yarn escolhem a versão mais recente de cada pacote que atenda aos requisitos especificados em. `package.json` Sempre existe o risco de que essas versões sejam quebradas (acidentalmente ou intencionalmente). Teste minuciosamente depois de atualizar as dependências do seu projeto.

AWS CDK expressões idiomáticas em JavaScript

Adereços

Todas as classes da AWS Construct Library são instanciadas usando três argumentos: o escopo no qual a construção está sendo definida (seu pai na árvore de construção), um id e props, um pacote de pares de chave/valor que a construção usa para configurar os recursos que cria. AWS Outras classes e métodos também usam o padrão “pacote de atributos” para argumentos.

Usar um IDE ou editor que tenha um bom JavaScript preenchimento automático ajudará a evitar erros ortográficos nos nomes das propriedades. Se uma construção está esperando uma `encryptionKeys` propriedade e você a digita`encryptionkeys`, ao instanciar a construção, você não passou o valor pretendido. Isso pode causar um erro no momento da síntese, se a propriedade for necessária, ou fazer com que a propriedade seja ignorada silenciosamente se for opcional. No último caso, você pode obter um comportamento padrão que pretendia substituir. Tome cuidado especial aqui.

Ao criar uma subclasse de uma classe da AWS Construct Library (ou substituir um método que usa um argumento semelhante a adereços), talvez você queira aceitar propriedades adicionais para seu próprio uso. Esses valores serão ignorados pela classe principal ou pelo método substituído, porque eles nunca são acessados nesse código, então você geralmente pode transmitir todas as props recebidas.

Uma versão futura do AWS CDK poderia coincidentemente adicionar uma nova propriedade com um nome que você usou para sua própria propriedade. Passar o valor que você recebe para a cadeia

de herança pode causar um comportamento inesperado. É mais seguro passar uma cópia rasa dos adereços que você recebeu com sua propriedade removida ou configurada para. `undefined` Por exemplo: .

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Como alternativa, nomeie suas propriedades para que fique claro que elas pertencem à sua construção. Dessa forma, é improvável que eles colidam com propriedades em AWS CDK versões futuras. Se houver muitos deles, use um único objeto com nome apropriado para mantê-los.

Valores ausentes

Valores ausentes em um objeto (como`props`) têm o valor `undefined` em JavaScript. As técnicas usuais se aplicam para lidar com elas. Por exemplo, um idioma comum para acessar uma propriedade de um valor que pode ser indefinido é o seguinte:

```
// 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;
```

No entanto, se a pudesse ter algum outro valor “falso”`undefined`, é melhor tornar o teste mais explícito. Aqui, aproveitaremos o fato de que `null` e `undefined` somos iguais para testar os dois ao mesmo tempo:

```
let c = a == null ? a : a.b;
```

> **Tip**
>
> O Node.js 14.0 e versões posteriores oferecem suporte a novos operadores que podem simplificar o tratamento de valores indefinidos. Para obter mais informações, consulte as propostas opcionais de encadeamento e coalescência nula.

Sintetizando e implantando

As pilhas definidas em seu AWS CDK aplicativo podem ser sintetizadas e implantadas individualmente ou em conjunto usando os comandos abaixo. Geralmente, você deve estar no diretório principal do seu projeto ao emiti-los.

- `cdk synth`: sintetiza um AWS CloudFormation modelo a partir de uma ou mais pilhas do seu aplicativo. AWS CDK
- `cdk deploy`: implanta os recursos definidos por uma ou mais pilhas em seu AWS CDK aplicativo para. AWS

Você pode especificar os nomes de várias pilhas a serem sintetizadas ou implantadas em um único comando. Se seu aplicativo definir apenas uma pilha, você não precisará especificá-la.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Você também pode usar os curingas * (qualquer número de caracteres) e? (qualquer caractere único) para identificar pilhas por padrão. Ao usar curingas, coloque o padrão entre aspas. Caso contrário, o shell pode tentar expandi-lo para os nomes dos arquivos no diretório atual antes de serem passados para o AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Você não precisa sintetizar explicitamente as pilhas antes de implantá-las; `cdk deploy` executa essa etapa para garantir que seu código mais recente seja implantado.

Para obter a documentação completa do `cdk` comando, consultethe section called “AWS CDK Kit de ferramentas”.

Usando TypeScript exemplos com JavaScript

TypeScriptté a linguagem que usamos para desenvolver o AWS CDK, e foi a primeira linguagem suportada para o desenvolvimento de aplicativos, então muitos exemplos de AWS CDK código disponíveis estão escritos em TypeScript. Esses exemplos de código podem ser um bom recurso para JavaScript desenvolvedores; você só precisa remover as partes TypeScript específicas do código.

TypeScript os snippets geralmente usam o ECMAScript `import` e as `export` palavras-chave mais recentes para importar objetos de outros módulos e declarar que os objetos serão disponibilizados

fora do módulo atual. O Node.js acaba de começar a oferecer suporte a essas palavras-chave em suas versões mais recentes. Dependendo da versão do Node.js que você está usando (ou deseja oferecer suporte), você pode reescrever as importações e exportações para usar a sintaxe mais antiga.

As importações podem ser substituídas por chamadas para a `require()` função.

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');
```

As exportações podem ser atribuídas ao `module.exports` objeto.

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**
>
> Uma alternativa para usar as importações e exportações de estilo antigo é usar o `esm`módulo.

Depois de classificar as importações e exportações, você pode se aprofundar no código real. Você pode se deparar com esses recursos comumente usados TypeScript :

- Anotações de tipo
- Definições de interface
- Conversões de tipo/transmissões
- Modificadores de acesso

As anotações de tipo podem ser fornecidas para variáveis, membros da classe, parâmetros da função e tipos de retorno da função. Para variáveis, parâmetros e membros, os tipos são especificados seguindo o identificador com dois pontos e o tipo. Os valores de retorno da função seguem a assinatura da função e consistem em dois pontos e no tipo.

Para converter o código com anotação de tipo em JavaScript, remova os dois pontos e o tipo. Os membros da classe devem ter algum valor em JavaScript; defina-os como `undefined` se eles tiverem apenas uma anotação de tipo em. 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) {
    // ...
}
```

Em TypeScript, as interfaces são usadas para dar um nome aos pacotes de propriedades obrigatórias e opcionais e seus tipos. Em seguida, você pode usar o nome da interface como uma anotação de tipo. TypeScript garantirá que o objeto que você usa como, por exemplo, argumento para uma função tenha as propriedades necessárias dos tipos certos.

```
interface myFuncProps {
    code: lambda.Code,
    handler?: string
}
```

JavaScript não tem um recurso de interface, portanto, depois de remover as anotações de tipo, exclua totalmente as declarações da interface.

Quando uma função ou método retorna um tipo de uso geral (como`object`), mas você deseja tratar esse valor como um tipo filho mais específico para acessar propriedades ou métodos que não fazem parte da interface do tipo mais geral, TypeScript você pode converter o valor usando `as` seguido por um nome de tipo ou interface. JavaScript não suporta (ou precisa) isso, então simplesmente remova `as` o identificador a seguir. Uma sintaxe de conversão menos comum é usar um nome de tipo entre colchetes`<LikeThis>`; essas conversões também devem ser removidas.

Finalmente, TypeScript suporta os modificadores de acesso `public``protected`, e `private` para membros de classes. Todos os alunos da turma JavaScript são públicos. Basta remover esses modificadores onde quer que você os veja.

Saber como identificar e remover esses TypeScript recursos ajuda muito a adaptar pequenos TypeScript trechos a. JavaScript Mas pode ser impraticável converter TypeScript exemplos mais longos dessa forma, pois é mais provável que eles usem outros TypeScript recursos. Para essas situações, recomendamos o Sucrase. Sucrase não reclamará se o código usar uma variável indefinida, por exemplo, como faria. `tsc` Se for sintaticamente válido, com poucas exceções, o

Sucrase pode traduzi-lo para. JavaScript Isso o torna particularmente valioso para converter trechos que podem não ser executáveis sozinhos.

Migrando para TypeScript

Muitos JavaScript desenvolvedores migram à TypeScriptmedida que seus projetos se tornam maiores e mais complexos. TypeScript é um superconjunto de JavaScript — todo JavaScript código é válido TypeScript , portanto, nenhuma alteração em seu código é necessária — e também é uma linguagem compatível. AWS CDK As anotações de tipo e outros TypeScript recursos são opcionais e podem ser adicionados ao seu AWS CDK aplicativo à medida que você achar valor neles. TypeScript também oferece acesso antecipado a novos JavaScript recursos, como encadeamento opcional e coalescência nula, antes de serem finalizados, sem exigir que você atualize o Node.js.

TypeScriptAs interfaces “baseadas em formas” da, que definem pacotes de propriedades obrigatórias e opcionais (e seus tipos) em um objeto, permitem que erros comuns sejam detectados enquanto você escreve o código e facilitam que seu IDE forneça preenchimento automático robusto e outras dicas de codificação em tempo real.

A codificação TypeScript envolve uma etapa adicional: compilar seu aplicativo com o TypeScript compilador,. `tsc` Para AWS CDK aplicativos típicos, a compilação requer alguns segundos no máximo.

A maneira mais fácil de migrar um JavaScript AWS CDK aplicativo existente para o novo projeto TypeScript é criar um novo TypeScript projeto usando `ecdk init app --language typescript`, em seguida, copiar seus arquivos de origem (e quaisquer outros arquivos necessários, como ativos como o código-fonte da AWS Lambda função) para o novo projeto. Renomeie seus JavaScript arquivos para finalizar `.ts` e começar a desenvolver em TypeScript.

Trabalhando com o AWS CDK em Python

Python é uma linguagem cliente totalmente compatível com o AWS Cloud Development Kit (AWS CDK) e é considerada estável. Trabalhar com o AWS CDK em Python usa ferramentas familiares, incluindo a implementação padrão do Python (CPython), ambientes virtuais com `virtualenv` e o instalador do pacote Python. `pip` Os módulos que compõem a AWS Construct Library são distribuídos via pypi.org. A versão Python do AWS CDK evento usa identificadores no estilo Python (por exemplo, nomes de métodos). `snake_case`

Você pode usar qualquer editor ou IDE. Muitos AWS CDK desenvolvedores usam o Visual Studio Code (ou seu equivalente de código aberto, o VSCodium), que tem um bom suporte para Python por

meio de uma extensão oficial. O editor IDLE incluído no Python será suficiente para começar. Os módulos Python do AWS CDK do têm dicas de tipo, que são úteis para uma ferramenta de linting ou um IDE que oferece suporte à validação de tipo.

Tópicos

Conceitos básicos do Python

Para trabalhar com o AWS CDK, você deve ter uma AWS conta e credenciais e ter instalado o Node.js e o AWS CDK Toolkit. Consulte Começando com o AWS CDK.

Os AWS CDK aplicativos Python exigem o Python 3.6 ou posterior. Se você ainda não o tiver instalado, baixe uma versão compatível para seu sistema operacional em python.org. Se você executa Linux, seu sistema pode ter vindo com uma versão compatível ou você pode instalá-la usando o gerenciador de pacotes da sua distribuição (`yum`,`apt`, etc.). Usuários de Mac podem se interessar pelo Homebrew, um gerenciador de pacotes no estilo Linux para macOS.

> **Note**
>
> Suspensão de uso de idioma de terceiros: a versão do idioma só é suportada até seu EOL (End Of Life) compartilhado pelo fornecedor ou pela comunidade e está sujeita a alterações mediante aviso prévio.

O instalador do pacote Python e `pip` o gerenciador de ambiente virtual também são necessários. `virtualenv` As instalações do Windows de versões compatíveis do Python incluem essas ferramentas. No Linux, `pip` e `virtualenv` podem ser fornecidos como pacotes separados em seu gerenciador de pacotes. Como alternativa, você pode instalá-los com os seguintes comandos:

```
python -m ensurepip --upgrade
```

```
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

Se você encontrar um erro de permissão, execute os comandos acima com o `--user` sinalizador para que os módulos sejam instalados em seu diretório de usuário ou use `sudo` para obter as permissões para instalar os módulos em todo o sistema.

> **Note**
>
> É comum que as distribuições Linux usem o nome executável do `python3` Python 3.x e se `python` refiram a uma instalação do Python 2.x. Algumas distribuições têm um pacote opcional que você pode instalar que faz com que o `python` comando se refira ao Python 3. Caso contrário, você pode ajustar o comando usado para executar seu aplicativo editando `cdk.json` no diretório principal do projeto.

> **Note**
>
> No Windows, talvez você queira invocar o Python (pipe) usando py o executável, o inicializador >Python para Windows. Entre outras coisas, o lançador permite que você especifique facilmente qual versão instalada do Python você deseja usar.
> Se digitar python na linha de comando resultar em uma mensagem sobre a instalação do Python na Windows Store, mesmo depois de instalar uma versão do Python para Windows, abra o painel de configurações Gerenciar aliases de execução de aplicativos do Windows e desative as duas entradas do App Installer para Python.

Criação de um projeto

Você cria um novo AWS CDK projeto invocando `cdk init` em um diretório vazio. Use a `--language` opção e especifique`python`:

```
mkdir my-project
cd my-project
cdk init app --language python
```

`cdk init`usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Os hífens no nome da pasta são convertidos em sublinhados. No

entanto, caso contrário, o nome deve seguir a forma de um identificador Python; por exemplo, ele não deve começar com um número ou conter espaços.

Para trabalhar com o novo projeto, ative o respectivo ambiente virtual. Isso permite que as dependências do projeto sejam instaladas localmente na pasta do projeto, em vez de globalmente.

```
source .venv/bin/activate
```

> **Note**
>
> É possível reconhecer isso como o comando Mac/Linux para ativar um ambiente virtual. Os modelos do Python incluem um arquivo em lote, `source.bat`, que permite que o mesmo comando seja utilizado no Windows. O comando tradicional do Windows também funciona. `.venv\Scripts\activate.bat`
> Se você inicializou seu AWS CDK projeto usando o CDK Toolkit v1.70.0 ou anterior, seu ambiente virtual está no diretório em vez de. `.env .venv`

> **Important**
>
> Ative o ambiente virtual do projeto sempre que começar a trabalhar nele. Caso contrário, você não terá acesso aos módulos instalados lá, e os módulos que você instalar entrarão no diretório global do módulo Python (ou resultarão em um erro de permissão).

Depois de ativar seu ambiente virtual pela primeira vez, instale as dependências padrão do aplicativo:

```
python -m pip install -r requirements.txt
```

Gerenciando módulos da AWS Construct Libr

Use o instalador de pacotes Python,pip, para instalar e atualizar os módulos da AWS Construct Library para uso por seus aplicativos, bem como por outros pacotes necessários. piptambém instala as dependências desses módulos automaticamente. Se seu sistema não reconhecer pip como um comando independente, invoque-o pip como um módulo Python, assim:

```
python -m pip PIP-COMMAND
```

A maioria das AWS CDK construções está pronta. `aws-cdk-lib` Os módulos experimentais estão em módulos separados chamados de`aws-cdk.SERVICE-NAME.alpha`. O nome do serviço inclui um prefixo aws. Se você não tiver certeza do nome de um módulo, procure-o no PyPI. Por exemplo, o comando abaixo instala a AWS CodeStar biblioteca.

```
python -m pip install aws-cdk.aws-codestar-alpha
```

Algumas construções de serviços estão em mais de um namespace. Por exemplo, além disso`aws-cdk.aws-route53`, há três namespaces adicionais do Amazon Route 53, chamados `aws-route53-targets``aws-route53-patterns`, e. `aws-route53resolver`

> **Note**
>
> A edição Python da CDK API Reference também mostra os nomes dos pacotes.

Os nomes usados para importar módulos da AWS Construct Library em seu código Python são semelhantes aos seguintes.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

Recomendamos as seguintes práticas ao importar AWS CDK classes e módulos da AWS Construct Library em seus aplicativos. Seguir essas diretrizes ajudará a tornar seu código consistente com outros AWS CDK aplicativos, além de ser mais fácil de entender.

- Geralmente, importe classes individuais do nível superior`aws_cdk`.

  ```
  from aws_cdk import App, Construct
  ```

- Se você precisar de muitas classes do`aws_cdk`, poderá usar um alias de namespace `cdk` em vez de importar classes individuais. Evite fazer as duas coisas.

  ```
  import aws_cdk as cdk
  ```

- Geralmente, importe Bibliotecas do AWS Construct usando aliases curtos de namespace.

  ```
  import aws_cdk.aws_s3 as s3
  ```

Depois de instalar um módulo, atualize o `requirements.txt` arquivo do seu projeto, que lista as dependências do seu projeto. É melhor fazer isso manualmente em vez de usar`pip freeze`. `pip freeze`captura as versões atuais de todos os módulos instalados em seu ambiente virtual Python, o que pode ser útil ao agrupar um projeto para ser executado em outro lugar.

Normalmente, porém, você `requirements.txt` deve listar somente as dependências de nível superior (módulos dos quais seu aplicativo depende diretamente) e não as dependências dessas bibliotecas. Essa estratégia simplifica a atualização de suas dependências.

Você pode editar `requirements.txt` para permitir atualizações; basta substituir o número de versão == anterior por permitir atualizações ~= para uma versão compatível superior ou remover totalmente o requisito de versão para especificar a versão mais recente disponível do módulo.

Com a `requirements.txt` edição adequada para permitir atualizações, emita este comando para atualizar os módulos instalados do seu projeto a qualquer momento:

```
pip install --upgrade -r requirements.txt
```

Gerenciando dependências em Python

Em Python, você especifica dependências colocando-as em aplicativos ou `setup.py` em `requirements.txt` bibliotecas de construção. As dependências são então gerenciadas com a ferramenta PIP. O PIP é invocado de uma das seguintes formas:

```
pip command options
python -m pip command options
```

A python -m pip invocação funciona na maioria dos sistemas; pip requer que o executável do PIP esteja no caminho do sistema. Se pip não funcionar, tente substituí-lo porpython -m pip.

O cdk init --language python comando cria um ambiente virtual para seu novo projeto. Isso permite que cada projeto tenha suas próprias versões de dependências e também um `requirements.txt` arquivo básico. Você deve ativar esse ambiente virtual executando source .venv/bin/activate sempre que começar a trabalhar com o projeto.

Aplicativos CDK

Veja a seguir um exemplo de arquivo `requirements.txt`. Como o PIP não tem um recurso de bloqueio de dependências, recomendamos que você use o operador == para especificar as versões exatas de todas as dependências, conforme mostrado aqui.

```
aws-cdk-lib==2.14.0
aws-cdk.aws-appsync-alpha==2.10.0a0
```

A instalação de um módulo com pip install não o adiciona automaticamente ao`requirements.txt`. Você deve fazer isso sozinho. Se você quiser atualizar para uma versão posterior de uma dependência, edite o número da versão em`requirements.txt`.

Para instalar ou atualizar as dependências do seu projeto depois de criar ou edita`requirements.txt`, execute o seguinte:

```
python -m pip install -r requirements.txt
```

> **Tip**
>
> O pip freeze comando gera as versões de todas as dependências instaladas em um formato que pode ser gravado em um arquivo de texto. Isso pode ser usado como um arquivo de requisitos com`pip install -r`. Esse arquivo é conveniente para fixar todas as dependências (inclusive as transitivas) nas versões exatas com as quais você testou. Para evitar problemas ao atualizar pacotes posteriormente, use um arquivo separado para isso, como `freeze.txt` (não`requirements.txt`). Em seguida, regenere ao atualizar as dependências do seu projeto.

Bibliotecas de construção de terceiros

Nas bibliotecas, as dependências são especificadas em`setup.py`, para que as dependências transitivas sejam baixadas automaticamente quando o pacote é consumido por um aplicativo. Caso contrário, todo aplicativo que quiser usar seu pacote precisará copiar suas dependências para o deles`requirements.txt`. Um exemplo `setup.py` é mostrado aqui.

```
from setuptools import setup

setup(
  name='my-package',
  version='0.0.1',
  install_requires=[
    'aws-cdk-lib==2.14.0',
  ],
```

```
  ...
)
```

Para trabalhar no pacote para desenvolvimento, crie ou ative um ambiente virtual e execute o comando a seguir.

```
python -m pip install -e .
```

Embora o PIP instale automaticamente dependências transitivas, só pode haver uma cópia instalada de qualquer pacote. A versão mais alta especificada na árvore de dependências é selecionada; os aplicativos sempre têm a última palavra em qual versão dos pacotes serão instalados.

AWS CDK expressões idiomáticas em Python

Conflitos linguísticos

Em Python, `lambda` é uma palavra-chave de linguagem, então você não pode usá-la como um nome para o módulo da biblioteca de AWS Lambda construção ou funções Lambda. A convenção do Python para esses conflitos é usar um sublinhado à direita, como no nome da `lambda_` variável.

Por convenção, o segundo argumento para AWS CDK construções é denominado`id`. Ao escrever suas próprias pilhas e construções, chame um parâmetro de `id` “sombrear” a função integrada do Python`id()`, que retorna o identificador exclusivo de um objeto. Essa função não é usada com muita frequência, mas se você precisar dela em sua construção, renomeie o argumento, por exemplo`construct_id`.

Argumentos e propriedades

Todas as classes da AWS Construct Library são instanciadas usando três argumentos: o escopo no qual a construção está sendo definida (seu pai na árvore de construção), um id e props, um pacote de pares de chave/valor que a construção usa para configurar os recursos que cria. Outras classes e métodos também usam o padrão “pacote de atributos” para argumentos.

scope e id devem sempre ser passados como argumentos posicionais, não como argumentos de palavra-chave, porque seus nomes mudam se a construção aceitar uma propriedade chamada scope ou id.

Em Python, os adereços são expressos como argumentos de palavras-chave. Se um argumento contém estruturas de dados aninhadas, elas são expressas usando uma classe que usa seus

próprios argumentos de palavra-chave na instanciação. O mesmo padrão é aplicado a outras chamadas de método que usam um argumento estruturado.

Por exemplo, no `add_lifecycle_rule` método de um bucket do Amazon S3, a `transitions` propriedade é uma lista de `Transition` instâncias.

```
bucket.add_lifecycle_rule(
  transitions=[
    Transition(
      storage_class=StorageClass.GLACIER,
      transition_after=Duration.days(10)
    )
  ]
)
```

Ao estender uma classe ou substituir um método, talvez você queira aceitar argumentos adicionais para seus próprios propósitos que não sejam compreendidos pela classe principal. Nesse caso, você deve aceitar os argumentos que não lhe interessam usar o `**kwargs` idioma e usar argumentos somente com palavras-chave para aceitar os argumentos nos quais está interessado. Ao chamar o construtor do pai ou o método substituído, passe somente os argumentos que ele espera (geralmente apenas). `**kwargs` Passar argumentos de que a classe ou o método pai não espera resulta em um erro.

```
class MyConstruct(Construct):
    def __init__(self, id, *, MyProperty=42, **kwargs):
        super().__init__(self, id, **kwargs)
        # ...
```

Uma versão futura do AWS CDK poderia coincidentemente adicionar uma nova propriedade com um nome que você usou para sua própria propriedade. Isso não causará problemas técnicos para os usuários de sua construção ou método (como sua propriedade não é passada “para cima na cadeia”, a classe principal ou o método substituído simplesmente usará um valor padrão), mas pode causar confusão. Você pode evitar esse problema em potencial nomeando suas propriedades de forma que elas claramente pertençam à sua construção. Se houver muitas propriedades novas, agrupe-as em uma classe com nome apropriado e passe-a como um único argumento de palavra-chave.

Valores ausentes

Os AWS CDK usos `None` para representar valores ausentes ou indefinidos. Ao trabalhar com`**kwargs`, use o `get()` método do dicionário para fornecer um valor padrão se uma

propriedade não for fornecida. Evite usar`kwargs[...]`, pois isso aumenta os valores `KeyError` ausentes.

```
encrypted = kwargs.get("encrypted")         # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)  # specify default of False if property is
 missing
```

Alguns AWS CDK métodos (como `tryGetContext()` obter um valor de contexto de tempo de execução) podem retornar`None`, o que você precisará verificar explicitamente.

Usando interfaces

O Python não tem um recurso de interface como algumas outras linguagens, embora tenha classes básicas abstratas, que são semelhantes. (Se você não está familiarizado com interfaces, a Wikipedia tem uma boa introdução.) TypeScript, a linguagem na qual o AWS CDK é implementado, fornece interfaces, e construções e outros AWS CDK objetos geralmente exigem um objeto que adira a uma interface específica, em vez de herdar de uma classe específica. Portanto, ele AWS CDK fornece seu próprio recurso de interface como parte da camada JSII.

Para indicar que uma classe implementa uma interface específica, você pode usar o `@jsii.implements` decorador:

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Armadilhas de tipo

O Python usa tipagem dinâmica, em que todas as variáveis podem se referir a um valor de qualquer tipo. Os parâmetros e valores de retorno podem ser anotados com tipos, mas essas são “dicas” e não são impostas. Isso significa que, em Python, é fácil passar o tipo incorreto de valor para uma AWS CDK construção. Em vez de receber um erro de tipo durante a construção, como você faria em uma linguagem com digitação estática, você pode receber um erro de tempo de execução quando a camada JSII (que se traduz entre o Python e o TypeScript núcleo) não consegue lidar com o AWS CDK tipo inesperado.

Em nossa experiência, os erros de tipo que os programadores de Python cometem tendem a se enquadrar nessas categorias.

- Passar um único valor em que uma construção espera um contêiner (lista ou dicionário do Python) ou vice-versa.
- Passar um valor de um tipo associado a uma construção de camada 1 (`CfnXxxxxx`) para uma construção L2 ou L3, ou vice-versa.

Os módulos do AWS CDK Python incluem anotações de tipo, então você pode usar ferramentas que os suportam para ajudar com os tipos. Se você não estiver usando um IDE que ofereça suporte a eles PyCharm, talvez queira chamar o validador MyPyde tipo como uma etapa do processo de criação. Também existem verificadores de tipo de tempo de execução que podem melhorar as mensagens de erro para erros relacionados ao tipo.

Sintetizando e implantando

As pilhas definidas em seu AWS CDK aplicativo podem ser sintetizadas e implantadas individualmente ou em conjunto usando os comandos abaixo. Geralmente, você deve estar no diretório principal do seu projeto ao emiti-los.

- `cdk synth`: sintetiza um AWS CloudFormation modelo a partir de uma ou mais pilhas do seu aplicativo. AWS CDK
- `cdk deploy`: implanta os recursos definidos por uma ou mais pilhas em seu AWS CDK aplicativo para. AWS

Você pode especificar os nomes de várias pilhas a serem sintetizadas ou implantadas em um único comando. Se seu aplicativo definir apenas uma pilha, você não precisará especificá-la.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Você também pode usar os curingas * (qualquer número de caracteres) e? (qualquer caractere único) para identificar pilhas por padrão. Ao usar curingas, coloque o padrão entre aspas. Caso contrário, o shell pode tentar expandi-lo para os nomes dos arquivos no diretório atual antes de serem passados para o AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
```

```
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Você não precisa sintetizar explicitamente as pilhas antes de implantá-las; `cdk deploy` executa essa etapa para garantir que seu código mais recente seja implantado.

Para obter a documentação completa do `cdk` comando, consultethe section called “AWS CDK Kit de ferramentas”.

Trabalhando com o AWS CDK em Java

Java é uma linguagem de cliente totalmente compatível com o AWS CDK e é considerada estável. Você pode desenvolver AWS CDK aplicativos em Java usando ferramentas familiares, incluindo o JDK (da Oracle ou uma distribuição do OpenJDK, como o Amazon Corretto) e o Apache Maven.

O AWS CDK suporta Java 8 e versões posteriores. No entanto, recomendamos usar a versão mais recente possível, pois as versões posteriores da linguagem incluem melhorias que são particularmente convenientes para o desenvolvimento de AWS CDK aplicativos. Por exemplo, o Java 9 introduz o `Map.of()` método (uma maneira conveniente de declarar hashmaps que seriam escritos como literais de objeto em). TypeScript O Java 10 introduz a inferência de tipos locais usando a `var` palavra-chave.

> **Note**
>
> A maioria dos exemplos de código neste Guia do desenvolvedor funciona com o Java 8. Alguns exemplos usam`Map.of()`; esses exemplos incluem comentários observando que eles exigem o Java 9.

Você pode usar qualquer editor de texto ou um IDE Java que possa ler projetos Maven para trabalhar em seus AWS CDK aplicativos. Fornecemos dicas sobre o Eclipse neste guia, mas o IntelliJ IDEA e outros IDEs podem importar projetos Maven e podem ser usados para desenvolver aplicativos em Java. NetBeans AWS CDK

É possível escrever AWS CDK aplicativos em linguagens hospedadas pela JVM que não sejam Java (por exemplo, Kotlin, Groovy, Clojure ou Scala), mas a experiência pode não ser particularmente idiomática e não podemos fornecer suporte para essas linguagens.

Tópicos

Conceitos básicos do Java

Para trabalhar com o AWS CDK, você deve ter uma AWS conta e credenciais e ter instalado o Node.js e o AWS CDK Toolkit. Consulte Começando com o AWS CDK.

AWS CDK Os aplicativos Java exigem o Java 8 (v1.8) ou posterior. Recomendamos o Amazon Corretto, mas você pode usar qualquer distribuição do OpenJDK ou o JDK da Oracle. Você também precisará do Apache Maven 3.5 ou posterior. Você também pode usar ferramentas como o Gradle, mas os esqueletos de aplicativos gerados pelo AWS CDK Toolkit são projetos Maven.

> **Note**
>
> Suspensão de uso de idioma de terceiros: a versão do idioma só é suportada até seu EOL (End Of Life) compartilhado pelo fornecedor ou pela comunidade e está sujeita a alterações mediante aviso prévio.

Criação de um projeto

Você cria um novo AWS CDK projeto invocando `cdk init` em um diretório vazio. Use a `--language` opção e especifique`java`:

```
mkdir my-project
cd my-project
```

```
cdk init app --language java
```

`cdk init`usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Os hífens no nome da pasta são convertidos em sublinhados. No entanto, caso contrário, o nome deve seguir a forma de um identificador Java; por exemplo, ele não deve começar com um número ou conter espaços.

O projeto resultante inclui uma referência ao pacote `software.amazon.awscdk` Maven. Ele e suas dependências são instalados automaticamente pelo Maven.

Se você estiver usando um IDE, agora você pode abrir ou importar o projeto. No Eclipse, por exemplo, escolha Arquivo > Importar > Maven > Projetos Maven existentes. Certifique-se de que as configurações do projeto estejam definidas para usar o Java 8 (1.8).

Gerenciando módulos da AWS Construct Libr

Use o Maven para instalar os pacotes da AWS Construct Library, que estão no grupo`software.amazon.awscdk`. A maioria das construções está no artefato`aws-cdk-lib`, que é adicionado aos novos projetos Java por padrão. Os módulos para serviços cujo suporte de CDK de alto nível ainda está sendo desenvolvido estão em pacotes “experimentais” separados, nomeados com uma versão curta (sem ou prefixo da AWS Amazon) do nome do serviço. Pesquise no Repositório Central do Maven para encontrar os nomes de todas as bibliotecas AWS CDK e do AWS Construct Module.

> **Note**
>
> A edição Java da CDK API Reference também mostra os nomes dos pacotes.

O suporte da AWS Construct Library de alguns serviços está em mais de um namespace. Por exemplo, o Amazon Route 53 tem sua funcionalidade dividida em `software.amazon.awscdk.route53` `route53-patterns`,`route53resolver`, `route53-targets` e.

O AWS CDK pacote principal é importado em código Java como`software.amazon.awscdk`. Os módulos dos vários serviços na AWS Construct Library estão abaixo `software.amazon.awscdk.services` e são nomeados de forma semelhante ao nome do pacote Maven. Por exemplo, o namespace do módulo Amazon S3 é. `software.amazon.awscdk.services.s3`

Recomendamos escrever uma `import` instrução Java separada para cada classe da AWS Construct Library usada em cada um dos seus arquivos de origem Java e evitar importações de caracteres curinga. Você sempre pode usar o nome totalmente qualificado de um tipo (incluindo seu namespace) sem uma declaração. `import`

Se seu aplicativo depender de um pacote experimental, edite o do seu projeto `pom.xml` e adicione um novo `<dependency>` elemento no `<dependencies>` contêiner. Por exemplo, o `<dependency>` elemento a seguir especifica o módulo da biblioteca de construção CodeStar experimental:

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>codestar-alpha</artifactId>
    <version>2.0.0-alpha.10</version>
</dependency>
```

> **Tip**
>
> Se você usa um IDE Java, ele provavelmente tem recursos para gerenciar dependências do Maven. No entanto, recomendamos editar `pom.xml` diretamente, a menos que você tenha certeza absoluta de que a funcionalidade do IDE corresponde ao que você faria manualmente.

Gerenciando dependências em Java

Em Java, as dependências são especificadas `pom.xml` e instaladas usando o Maven. O `<dependencies>` contêiner inclui um `<dependency>` elemento para cada pacote. A seguir está uma seção `pom.xml` de um aplicativo Java típico do CDK.

```
<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**
>
> Muitos IDEs Java têm suporte integrado ao Maven e `pom.xml` editores visuais, que podem ser convenientes para gerenciar dependências.

O Maven não oferece suporte ao bloqueio de dependências. Embora seja possível especificar intervalos de versões`pom.xml`, recomendamos que você sempre use versões exatas para manter suas compilações repetíveis.

O Maven instala automaticamente dependências transitivas, mas só pode haver uma cópia instalada de cada pacote. A versão mais alta especificada na árvore POM é selecionada; os aplicativos sempre têm a última palavra em qual versão dos pacotes serão instalados.

O Maven instala ou atualiza automaticamente suas dependências sempre que você constrói (mvn compile) ou empacota (mvn package) seu projeto. O CDK Toolkit faz isso automaticamente toda vez que você o executa, então geralmente não há necessidade de invocar manualmente o Maven.

AWS CDK expressões idiomáticas em Java

Adereços

Todas as classes da AWS Construct Library são instanciadas usando três argumentos: o escopo no qual a construção está sendo definida (seu pai na árvore de construção), um id e props, um pacote de pares de chave/valor que a construção usa para configurar os recursos que cria. Outras classes e métodos também usam o padrão “pacote de atributos” para argumentos.

Em Java, os adereços são expressos usando o padrão Builder. Cada tipo de construção tem um tipo de props correspondente; por exemplo, a `Bucket` construção (que representa um bucket do Amazon S3) usa como props uma instância de. `BucketProps`

A `BucketProps` classe (como toda classe de adereços da AWS Construct Library) tem uma classe interna chamada`Builder`. O `BucketProps.Builder` tipo oferece métodos para definir as várias propriedades de uma `BucketProps` instância. Cada método retorna a `Builder` instância, para que as chamadas do método possam ser encadeadas para definir várias propriedades. No final da cadeia, você liga `build()` para realmente produzir o `BucketProps` objeto.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build());
```

Construções e outras classes que usam um objeto semelhante a adereços como argumento final oferecem um atalho. A classe tem uma `Builder` própria que instancia ela e seu objeto de adereços em uma única etapa. Dessa forma, você não precisa instanciar explicitamente (por exemplo) ambos `BucketProps` e a `Bucket` — e não precisa de uma importação para o tipo props.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build();
```

Ao derivar sua própria construção de uma construção existente, talvez você queira aceitar propriedades adicionais. Recomendamos que você siga esses padrões do construtor. No entanto, isso não é tão simples quanto criar uma subclasse de uma classe de construção. Você mesmo deve fornecer as partes móveis das duas novas `Builder` classes. Você pode preferir simplesmente que sua construção aceite um ou mais argumentos adicionais. Você deve fornecer construtores adicionais quando um argumento for opcional.

Estruturas genéricas

Em algumas APIs, o AWS CDK usa JavaScript matrizes ou objetos não digitados como entrada para um método. (Veja, por exemplo, AWS CodeBuild o `BuildSpec.fromObject()`método de.) Em Java, esses objetos são representados com`java.util.Map<String, Object>`. Nos casos em que os valores são todos cadeias de caracteres, você pode usar`Map<String, String>`.

O Java não fornece uma maneira de escrever literais para esses contêineres, como fazem algumas outras linguagens. No Java 9 e versões posteriores, você pode usar `java.util.Map.of()`para definir convenientemente mapas de até dez entradas em linha com uma dessas chamadas.

```
java.util.Map.of(
    "base-directory", "dist",
    "files", "LambdaStack.template.json"
 )
```

Para criar mapas com mais de dez entradas, use `java.util.Map.ofEntries()`.

Se você estiver usando o Java 8, poderá fornecer seus próprios métodos semelhantes a esses.

JavaScript matrizes são representadas como `List<Object>` ou `List<String>` em Java. O método `java.util.Arrays.asList` é conveniente para definir `List` s curtos.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

Valores ausentes

Em Java, valores ausentes em AWS CDK objetos como adereços são representados por`null`. Você deve testar explicitamente qualquer valor que possa existir `null` para garantir que ele contenha um valor antes de fazer qualquer coisa com ele. Java não tem "açúcar sintático" para ajudar a lidar com valores nulos, como algumas outras linguagens. Você pode achar o Apache ObjectUtil firstNonNullútil em algumas situações. defaultIfNull Como alternativa, escreva seus próprios métodos auxiliares estáticos para facilitar o tratamento de valores potencialmente nulos e tornar seu código mais legível.

Construindo, sintetizando e implantando

O compila AWS CDK automaticamente seu aplicativo antes de executá-lo. No entanto, pode ser útil criar seu aplicativo manualmente para verificar erros e executar testes. Você pode fazer isso em seu IDE (por exemplo, pressione Control-B no Eclipse) ou emitindo em um prompt de comando enquanto estiver `mvn compile` no diretório raiz do seu projeto.

Execute todos os testes que você escreveu executando `mvn test` em um prompt de comando.

As pilhas definidas em seu AWS CDK aplicativo podem ser sintetizadas e implantadas individualmente ou em conjunto usando os comandos abaixo. Geralmente, você deve estar no diretório principal do seu projeto ao emiti-los.

- `cdk synth`: sintetiza um AWS CloudFormation modelo a partir de uma ou mais pilhas do seu aplicativo. AWS CDK
- `cdk deploy`: implanta os recursos definidos por uma ou mais pilhas em seu AWS CDK aplicativo para. AWS

Você pode especificar os nomes de várias pilhas a serem sintetizadas ou implantadas em um único comando. Se seu aplicativo definir apenas uma pilha, você não precisará especificá-la.

```
cdk synth                # app defines single stack
```

```
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Você também pode usar os curingas * (qualquer número de caracteres) e? (qualquer caractere único) para identificar pilhas por padrão. Ao usar curingas, coloque o padrão entre aspas. Caso contrário, o shell pode tentar expandi-lo para os nomes dos arquivos no diretório atual antes de serem passados para o AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Você não precisa sintetizar explicitamente as pilhas antes de implantá-las; `cdk deploy` executa essa etapa para garantir que seu código mais recente seja implantado.

Para obter a documentação completa do `cdk` comando, consultethe section called “AWS CDK Kit de ferramentas”.

Trabalhando com o AWS CDK em C#

.NET é uma linguagem de cliente totalmente compatível com o AWS CDK e é considerada estável. C# é a principal linguagem.NET para a qual fornecemos exemplos e suporte. Você pode optar por escrever AWS CDK aplicativos em outras linguagens.NET, como Visual Basic ou F#, mas AWS oferece suporte limitado para o uso dessas linguagens com o CDK.

Você pode desenvolver AWS CDK aplicativos em C# usando ferramentas conhecidas, incluindo Visual Studio, Visual Studio Code, o `dotnet` comando e o gerenciador de NuGet pacotes. Os módulos que compõem a AWS Construct Library são distribuídos via nuget.org.

Sugerimos usar o Visual Studio 2019 (qualquer edição) no Windows para desenvolver AWS CDK aplicativos em C#.

Tópicos

- Conceitos básicos do C#
- Criação de um projeto
- Gerenciando módulos da AWS Construct Libr

- Gerenciando dependências em C#
- AWS CDK expressões idiomáticas em C#
- Construindo, sintetizando e implantando

Conceitos básicos do C#

Para trabalhar com o AWS CDK, você deve ter uma AWS conta e credenciais e ter instalado o Node.js e o AWS CDK Toolkit. Consulte Começando com o AWS CDK.

Os AWS CDK aplicativos C# exigem o.NET Core v3.1 ou posterior, disponível aqui.

O conjunto de ferramentas do.NET inclui `dotnet` uma ferramenta de linha de comando para criar e executar aplicativos.NET e gerenciar pacotes. NuGet Mesmo que você trabalhe principalmente no Visual Studio, esse comando pode ser útil para operações em lote e para instalar pacotes da AWS Construct Library.

Criação de um projeto

Você cria um novo AWS CDK projeto invocando `cdk init` em um diretório vazio. Use a `--language` opção e especifique`csharp`:

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init`usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Os hífens no nome da pasta são convertidos em sublinhados. No entanto, caso contrário, o nome deve seguir a forma de um identificador C#; por exemplo, ele não deve começar com um número ou conter espaços.

O projeto resultante inclui uma referência ao `Amazon.CDK.Lib` NuGet pacote. Ele e suas dependências são instalados automaticamente pelo NuGet.

Gerenciando módulos da AWS Construct Libr

O ecossistema.NET usa o gerenciador de NuGet pacotes. O pacote CDK principal, que contém as classes principais e todas as construções de serviço estáveis, é. `Amazon.CDK.Lib` Os módulos

experimentais, nos quais a nova funcionalidade está em desenvolvimento ativo, são nomeados como`Amazon.CDK.AWS.SERVICE-NAME.Alpha`, em que o nome do serviço é um nome curto sem um prefixo AWS ou da Amazon. Por exemplo, o nome do NuGet pacote do AWS IoT módulo é`Amazon.CDK.AWS.IoT.Alpha`. Se você não conseguir encontrar o pacote desejado, pesquise NuGet.org.

> **Note**
>
> A edição.NET da CDK API Reference também mostra os nomes dos pacotes.

O suporte da AWS Construct Library de alguns serviços está em mais de um módulo. Por exemplo, AWS IoT tem um segundo módulo chamado`Amazon.CDK.AWS.IoT.Actions.Alpha`.

O módulo principal AWS CDK do, que você precisará na maioria dos AWS CDK aplicativos, é importado no código C# como`Amazon.CDK`. Os módulos para os vários serviços da AWS Construct Library estão abaixo`Amazon.CDK.AWS`. Por exemplo, o namespace do módulo Amazon S3 é. `Amazon.CDK.AWS.S3`

Recomendamos escrever `using` diretivas em C# para as construções principais do CDK e para cada AWS serviço que você usa em cada um dos seus arquivos de origem em C#. Talvez seja conveniente usar um alias para um namespace ou tipo para ajudar a resolver conflitos de nomes. Você sempre pode usar o nome totalmente qualificado de um tipo (incluindo seu namespace) sem uma declaração. `using`

Gerenciando dependências em C#

Em AWS CDK aplicativos C#, você gerencia dependências usando. NuGet NuGet tem quatro interfaces padrão, em sua maioria equivalentes. Use aquele que se adapte às suas necessidades e estilo de trabalho. Você também pode usar ferramentas compatíveis, como o Paket MyGetou até mesmo editar o `.csproj` arquivo diretamente.

NuGet não permite que você especifique intervalos de versão para dependências. Cada dependência é fixada em uma versão específica.

Depois de atualizar suas dependências, o Visual Studio usará NuGet para recuperar as versões especificadas de cada pacote na próxima vez que você criar. Se você não estiver usando o Visual Studio, use o dotnet restore comando para atualizar suas dependências.

Editando o arquivo do projeto diretamente

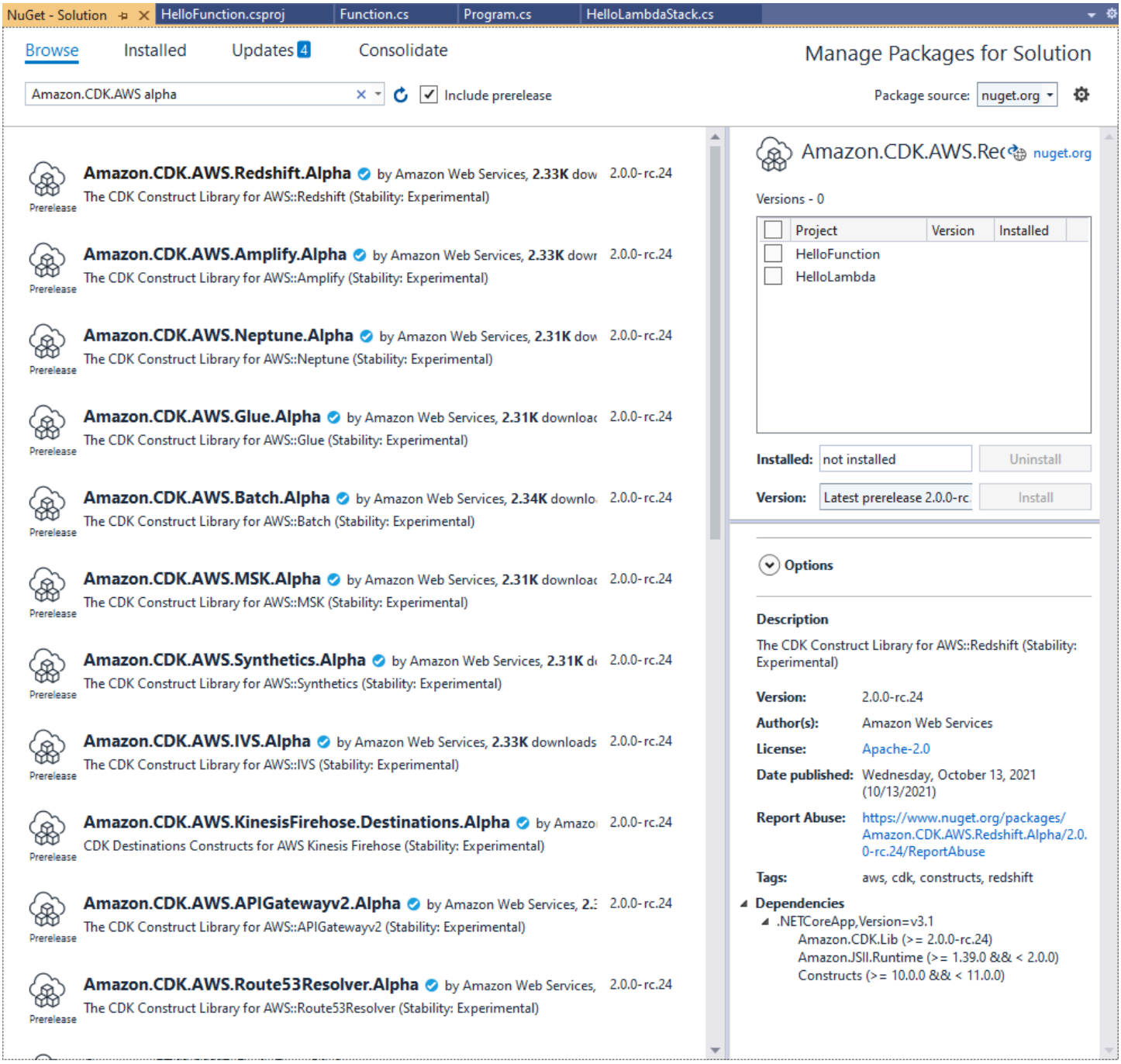
O `.csproj` arquivo do seu projeto contém um `<ItemGroup>` contêiner que lista suas dependências como `<PackageReference` elementos.

```
<ItemGroup>
    <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
    <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

A NuGet interface gráfica do Visual Studio

As NuGet ferramentas do Visual Studio podem ser acessadas em Tools > NuGet Package Manager > Manage NuGet Packages for Solution. Use a guia Procurar para encontrar os pacotes da AWS Construct Library que você deseja instalar. Você pode escolher a versão desejada, incluindo as versões de pré-lançamento de seus módulos, e adicioná-las a qualquer um dos projetos abertos.

> **Note**
>
> Todos os módulos da AWS Construct Library considerados “experimentais” (consultethe section called “Versionamento”) estão marcados como pré-lançamento em NuGet e têm um sufixo de nome. `alpha`



Consulte a página Atualizações para instalar novas versões dos seus pacotes.

O NuGet console

O NuGet console é uma interface PowerShell baseada NuGet que funciona no contexto de um projeto do Visual Studio. Você pode abri-lo no Visual Studio escolhendo Tools > NuGet Package Manager > Package Manager Console. Para obter mais informações sobre o uso dessa ferramenta, consulte Instalar e gerenciar pacotes com o console do Package Manager no Visual Studio.

O **dotnet** comando

O `dotnet` comando é a principal ferramenta de linha de comando para trabalhar com projetos do Visual Studio C#. Você pode invocá-lo em qualquer prompt de comando do Windows. Entre seus muitos recursos, `dotnet` pode adicionar NuGet dependências a um projeto do Visual Studio.

Supondo que você esteja no mesmo diretório do arquivo de projeto (`.csproj`) do Visual Studio, emita um comando como o seguinte para instalar um pacote. Como a biblioteca principal do CDK está incluída quando você cria um projeto, você só precisa instalar explicitamente os módulos experimentais. Os módulos experimentais exigem que você especifique um número de versão explícito.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Você pode emitir o comando de outro diretório. Para fazer isso, inclua o caminho para o arquivo do projeto ou para o diretório que o contém após a `add` palavra-chave. O exemplo a seguir pressupõe que você esteja no diretório principal do seu AWS CDK projeto.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Para instalar uma versão específica de um pacote, inclua o `-v` sinalizador e a versão desejada.

Para atualizar um pacote, emita o mesmo `dotnet add` comando usado para instalá-lo. Para módulos experimentais, novamente, você deve especificar um número de versão explícito.

Para obter mais informações sobre como gerenciar pacotes usando o `dotnet` comando, consulte Instalar e gerenciar pacotes usando a CLI dotnet.

O **nuget** comando

A ferramenta de linha de `nuget` comando pode instalar e atualizar NuGet pacotes. No entanto, isso exige que seu projeto do Visual Studio seja configurado de forma diferente da forma como `cdk init` configura projetos. (Detalhes técnicos: `nuget` trabalha com `Packages.config` projetos e `cdk init` cria um projeto de estilo mais novo`PackageReference`.)

Não recomendamos o uso da `nuget` ferramenta com AWS CDK projetos criados por`cdk init`. Se você estiver usando outro tipo de projeto e quiser usá-lo`nuget`, consulte a Referência da NuGet CLI.

AWS CDK expressões idiomáticas em C#

Adereços

Todas as classes da AWS Construct Library são instanciadas usando três argumentos: o escopo no qual a construção está sendo definida (seu pai na árvore de construção), um id e props, um pacote de pares de chave/valor que a construção usa para configurar os recursos que cria. Outras classes e métodos também usam o padrão “pacote de atributos” para argumentos.

Em C#, adereços são expressos usando um tipo de adereços. No estilo idiomático do C#, podemos usar um inicializador de objetos para definir as várias propriedades. Aqui, estamos criando um bucket do Amazon S3 usando a `Bucket` construção; seu tipo de props correspondente é. `BucketProps`

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
    Versioned = true
});
```

> **Tip**
>
> Adicione o pacote `Amazon.JSII.Analyzers` ao seu projeto para obter os valores necessários verificando suas definições de props dentro do Visual Studio.

Ao estender uma classe ou substituir um método, talvez você queira aceitar adereços adicionais para seus próprios propósitos que não sejam compreendidos pela classe principal. Para fazer isso, subclassifique o tipo de props apropriado e adicione os novos atributos.

```
// 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"
});
```

Ao chamar o inicializador ou o método substituído da classe principal, geralmente você pode passar os adereços que recebeu. O novo tipo é compatível com seu pai, e os adereços extras adicionados são ignorados.

Uma versão futura do AWS CDK poderia coincidentemente adicionar uma nova propriedade com um nome que você usou para sua própria propriedade. Isso não causará problemas técnicos ao usar sua construção ou método (como sua propriedade não é passada "para cima na cadeia", a classe principal ou o método substituído simplesmente usará um valor padrão), mas pode causar confusão para os usuários da sua construção. Você pode evitar esse problema em potencial nomeando suas propriedades de forma que elas claramente pertençam à sua construção. Se houver muitas propriedades novas, agrupe-as em uma classe com nome apropriado e passe-as como uma única propriedade.

Estruturas genéricas

Em algumas APIs, o AWS CDK usa JavaScript matrizes ou objetos não digitados como entrada para um método. (Veja, por exemplo, AWS CodeBuild o `BuildSpec.fromObject()`método de.) Em C#, esses objetos são representados como`System.Collections.Generic.Dictionary<String, Object>`. Nos casos em que os valores são todos cadeias de caracteres, você pode usar`Dictionary<String, String>`. JavaScript matrizes são representadas como `object[]` ou tipos `string[]` de matriz em C#.

> **Tip**
>
> Você pode definir aliases curtos para facilitar o trabalho com esses tipos específicos de dicionário.
>
> ```
> using StringDict = System.Collections.Generic.Dictionary<string, string>;
> using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
> ```

Valores ausentes

Em C#, valores ausentes em AWS CDK objetos como adereços são representados por. `null` O operador de acesso de membro condicional nulo `?.` e o operador de coalescência nula são convenientes para trabalhar com esses valores. `??`

```
// 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";
```

Construindo, sintetizando e implantando

O compila AWS CDK automaticamente seu aplicativo antes de executá-lo. No entanto, pode ser útil criar seu aplicativo manualmente para verificar erros e executar testes. Você pode fazer isso pressionando F6 no Visual Studio ou emitindo a `dotnet build src` partir da linha de comando, onde `src` está o diretório no diretório do projeto que contém o arquivo Visual Studio Solution (`.sln`).

As pilhas definidas em seu AWS CDK aplicativo podem ser sintetizadas e implantadas individualmente ou em conjunto usando os comandos abaixo. Geralmente, você deve estar no diretório principal do seu projeto ao emiti-los.

- `cdk synth`: sintetiza um AWS CloudFormation modelo a partir de uma ou mais pilhas do seu aplicativo. AWS CDK
- `cdk deploy`: implanta os recursos definidos por uma ou mais pilhas em seu AWS CDK aplicativo para. AWS

Você pode especificar os nomes de várias pilhas a serem sintetizadas ou implantadas em um único comando. Se seu aplicativo definir apenas uma pilha, você não precisará especificá-la.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

Você também pode usar os curingas * (qualquer número de caracteres) e? (qualquer caractere único) para identificar pilhas por padrão. Ao usar curingas, coloque o padrão entre aspas. Caso contrário, o shell pode tentar expandi-lo para os nomes dos arquivos no diretório atual antes de serem passados para o AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Você não precisa sintetizar explicitamente as pilhas antes de implantá-las; `cdk deploy` executa essa etapa para garantir que seu código mais recente seja implantado.

Para obter a documentação completa do `cdk` comando, consultethe section called “AWS CDK Kit de ferramentas”.

Trabalhando com o AWS CDK in Go

Go é uma linguagem de cliente totalmente compatível com o AWS Cloud Development Kit (AWS CDK) e é considerada estável. Trabalhar com o AWS CDK in Go usa ferramentas familiares. A versão Go do AWS CDK evento usa identificadores no estilo Go.

Ao contrário das outras linguagens suportadas pelo CDK, o Go não é uma linguagem de programação tradicional orientada a objetos. O Go usa composição em que outras linguagens geralmente aproveitam a herança. Tentamos empregar abordagens idiomáticas de Go o máximo possível, mas há lugares em que o CDK pode ser diferente.

Este tópico fornece orientação ao trabalhar com o AWS CDK in Go. Veja a postagem do blog do anúncio para ver um passo a passo de um projeto Go simples para o. AWS CDK

Tópicos

- Conceitos básicos do Go
- Criação de um projeto
- Gerenciando módulos da AWS Construct Libr
- Gerenciando dependências em Go
- AWS CDK expressões idiomáticas em Go
- Construindo, sintetizando e implantando

Conceitos básicos do Go

Para trabalhar com o AWS CDK, você deve ter uma AWS conta e credenciais e ter instalado o Node.js e o AWS CDK Toolkit. Consulte Começando com o AWS CDK.

As vinculações Go para o AWS CDK usam o conjunto de ferramentas Go padrão, v1.18 ou posterior. Você pode usar o editor de sua escolha.

> **Note**
>
> Suspensão de uso de idioma de terceiros: a versão do idioma só é suportada até seu EOL (End Of Life) compartilhado pelo fornecedor ou pela comunidade e está sujeita a alterações mediante aviso prévio.

Criação de um projeto

Você cria um novo AWS CDK projeto invocando `cdk init` em um diretório vazio. Use a `--language` opção e especifiquego:

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init`usa o nome da pasta do projeto para nomear vários elementos do projeto, incluindo classes, subpastas e arquivos. Os hífens no nome da pasta são convertidos em sublinhados. No entanto, caso contrário, o nome deve seguir a forma de um identificador Go; por exemplo, ele não deve começar com um número ou conter espaços.

O projeto resultante inclui uma referência ao módulo AWS CDK Go principal,`github.com/aws/aws-cdk-go/awscdk/v2`, em`go.mod`. Problema go get para instalar este e outros módulos necessários.

Gerenciando módulos da AWS Construct Libr

Na maioria das AWS CDK documentações e exemplos, a palavra “módulo” é frequentemente usada para se referir aos módulos da AWS Construct Library, um ou mais por AWS serviço, o que difere do uso idiomático do termo em Go. A CDK Construct Library é fornecida em um módulo Go com os

módulos individuais da Construct Library, que suportam os vários AWS serviços, fornecidos como pacotes Go dentro desse módulo.

O suporte da AWS Construct Library de alguns serviços está em mais de um módulo da Construct Library (pacote Go). Por exemplo, o Amazon Route 53 tem três módulos da Construct Library, além `awsroute53` do pacote principal `awsroute53patternsawsroute53resolver`, chamado, `awsroute53targets` e.

O AWS CDK pacote principal, que você precisará na maioria dos AWS CDK aplicativos, é importado no código Go como`github.com/aws/aws-cdk-go/awscdk/v2`. Os pacotes para os vários serviços da AWS Construct Library estão abaixo`github.com/aws/aws-cdk-go/awscdk/v2`. Por exemplo, o namespace do módulo Amazon S3 é. `github.com/aws/aws-cdk-go/awscdk/v2/awss3`

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

Depois de importar os módulos da Construct Library (pacotes Go) para os serviços que você deseja usar em seu aplicativo, você acessa as construções desse módulo usando, por exemplo,`awss3.Bucket`.

Gerenciando dependências em Go

Em Go, as versões das dependências são definidas em`go.mod`. O padrão `go.mod` é semelhante ao mostrado aqui.

```
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
)
```

Os nomes dos pacotes (módulos, no jargão do Go) são especificados por URL com o número de versão necessário anexado. O sistema de módulos do Go não suporta intervalos de versões.

go getExecute o comando para instalar todos os módulos necessários e atualizargo.mod. Para ver uma lista das atualizações disponíveis para suas dependências, emitago list -m -u all.

AWS CDK expressões idiomáticas em Go

Nomes de campos e métodos

Os nomes de campos e métodos usam camel casing (`likeThis`) in TypeScript, o idioma de origem do CDK. Em Go, eles seguem as convenções de Go, assim como Pascal-cased (). `LikeThis`

Limpeza

Em seu `main` método, use `defer jsii.Close()` para garantir que seu aplicativo CDK se limpe sozinho.

Valores ausentes e conversão de ponteiro

Em Go, os valores ausentes em AWS CDK objetos, como pacotes de propriedades, são representados por`nil`. Go não tem tipos anuláveis; o único tipo que pode conter `nil` é um ponteiro. Para permitir que os valores sejam opcionais, então, todas as propriedades, argumentos e valores de retorno do CDK são ponteiros, mesmo para tipos primitivos. Isso se aplica tanto aos valores obrigatórios quanto aos opcionais, portanto, se um valor obrigatório se tornar opcional posteriormente, nenhuma alteração significativa no tipo será necessária.

Ao passar valores ou expressões literais, use as seguintes funções auxiliares para criar ponteiros para os valores.

- `jsii.String`
- `jsii.Number`
- `jsii.Bool`
- `jsii.Time`

Para fins de consistência, recomendamos que você use ponteiros de forma semelhante ao definir suas próprias construções, mesmo que pareça mais conveniente, por exemplo, receber suas construções `id` como uma string em vez de um ponteiro para uma string.

Ao lidar com AWS CDK valores opcionais, incluindo valores primitivos e tipos complexos, você deve testar explicitamente os ponteiros para garantir que não estejam `nil` antes de fazer nada com eles. O Go não tem "açúcar sintático" para ajudar a lidar com valores vazios ou ausentes, como algumas outras linguagens. No entanto, é garantido que os valores necessários em pacotes de propriedades e estruturas similares existam (caso contrário, a construção falhará), portanto, esses valores não precisam ser verificados`nil`.

Construções e adereços

As construções, que representam um ou mais AWS recursos e seus atributos associados, são representadas em Go como interfaces. Por exemplo, `awss3.Bucket` é uma interface. Cada construção tem uma função de fábrica`awss3.NewBucket`, como retornar uma estrutura que implementa a interface correspondente.

Todas as funções de fábrica usam três argumentos: o `scope` no qual a construção está sendo definida (seu pai na árvore de construção)`id`, an e `props` um pacote de pares de chave/valor que a construção usa para configurar os recursos que cria. O padrão "pacote de atributos" também é usado em outras partes do AWS CDK.

Em Go, os adereços são representados por um tipo de estrutura específico para cada construção. Por exemplo, an `awss3.Bucket` usa um argumento props do tipo`awss3.BucketProps`. Use uma estrutura literal para escrever argumentos de adereços.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

Estruturas genéricas

Em alguns lugares, o AWS CDK usa JavaScript matrizes ou objetos não digitados como entrada para um método. (Veja, por exemplo, AWS CodeBuild o `BuildSpec.fromObject()`método de.) Em Go, esses objetos são representados como fatias e uma interface vazia, respectivamente.

O CDK fornece funções auxiliares variáveis, como `jsii.Strings` para criar fatias contendo tipos primitivos.

```
jsii.Strings("One", "Two", "Three")
```

Desenvolvendo construções personalizadas

Em Go, geralmente é mais simples escrever uma nova construção do que estender uma existente. Primeiro, defina um novo tipo de estrutura, incorporando anonimamente um ou mais tipos existentes se uma semântica semelhante à extensão for desejada. Escreva métodos para qualquer nova funcionalidade que você esteja adicionando e os campos necessários para armazenar os dados necessários. Defina uma interface de adereços se sua construção precisar de uma. Por fim, escreva uma função de fábrica `NewMyConstruct()` para retornar uma instância da sua construção.

Se você está simplesmente alterando alguns valores padrão em uma construção existente ou adicionando um comportamento simples na instanciação, você não precisa de todo esse encanamento. Em vez disso, escreva uma função de fábrica que chame a função de fábrica da construção que você está "estendendo". Em outras linguagens de CDK, por exemplo, você pode criar uma `TypedBucket` construção que imponha o tipo de objeto em um bucket do Amazon S3 substituindo `s3.Bucket` o tipo e, no inicializador do seu novo tipo, adicionando uma política de bucket que permite que somente extensões de nome de arquivo especificadas sejam adicionadas ao bucket. Em Go, é mais fácil simplesmente escrever um `NewTypedBucket` que retorne um `s3.Bucket` (uso instanciados`s3.NewBucket`) ao qual você adicionou uma política de bucket apropriada. Nenhum novo tipo de construção é necessário porque a funcionalidade já está disponível na construção padrão do bucket; a nova "construção" fornece apenas uma maneira mais simples de configurá-la.

Construindo, sintetizando e implantando

Ele compila AWS CDK automaticamente seu aplicativo antes de executá-lo. No entanto, pode ser útil criar seu aplicativo manualmente para verificar erros e executar testes. Você pode fazer isso emitindo `go build` em um prompt de comando enquanto estiver no diretório raiz do seu projeto.

Execute todos os testes que você escreveu executando `go test` em um prompt de comando.

As pilhas definidas em seu AWS CDK aplicativo podem ser sintetizadas e implantadas individualmente ou em conjunto usando os comandos abaixo. Geralmente, você deve estar no diretório principal do seu projeto ao emiti-los.

- `cdk synth`: sintetiza um AWS CloudFormation modelo a partir de uma ou mais pilhas do seu aplicativo. AWS CDK
- `cdk deploy`: implanta os recursos definidos por uma ou mais pilhas em seu AWS CDK aplicativo para. AWS

Você pode especificar os nomes de várias pilhas a serem sintetizadas ou implantadas em um único comando. Se seu aplicativo definir apenas uma pilha, você não precisará especificá-la.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Você também pode usar os curingas * (qualquer número de caracteres) e? (qualquer caractere único) para identificar pilhas por padrão. Ao usar curingas, coloque o padrão entre aspas. Caso contrário, o shell pode tentar expandi-lo para os nomes dos arquivos no diretório atual antes de serem passados para o AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Você não precisa sintetizar explicitamente as pilhas antes de implantá-las; `cdk deploy` executa essa etapa para garantir que seu código mais recente seja implantado.

Para obter a documentação completa do `cdk` comando, consultethe section called “AWS CDK Kit de ferramentas”.

Desenvolvendo AWS CDK aplicativos

Desenvolva AWS Cloud Development Kit (AWS CDK) aplicativos.

Tópicos

Personalizando construções da Construct Library AWS

Personalize construções da AWS Construct Library por meio de escotilhas de escape, substituições brutas e recursos personalizados.

Tópicos

Usando escotilhas de escape

A AWS Construct Library fornece construções de vários níveis de abstração.

No nível mais alto, seu AWS CDK aplicativo e as pilhas nele contidas são, em si, abstrações de toda a sua infraestrutura de nuvem ou de partes significativas dela. Eles podem ser parametrizados para implantá-los em diferentes ambientes ou para diferentes necessidades.

As abstrações são ferramentas poderosas para projetar e implementar aplicativos em nuvem. AWS CDK Isso lhe dá o poder não apenas de construir com suas abstrações, mas também de criar novas abstrações. Usando as construções L2 e L3 de código aberto existentes como orientação, você pode criar suas próprias construções L2 e L3 para refletir as melhores práticas e opiniões de sua própria organização.

Nenhuma abstração é perfeita, e mesmo boas abstrações não podem cobrir todos os casos de uso possíveis. Durante o desenvolvimento, você pode encontrar uma construção que quase atenda às suas necessidades, exigindo uma personalização pequena ou grande.

Por esse motivo, AWS CDK fornece maneiras de sair do modelo de construção. Isso inclui mudar para uma abstração de nível inferior ou para um modelo totalmente diferente. As escotilhas de escape permitem que você escape do AWS CDK paradigma e o personalize de forma que atenda às suas necessidades. Em seguida, você pode agrupar suas alterações em uma nova construção para abstrair a complexidade subjacente e fornecer uma API limpa para outros desenvolvedores.

Veja a seguir exemplos de situações em que você pode usar escotilhas de escape:

- Um recurso AWS de serviço está disponível por meio AWS CloudFormation dele, mas não há construções L2 para ele.
- Um recurso AWS de serviço está disponível por meio de AWS CloudFormation, e há construções L2 para o serviço, mas elas ainda não expõem o recurso. Como as construções L2 são selecionadas pela equipe do CDK, elas podem não estar imediatamente disponíveis para novos recursos.
- O recurso ainda não está disponível por meio AWS CloudFormation de nada.

  Para determinar se um recurso está disponível por meio de AWS CloudFormation, consulte Referência de tipos de AWS recursos e propriedades.

Desenvolva escotilhas de escape para construções L1

Se as construções L2 não estiverem disponíveis para o serviço, você poderá usar as construções L1 geradas automaticamente. Esses recursos podem ser reconhecidos pelo nome que começa com`Cfn`, como `CfnBucket` ou`CfnRole`. Você os instancia exatamente como usaria o recurso equivalente AWS CloudFormation .

Por exemplo, para instanciar um bucket L1 de baixo nível do Amazon S3 com análises ativadas, você escreveria algo como o seguinte.

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",
        // ...
    }
});
```

Pode haver casos raros em que você queira definir um recurso que não tenha uma `CfnXxx` classe correspondente. Esse pode ser um novo tipo de recurso que ainda não foi publicado na especificação do AWS CloudFormation recurso. Em casos como esse, você pode instanciar o `cdk.CfnResource` diretamente e especificar o tipo e as propriedades do recurso. Isso é mostrado no exemplo a seguir.

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"
            }
        }
    }
```

```
});
```

Desenvolva escotilhas de escape para construções L2

Se uma construção L2 não tiver um recurso ou você estiver tentando contornar um problema, você pode modificar a construção L1 que está encapsulada pela construção L2.

Todas as construções L2 contêm dentro delas a construção L1 correspondente. Por exemplo, a construção de alto nível envolve a `Bucket` construção de baixo `CfnBucket` nível. Como `CfnBucket` corresponde diretamente ao AWS CloudFormation recurso, ele expõe todos os recursos que estão disponíveis por meio AWS CloudFormation de.

A abordagem básica para obter acesso à construção L1 é usar `construct.node.defaultChild` (Python`default_child`:), convertê-la para o tipo certo (se necessário) e modificar suas propriedades. Novamente, vamos dar o exemplo de um`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",
        // ...
    }
};
```

Você também pode usar esse objeto para alterar AWS CloudFormation opções como `Metadata` `UpdatePolicy` e.

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"
};
```

Escotilhas anti-fuga

Isso AWS CDK também fornece a capacidade de subir um nível de abstração, que podemos chamar de escotilha de “saída”. Se você tiver uma construção L1, como`CfnBucket`, você pode criar uma nova construção L2 (`Bucket`nesse caso) para encapsular a construção L1.

Isso é conveniente quando você cria um recurso L1, mas deseja usá-lo com uma construção que requer um recurso L2. Também é útil quando você deseja usar métodos de conveniência como esses `.grantXxxx()` que não estão disponíveis na construção L1.

Você passa para o nível de abstração mais alto usando um método estático na classe L2 chamado `.fromCfnXxxxx()` —por exemplo, para buckets do Amazon `Bucket.fromCfnBucket()` S3. O recurso L1 é o único parâmetro.

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);
```

As construções L2 criadas a partir de construções L1 são objetos proxy que se referem ao recurso L1, semelhantes às criadas a partir de nomes de recursos, ARNs ou pesquisas. As modificações nessas construções não afetam o AWS CloudFormation modelo final sintetizado (já que você tem o recurso L1, no entanto, você pode modificá-lo). Para obter mais informações sobre objetos proxy, consultethe section called “Referenciando recursos em sua conta AWS”.

Para evitar confusão, não crie várias construções L2 que se refiram à mesma construção L1. Por exemplo, se você extrair o `CfnBucket` de a `Bucket` usando a técnica da seção anterior, você

não deve criar uma segunda `Bucket` instância chamando `Bucket.fromCfnBucket()` com isso`CfnBucket`. Na verdade, funciona conforme o esperado (apenas um `AWS::S3::Bucket` é sintetizado), mas dificulta a manutenção do código.

Substituições brutas

Se faltarem propriedades na construção L1, você poderá ignorar toda a digitação usando substituições brutas. Isso também possibilita a exclusão de propriedades sintetizadas.

Use um dos `addOverride` métodos method (Python:`add_override`), conforme mostrado no exemplo a seguir.

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");
```

Recursos personalizados

Se o recurso não estiver disponível por meio de AWS CloudFormation, mas somente por meio de uma chamada direta de API, você deverá escrever um recurso AWS CloudFormation personalizado para fazer a chamada de API necessária. Você pode usar o AWS CDK para escrever recursos personalizados e agrupá-los em uma interface de construção regular. Do ponto de vista de um consumidor de sua marca, a experiência parecerá nativa.

A criação de um recurso personalizado envolve escrever uma função Lambda que responda aos eventos do ciclo de vida e `DELETE` do `CREATE` recurso. `UPDATE` Se seu recurso personalizado precisar fazer apenas uma única chamada de API, considere usar AwsCustomResourceo. Isso possibilita a realização de chamadas arbitrárias do SDK durante uma AWS CloudFormation

implantação. Caso contrário, você deve escrever sua própria função Lambda para realizar o trabalho necessário.

O assunto é muito amplo para ser abordado completamente aqui, mas os links a seguir devem ajudar você a começar:

- Recursos personalizados
- Exemplo de recurso personalizado
- Para obter um exemplo mais completo, consulte a DnsValidatedCertificateclasse na biblioteca padrão do CDK. Isso é implementado como um recurso personalizado.

Obter um valor de uma variável de ambiente

Para obter o valor de uma variável de ambiente, use um código como o seguinte. Esse código obtém o valor da variável de ambiente`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";
```

Use um AWS CloudFormation valor

Consulte the section called “Parâmetros” para obter informações sobre o uso de AWS CloudFormation parâmetros com AWS CDK o.

Para obter uma referência a um recurso em um AWS CloudFormation modelo existente, consultethe section called “Importar um AWS CloudFormation modelo”.

Importar um AWS CloudFormation modelo existente

Importe recursos de um AWS CloudFormation modelo para seus AWS Cloud Development Kit (AWS CDK) aplicativos usando a `cloudformation-include.CfnInclude`construção para converter recursos em construções L1.

Após a importação, você pode trabalhar com esses recursos em seu aplicativo da mesma forma que faria se eles fossem originalmente definidos em AWS CDK código. Você também pode usar essas construções L1 em construções de nível superior AWS CDK . Por exemplo, isso pode permitir que você use os métodos de concessão de permissão L2 com os recursos que eles definem.

A `cloudformation-include.CfnInclude` construção basicamente adiciona um wrapper de AWS CDK API a qualquer recurso em seu AWS CloudFormation modelo. Use esse recurso para importar seus AWS CloudFormation modelos existentes para AWS CDK uma peça por vez. Ao fazer isso, você pode gerenciar seus recursos existentes usando AWS CDK construções para utilizar os benefícios das abstrações de alto nível. Você também pode usar esse recurso para vender seus AWS CloudFormation modelos aos AWS CDK desenvolvedores fornecendo uma API de AWS CDK construção.

> **Note**
>
> AWS CDK A v1 também foi incluída `aws-cdk-lib.CfnInclude`, que era usada anteriormente para o mesmo propósito geral. No entanto, ele carece de muitas das funcionalidades do`cloudformation-include.CfnInclude`.

Tópicos

Importando um modelo AWS CloudFormation

Veja a seguir um exemplo AWS CloudFormation de modelo que usaremos para fornecer exemplos neste tópico. Copie e salve o modelo `my-template.json` a seguir. Depois de analisar esses exemplos, você pode explorar mais usando qualquer um dos AWS CloudFormation modelos implantados existentes. Você pode obtê-los no AWS CloudFormation console.

```
{
  "Resources": {
    "MyBucket": {
```

```
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

Você pode trabalhar com modelos JSON ou YAML. Recomendamos JSON, se disponível, pois os analisadores YAML podem variar um pouco no que aceitam.

Veja a seguir um exemplo de como importar o modelo de amostra para seu AWS CDK aplicativo usando `cloudformation-include` o. Os modelos são importados dentro do contexto de uma pilha de 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"
            });
        }
    }
}
```

Por padrão, a importação de um recurso preserva a ID lógica original do recurso no modelo. Esse comportamento é adequado para importar um AWS CloudFormation modelo para o AWS CDK, onde as IDs lógicas devem ser mantidas. AWS CloudFormation precisa dessas informações para reconhecer esses recursos importados como os mesmos recursos do AWS CloudFormation modelo.

Se você estiver desenvolvendo um wrapper de AWS CDK construção para o modelo para que ele possa ser usado por outros AWS CDK desenvolvedores, faça com que eles AWS CDK gerem novos IDs de recursos. Ao fazer isso, a construção pode ser usada várias vezes em uma pilha sem conflitos de nome. Para fazer isso, defina a `preserveLogicalIds` propriedade como `false` ao importar o modelo. Veja um exemplo a seguir:

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
});
```

Para colocar os recursos importados sob o controle do seu AWS CDK aplicativo, adicione a pilha aoApp:

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");
        }
    }
}
```

Para verificar se não haverá nenhuma alteração não intencional nos AWS recursos na pilha, você pode fazer uma comparação. Use o AWS CDK CLI `cdk diff` comando e omita qualquer AWS CDK metadado específico. Veja um exemplo a seguir:

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

Depois de importar um AWS CloudFormation modelo, o AWS CDK aplicativo deve se tornar a fonte confiável dos recursos importados. Para fazer alterações em seus recursos, modifique-os em seu AWS CDK aplicativo e implante com o AWS CDK CLI cdk deploy comando.

Acessando recursos importados

O nome `template` no código de exemplo representa o AWS CloudFormation modelo importado. Para acessar um recurso a partir dele, use o `getResource()`método do objeto. Para acessar o recurso retornado como um tipo específico de recurso, converta o resultado no tipo desejado. Isso não é necessário em Python ou. JavaScript Veja um exemplo a seguir:

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");
```

A partir desse exemplo, agora `cfnBucket` é uma instância da `aws-s3.CfnBucket`classe. Essa é uma construção L1 que representa o AWS CloudFormation recurso correspondente. Você pode tratá-lo como qualquer outro recurso desse tipo. Por exemplo, você pode obter seu valor ARN com a `bucket.attrArn` propriedade.

Em vez disso, para agrupar o `CfnBucket` recurso L1 em uma `aws-s3.Bucket`instância L2, use os métodos estáticos `fromBucketArn()``fromBucketAttributes()`, ou. `fromBucketName()` Normalmente, o `fromBucketName()` método é mais conveniente. Veja um exemplo a seguir:

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);
```

Outras construções L2 têm métodos semelhantes para criar a construção a partir de um recurso existente.

Quando você envolve uma construção L1 em uma construção L2, ela não cria um novo recurso. No nosso exemplo, não estamos criando um segundo bucket S3;. Em vez disso, a nova `Bucket` instância encapsula a existente. `CfnBucket`

A partir do exemplo, agora `bucket` é uma `Bucket` construção L2 que se comporta como qualquer outra construção L2. Por exemplo, você pode conceder a uma AWS Lambda função acesso de gravação ao bucket usando o `grantWrite()`método conveniente do bucket. Você não precisa definir a política necessária AWS Identity and Access Management (IAM) manualmente. Veja um exemplo a seguir:

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

Substituindo parâmetros

Se seu AWS CloudFormation modelo contiver parâmetros, você poderá substituí-los por valores de tempo de construção na importação usando a `parameters` propriedade. No exemplo a seguir, substituímos o `UploadBucket` parâmetro pelo ARN de um bucket definido em outro lugar em nosso AWS CDK código.

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 }
    }
});
```

Outros elementos do modelo

Você pode importar qualquer elemento AWS CloudFormation do modelo, não apenas recursos. Os elementos importados se tornam parte da AWS CDK pilha. Para importar esses elementos, use os seguintes métodos do `CfnInclude` objeto:

- `getCondition()`— AWS CloudFormation condições.
- `getHook()`— AWS CloudFormation ganchos para implantações em azul/verde.

- `getMapping()`— AWS CloudFormation mapeamentos.
- `getOutput()`— AWS CloudFormation saídas.
- `getParameter()`— AWS CloudFormation parâmetros.
- `getRule()`— AWS CloudFormation regras para AWS Service Catalog modelos.

Cada um desses métodos retorna uma instância de uma classe que representa o tipo específico de AWS CloudFormation elemento. Esses objetos são mutáveis. As alterações que você fizer nelas aparecerão no modelo gerado a partir da AWS CDK pilha. Veja a seguir um exemplo que importa um parâmetro do modelo e modifica seu valor padrão:

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";
```

Pilhas aninhadas

Você pode importar pilhas aninhadas especificando-as ao importar o modelo principal ou em algum momento posterior. O modelo aninhado deve ser armazenado em um arquivo local, mas referenciado como um `NestedStack` recurso no modelo principal. Além disso, o nome do recurso usado no AWS CDK código deve corresponder ao nome usado para a pilha aninhada no modelo principal.

Dada essa definição de recurso no modelo principal, o código a seguir mostra como importar a pilha aninhada referenciada nos dois sentidos.

```
"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'
});
```

Você pode importar várias pilhas aninhadas com qualquer um dos métodos. Ao importar o modelo principal, você fornece um mapeamento entre o nome do recurso de cada pilha aninhada e seu arquivo de modelo. Esse mapeamento pode conter qualquer número de entradas. Para fazer isso após a importação inicial, chame `loadNestedStack()` uma vez para cada pilha aninhada.

Depois de importar uma pilha aninhada, você pode acessá-la usando o método do modelo principal. `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;
```

O `getNestedStack()` método retorna uma `IncludedNestedStack`instância. Nessa instância, você pode acessar a AWS CDK `NestedStack`instância por meio da `stack` propriedade, conforme mostrado no exemplo. Você também pode acessar o objeto AWS CloudFormation de modelo original por meio `includedTemplate` do qual você pode carregar recursos e outros AWS CloudFormation elementos.

Obtenha um valor do Systems Manager Parameter Store

Eles AWS Cloud Development Kit (AWS CDK) podem recuperar o valor dos atributos do AWS Systems Manager Parameter Store. Durante a síntese, AWS CDK produz um token que é resolvido AWS CloudFormation durante a implantação.

O AWS CDK suporta a recuperação de valores simples e seguros. Você pode solicitar uma versão específica de qualquer tipo de valor. Para valores simples, você pode omitir a versão da sua solicitação para recuperar a versão mais recente. Para valores seguros, você deve especificar a versão ao solicitar o valor do atributo seguro.

> **Note**
>
> Este tópico mostra como ler atributos do AWS Systems Manager Parameter Store. Você também pode ler os segredos do AWS Secrets Manager (consulteObtenha um valor de AWS Secrets Manager).

Tópicos

- Leia os valores do Systems Manager no momento da implantação
- Leia os valores do Systems Manager no momento da síntese
- Grave valores no Systems Manager

Leia os valores do Systems Manager no momento da implantação

Para ler valores do Systems Manager Parameter Store, use o valueForStringparâmetro e valueForSecureStringParameteros métodos. Escolha um método com base no fato de o atributo que você deseja ser uma string simples ou um valor de string seguro. Esses métodos retornam tokens, não o valor real. O valor é resolvido AWS CloudFormation durante a implantação. Veja um exemplo a seguir:

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
```

Atualmente, um número limitado de AWS serviços oferece suporte a esse recurso.

Leia os valores do Systems Manager no momento da síntese

Às vezes, é útil fornecer um parâmetro no momento da síntese. Ao fazer isso, o AWS CloudFormation modelo sempre usará o mesmo valor em vez de resolver o valor durante a implantação.

Para ler um valor do Systems Manager Parameter Store no momento da síntese, use o valueFromLookupmétodo (Python:`value_from_lookup`). Esse método retorna o valor real do parâmetro como um the section called “Contexto” valor. Se o valor ainda não estiver armazenado em cache `cdk.json` ou passado na linha de comando, ele será recuperado da conta atual AWS . Por esse motivo, a pilha deve ser sintetizada com informações explícitas AWS do ambiente.

Veja um exemplo a seguir:

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");
```

Somente cadeias de caracteres simples do Systems Manager podem ser recuperadas. Cadeias de caracteres seguras não podem ser recuperadas. A versão mais recente sempre será devolvida. Versões específicas não podem ser solicitadas.

> **Important**
>
> O valor recuperado terminará em seu modelo sintetizado AWS CloudFormation . Isso pode ser um risco de segurança, dependendo de quem tem acesso aos seus AWS CloudFormation modelos e do tipo de valor que eles representam. Geralmente, não use esse recurso para senhas, chaves ou outros valores que você queira manter privados.

Grave valores no Systems Manager

Você pode usar a AWS CLI, a ou um AWS SDK para definir os AWS Management Console valores dos parâmetros do Systems Manager. Os exemplos a seguir usam o comando SSM put-parameter CLI.

```
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"
```

Ao atualizar um valor de SSM que já existe, inclua também a `--overwrite` opção.

```
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"
```

Obtenha um valor de AWS Secrets Manager

Para usar valores do AWS Secrets Manager seu AWS CDK aplicativo, use o método fromSecretAttributes(). Ele representa um valor que é recuperado do Secrets Manager e usado no momento da AWS CloudFormation implantação. Veja um exemplo a seguir:

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

Use o comando da CLI AWS CLI create-secret para criar um segredo na linha de comando, como ao testar:

```
aws secretsmanager create-secret --name ImportedSecret --secret-string
 mygroovybucket
```

O comando retorna um ARN que você pode usar com o exemplo anterior.

Depois de criar uma `Secret` instância, você pode obter o valor do segredo a partir do `secretValue` atributo da instância. O valor é representado por uma `SecretValue` instância, um tipo especial dethe section called “Tokens”. Por ser um símbolo, ele só tem significado após a resolução. Seu aplicativo CDK não precisa acessar seu valor real. Em vez disso, o aplicativo pode passar a `SecretValue` instância (ou sua string ou representação numérica) para qualquer método CDK que precise do valor.

Defina um CloudWatch alarme

Use o pacote aws-cloudwatch para configurar alarmes da Amazon CloudWatch em métricas. CloudWatch Você pode usar métricas predefinidas ou criar suas próprias.

Tópicos

Usando uma métrica existente

Muitos módulos da AWS Construct Library permitem que você defina um alarme em uma métrica existente passando o nome da métrica para um método de conveniência em uma instância de um objeto que tem métricas. Por exemplo, considerando uma fila do Amazon SQS, você pode obter a métrica ApproximateNumberOfMessagesVisibledo método metric () da fila:

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");
```

Criando sua própria métrica

Crie sua própria métrica da seguinte forma, em que o valor do namespace deve ser algo como AWS/SQS para uma fila do Amazon SQS. Você também precisa especificar o nome e a dimensão da sua métrica:

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" }
    }
});
```

Criando o alarme

Depois de ter uma métrica, existente ou definida por você, você pode criar um alarme. Neste exemplo, o alarme é acionado quando há mais de 100 de sua métrica em dois dos últimos três períodos de avaliação. Você pode usar comparações como less-than em seus alarmes por meio da propriedade. `comparison0perator` G reater-than-or-equal -to é o AWS CDK padrão, então não precisamos especificá-lo.

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
});
```

Uma forma alternativa de criar um alarme é usar o método createAlarm () da métrica, que usa basicamente as mesmas propriedades do `Alarm` construtor. Você não precisa passar a métrica, porque ela já é conhecida.

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
});
```

Salvar e recuperar valores de variáveis de contexto

Você pode especificar variáveis de contexto com o AWS Cloud Development Kit (AWS CDK) CLI ou no `cdk.json` arquivo. Em seguida, use o `TryGetContext` método para recuperar valores.

Tópicos

Especificar variáveis de contexto

Você pode especificar uma variável de contexto como parte de um AWS CDK CLI comando ou em`cdk.json`.

Para criar uma variável de contexto de linha de comando, use a opção --context (-c), conforme mostrado no exemplo a seguir.

```
cdk synth -c bucket_name=mygroovybucket
```

Para especificar a mesma variável de contexto e valor no `cdk.json` arquivo, use o código a seguir.

```
{
  "context": {
    "bucket_name": "myotherbucket"
  }
}
```

Se você especificar uma variável de contexto usando o `cdk.json` arquivo AWS CDK CLI e, o AWS CDK CLI valor terá precedência.

Recuperar valores de variáveis de contexto

Para obter o valor de uma variável de contexto em seu aplicativo, use o `TryGetContext` método no contexto de uma construção. (Ou seja, quando`this`, ou `self` em Python, é uma instância de alguma construção.)

Neste exemplo, recuperamos o valor da variável de `bucket_name` contexto. Se o valor solicitado não estiver definido, `TryGetContext` retornará `undefined` (`None`em Python; `null` em Java e C#; `nil` em Go) em vez de gerar uma exceção.

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");
```

Fora do contexto de uma construção, você pode acessar a variável de contexto do objeto do aplicativo, dessa forma.

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");
```

Para obter mais detalhes sobre como trabalhar com variáveis de contexto, consultethe section called "Contexto".

Usando recursos do Registro AWS CloudFormation Público

O Registro AWS CloudFormation Público permite gerenciar extensões, públicas e privadas, como recursos, módulos e ganchos que estão disponíveis para uso em seu Conta da AWS. Você pode usar extensões de recursos públicos em seus AWS Cloud Development Kit (AWS CDK) aplicativos com a `CfnResource`construção.

Para saber mais sobre o Registro AWS CloudFormation Público, consulte Usando o AWS CloudFormation registro no Guia do AWS CloudFormation Usuário.

Todas as extensões públicas publicadas pela AWS estão disponíveis para todas as contas em todas as regiões sem nenhuma ação de sua parte. No entanto, você deve ativar cada extensão de terceiros que deseja usar, em cada conta e região em que deseja usá-la.

> **Note**
>
> Ao usar AWS CloudFormation com tipos de recursos de terceiros, você incorrerá em cobranças. As cobranças são baseadas no número de operações do manipulador que você executa por mês e na duração da operação do manipulador. Consulte CloudFormation os preços para obter detalhes completos.

Para saber mais sobre extensões públicas, consulte Usando extensões públicas CloudFormation no Guia do AWS CloudFormation usuário

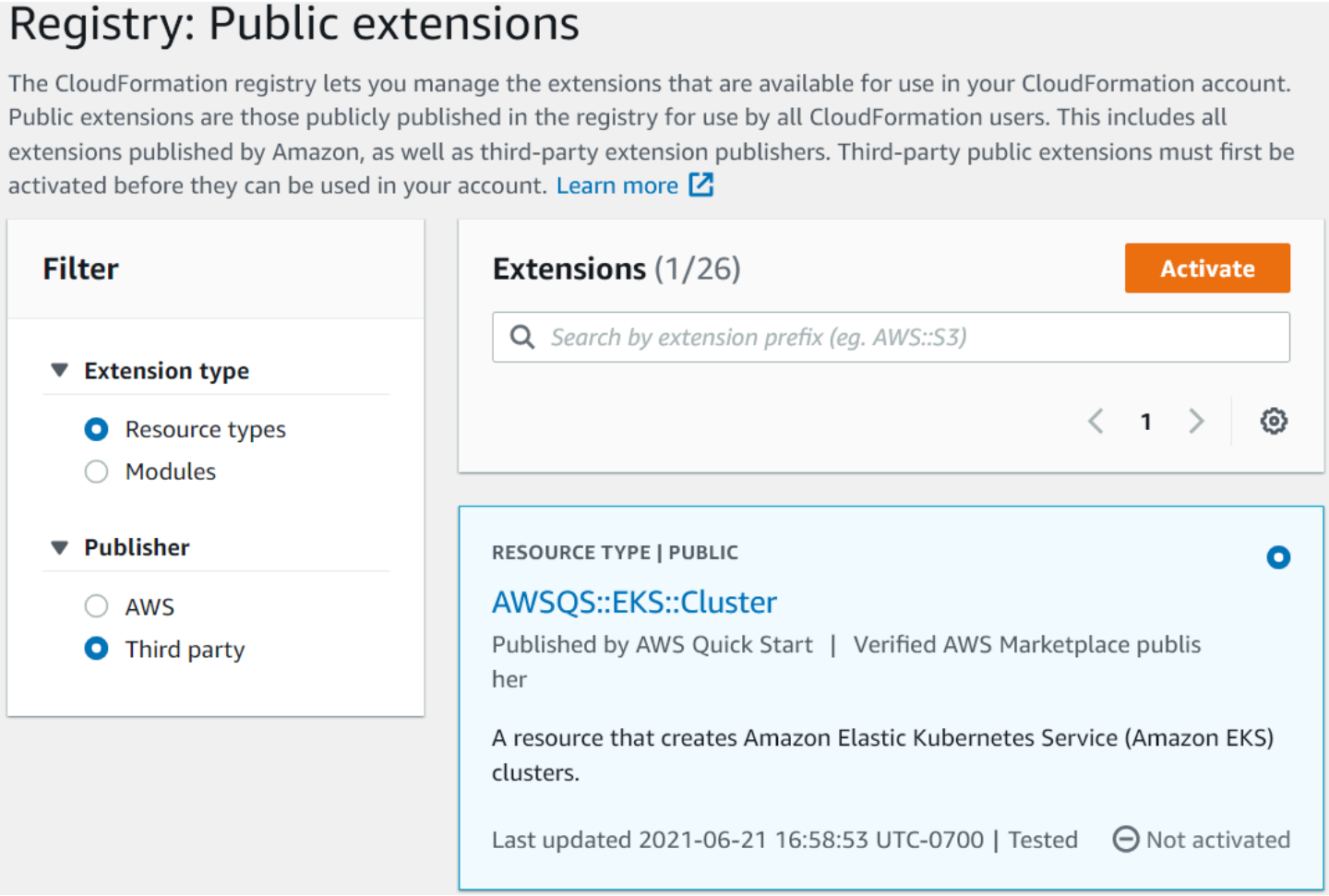
Tópicos

- Ativando um recurso de terceiros em sua conta e região
- Adicionar um recurso do Registro AWS CloudFormation Público ao seu aplicativo CDK

Ativando um recurso de terceiros em sua conta e região

As extensões publicadas por AWS não exigem ativação. Eles estão sempre disponíveis em todas as contas e regiões. Você pode ativar uma extensão de terceiros por meio do AWS Management Console, por meio do AWS Command Line Interface ou implantando um AWS CloudFormation recurso especial.

Para ativar uma extensão de terceiros por meio do AWS Management Console ou ver quais recursos estão disponíveis



1. Faça login na AWS conta na qual você deseja usar a extensão e, em seguida, alterne para a região em que deseja usá-la.
2. Navegue até o CloudFormation console por meio do menu Serviços.
3. Escolha Extensões públicas na barra de navegação e ative o botão de rádio de terceiros em Publisher. Uma lista das extensões públicas de terceiros disponíveis é exibida. (Você também

pode AWSoptar por ver uma lista das extensões públicas publicadas por AWS, embora não precise ativá-las.)

4. Navegue pela lista e encontre a extensão que você deseja ativar. Como alternativa, procure-o e ative o botão de rádio no canto superior direito do cartão da extensão.
5. Escolha o botão Ativar na parte superior da lista para ativar a extensão selecionada. A página Ativar da extensão é exibida.
6. Na página Ativar, você pode substituir o nome padrão da extensão e especificar uma função de execução e uma configuração de registro. Você também pode escolher se deseja atualizar automaticamente a extensão quando uma nova versão for lançada. Depois de definir essas opções conforme desejar, escolha Ativar extensão na parte inferior da página.

Para ativar uma extensão de terceiros usando o AWS CLI

- Use o comando `activate-type`. Substitua o ARN do tipo personalizado que você deseja usar onde indicado.

  Veja um exemplo a seguir:

  ```
  aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
  ```

Para ativar uma extensão de terceiros por meio de CloudFormation nosso CDK

- Implante um recurso do tipo `AWS::CloudFormation::TypeActivation` e especifique as seguintes propriedades:

  a. `TypeName`- O nome do tipo, como`AWSQS::EKS::Cluster`.

  b. `MajorVersion`- O número da versão principal da extensão que você deseja. Omita se você quiser a versão mais recente.

  c. `AutoUpdate`- Se essa extensão deve ser atualizada automaticamente quando uma nova versão secundária for lançada pelo editor. (As principais atualizações da versão exigem a alteração explícita da `MajorVersion` propriedade.)

  d. `ExecutionRoleArn`- O ARN da função do IAM sob a qual essa extensão será executada.

  e. `LoggingConfig`- A configuração de registro da extensão.

O `TypeActivation` recurso pode ser implantado pelo CDK usando a `CfnResource`construção. Isso é mostrado para as extensões reais na seção a seguir.

Adicionar um recurso do Registro AWS CloudFormation Público ao seu aplicativo CDK

Use a `CfnResource`construção para incluir um recurso do Registro AWS CloudFormation Público em seu aplicativo. Essa construção está no `aws-cdk-lib` módulo do CDK.

Por exemplo, suponha que haja um recurso público chamado `MY::S5::UltimateBucket` que você deseja usar em seu AWS CDK aplicativo. Esse recurso usa uma propriedade: o nome do bucket. A `CfnResource` instanciação correspondente tem esta aparência.

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"
    }
});
```

Implantação de aplicativos AWS CDK

Implemente AWS Cloud Development Kit (AWS CDK) aplicativos.

Tópicos

AWS CDK validação da política no momento da síntese

Tópicos

Validação da política no momento da síntese

Se você ou sua organização usarem alguma ferramenta de validação de políticas, como AWS CloudFormation Guarda OPA, para definir restrições em seu AWS CloudFormation modelo, você poderá integrá-las ao AWS CDK no momento da síntese. Ao usar o plug-in de validação de política apropriado, você pode fazer com que o AWS CDK aplicativo verifique o AWS CloudFormation modelo gerado em relação às suas políticas imediatamente após a síntese. Se houver alguma violação, a síntese falhará e um relatório será impresso no console.

A validação realizada pelo AWS CDK no momento da síntese valida os controles em um ponto do ciclo de vida da implantação, mas eles não podem afetar as ações que ocorrem fora da síntese. Os exemplos incluem ações realizadas diretamente no console ou por meio de APIs de serviço. Eles não são resistentes à alteração dos AWS CloudFormation modelos após a síntese. Algum outro mecanismo para validar o mesmo conjunto de regras com mais autoridade deve ser configurado de forma independente, como AWS CloudFormation ganchos ou. AWS Config No entanto, a capacidade do de AWS CDK avaliar o conjunto de regras durante o desenvolvimento ainda é útil, pois melhorará a velocidade de detecção e a produtividade do desenvolvedor.

O objetivo da validação da AWS CDK política é minimizar a quantidade de configuração necessária durante o desenvolvimento e torná-la o mais fácil possível.

> **Note**
>
> Esse recurso é considerado experimental, e tanto a API do plug-in quanto o formato do relatório de validação estão sujeitos a alterações no futuro.

Tópicos

Para desenvolvedores de aplicativos

Para usar um ou mais plug-ins de validação em seu aplicativo, use a `policyValidationBeta1` propriedade de`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: [...],
});
```

Imediatamente após a síntese, todos os plug-ins registrados dessa forma serão invocados para validar todos os modelos gerados no escopo que você definiu. Em particular, se você registrar os modelos no `App` objeto, todos os modelos estarão sujeitos à validação.

> **Warning**
>
> Além de modificar a montagem da nuvem, os plug-ins podem fazer tudo o que seu AWS CDK aplicativo pode. Eles podem ler dados do sistema de arquivos, acessar a rede etc. É sua responsabilidade, como consumidor de um plug-in, verificar se ele é seguro de usar.

AWS CloudFormation Guard plug-in

O uso do CfnGuardValidatorplug-in permite que você o use AWS CloudFormation Guardpara realizar validações de políticas. O `CfnGuardValidator` plug-in vem com um conjunto selecionado de controles AWS Control Tower proativos incorporados. O conjunto atual de regras pode ser encontrado na documentação do projeto. Conforme mencionado emValidação da política no momento da síntese, recomendamos que as organizações criem um método de validação mais confiável usando AWS CloudFormation ganchos.

Para AWS Control Toweros clientes, esses mesmos controles proativos podem ser implantados em toda a sua organização. Quando você ativa controles AWS Control Tower proativos em seu AWS Control Tower ambiente, os controles podem interromper a implantação de recursos não compatíveis implantados via. AWS CloudFormation Para obter mais informações sobre controles proativos gerenciados e como eles funcionam, consulte a AWS Control Tower documentação.

Esses controles AWS CDK agrupados e controles AWS Control Tower proativos gerenciados são melhor usados juntos. Nesse cenário, você pode configurar esse plug-in de validação com os mesmos controles proativos que estão ativos em seu ambiente de AWS Control Tower nuvem. Em seguida, você pode rapidamente ter certeza de que seu AWS CDK aplicativo passará pelos AWS Control Tower controles sendo executado `cdk synth` localmente.

Relatório de validação

Quando você sintetiza o AWS CDK aplicativo, os plug-ins do validador serão chamados e os resultados serão impressos. Um exemplo de relatório está sendo exibido abaixo.

```
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
```

Por padrão, o relatório será impresso em um formato legível por humanos. Se você quiser um relatório no formato JSON, ative-o usando `@aws-cdk/core:validationReportJson` a CLI ou passando-o diretamente para o aplicativo:

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

Como alternativa, você pode definir esse par de chave-valor de contexto usando os `cdk.context.json` arquivos `cdk.json` ou no diretório do seu projeto (consulteContexto de runtime).

Se você escolher o formato JSON, eles AWS CDK imprimirão o relatório de validação da política em um arquivo chamado `policy-validation-report.json` no diretório de montagem na nuvem. Para o formato padrão legível por humanos, o relatório será impresso na saída padrão.

Para autores de plug-ins

Plug-ins

A estrutura AWS CDK principal é responsável por registrar e invocar plug-ins e, em seguida, exibir o relatório de validação formatado. A responsabilidade do plug-in é atuar como a camada de tradução entre a AWS CDK estrutura e a ferramenta de validação de políticas. Um plug-in pode ser criado em qualquer idioma suportado pelo AWS CDK. Se você estiver criando um plug-in que pode ser consumido por vários idiomas, é recomendável criar o plug-in `TypeScript` para poder usar o JSII para publicar o plug-in em cada AWS CDK idioma.

Criação de plug-ins

O protocolo de comunicação entre o módulo AWS CDK principal e sua ferramenta de política é definido pela `IPolicyValidationPluginBeta1` interface. Para criar um novo plug-in, você deve escrever uma classe que implemente essa interface. Há duas coisas que você precisa implementar: o nome do plug-in (substituindo a `name` propriedade) e o `validate()` método.

A estrutura chamará`validate()`, passando um `IValidationContextBeta1` objeto. A localização dos modelos a serem validados é fornecida por`templatePaths`. O plug-in deve retornar uma instância de`ValidationPluginReportBeta1`. Esse objeto representa o relatório que o usuário receberá ao final da síntese.

```
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',
      }],
    }],
  };
```

```
}
```

Observe que os plug-ins não podem modificar nada na montagem da nuvem. Qualquer tentativa de fazer isso resultará em falha na síntese.

Se o seu plug-in depender de uma ferramenta externa, lembre-se de que alguns desenvolvedores talvez ainda não tenham essa ferramenta instalada em suas estações de trabalho. Para minimizar o atrito, é altamente recomendável que você forneça algum script de instalação junto com o pacote de plug-ins, para automatizar todo o processo. Melhor ainda, execute esse script como parte da instalação do seu pacote. Comnpm, por exemplo, você pode adicioná-lo ao `postinstall` script no `package.json` arquivo.

Lidando com isenções

Se sua organização tiver um mecanismo para lidar com isenções, ele poderá ser implementado como parte do plug-in validador.

Um exemplo de cenário para ilustrar um possível mecanismo de isenção:

- Uma organização tem uma regra de que buckets públicos do Amazon S3 não são permitidos, exceto em determinados cenários.
- Um desenvolvedor está criando um bucket do Amazon S3 que se enquadra em um desses cenários e solicita uma isenção (crie um ticket, por exemplo).
- As ferramentas de segurança sabem como ler o sistema interno que registra isenções

Nesse cenário, o desenvolvedor solicitaria uma exceção no sistema interno e, em seguida, precisaria de alguma forma de “registrar” essa exceção. Além do exemplo do plug-in guard, você pode criar um plug-in que lida com isenções filtrando as violações que têm uma isenção correspondente em um sistema interno de tíquetes.

Veja os plug-ins existentes para ver exemplos de implementações.

- @cdklabs/cdk-validator-cfnguard

Integração e entrega contínuas (CI/CD) usando CDK Pipelines

Use o módulo CDK Pipelines da AWS Construct Library para configurar a entrega contínua de aplicativos. AWS CDK Quando você confirma o código-fonte do seu aplicativo CDK em

AWS CodeCommitGitHub, ou AWS CodeStar, o CDK Pipelines pode criar, testar e implantar automaticamente sua nova versão.

O CDK Pipelines é atualizado automaticamente. Se você adicionar estágios ou pilhas de aplicativos, o pipeline se reconfigura automaticamente para implantar esses novos estágios ou pilhas.

> **Note**
>
> O CDK Pipelines oferece suporte a duas APIs. Uma delas é a API original que foi disponibilizada no CDK Pipelines Developer Preview. A outra é uma API moderna que incorpora o feedback dos clientes do CDK recebido durante a fase de pré-visualização. Os exemplos neste tópico usam a API moderna. Para obter detalhes sobre as diferenças entre as duas APIs compatíveis, consulte a API original do CDK Pipelines no repositório aws-cdk. GitHub

Tópicos

Inicialize seus ambientes AWS

Antes de usar o CDK Pipelines, você deve inicializar AWS o ambiente no qual implantará suas pilhas.

Um pipeline de CDK envolve pelo menos dois ambientes. O primeiro ambiente é onde o pipeline é provisionado. O segundo ambiente é onde você deseja implantar as pilhas ou estágios do aplicativo (os estágios são grupos de pilhas relacionadas). Esses ambientes podem ser os mesmos, mas uma recomendação de melhores práticas é isolar os estágios uns dos outros em ambientes diferentes.

> **Note**
>
> Consulte the section called “Bootstrapping” para obter mais informações sobre os tipos de recursos criados pelo bootstrap e como personalizar a pilha de bootstrap.

A implantação contínua com o CDK Pipelines exige que o seguinte seja incluído na pilha do CDK Toolkit:

- Um bucket do Amazon Simple Storage Service (Amazon S3).
- Um repositório Amazon ECR.
- Funções do IAM para dar às várias partes de um pipeline as permissões de que elas precisam.

O CDK Toolkit atualizará sua pilha de bootstrap existente ou criará uma nova, se necessário.

Para inicializar um ambiente que possa provisionar um AWS CDK pipeline, invoque `cdk bootstrap` conforme mostrado no exemplo a seguir. Invocar o AWS CDK kit de ferramentas por meio do `npx` comando o instala temporariamente, se necessário. Ele também usará a versão do kit de ferramentas instalada no projeto atual, se houver.

--cloudformation-execution-policiesespecifica o ARN de uma política sob a qual as futuras implantações do CDK Pipelines serão executadas. A `AdministratorAccess` política padrão garante que seu pipeline possa implantar todo tipo de AWS recurso. Se você usar essa política, certifique-se de confiar em todos os códigos e dependências que compõem seu AWS CDK aplicativo.

A maioria das organizações exige controles mais rígidos sobre quais tipos de recursos podem ser implantados pela automação. Consulte o departamento apropriado da sua organização para determinar a política que seu funil deve usar.

Você pode omitir a --profile opção se seu AWS perfil padrão contiver a configuração de autenticação necessária e. Região da 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
```

Para inicializar ambientes adicionais nos quais os AWS CDK aplicativos serão implantados pelo pipeline, use os comandos a seguir. A --trust opção indica qual outra conta deve ter permissões para implantar AWS CDK aplicativos nesse ambiente. Para essa opção, especifique o ID da AWS conta do funil.

Novamente, você pode omitir a --profile opção se seu AWS perfil padrão contiver a configuração de autenticação necessária e. Região da 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**
>
> Use credenciais administrativas somente para inicializar e provisionar o pipeline inicial. Depois, use o pipeline em si, não sua máquina local, para implantar as alterações.

Se você estiver atualizando um ambiente antigo inicializado, o bucket anterior do Amazon S3 ficará órfão quando o novo bucket for criado. Exclua-o manualmente usando o console do Amazon S3.

Inicializar um projeto

Crie um novo GitHub projeto vazio e clone-o em sua estação de trabalho no `my-pipeline` diretório. (Nossos exemplos de código neste tópico usam GitHub. Você também pode usar AWS CodeStar ou AWS CodeCommit.)

```
git clone GITHUB-CLONE-URL my-pipeline
cd my-pipeline
```

> **Note**
>
> Você pode usar um nome diferente do `my-pipeline` diretório principal do seu aplicativo. No entanto, se você fizer isso, precisará ajustar os nomes dos arquivos e das classes posteriormente neste tópico. Isso ocorre porque o AWS CDK Toolkit baseia alguns nomes de arquivos e classes no nome do diretório principal.

Após a clonagem, inicialize o projeto normalmente.

TypeScript

```
cdk init app --language typescript
```

JavaScript

```
cdk init app --language javascript
```

Python

```
cdk init app --language python
```

Depois que o aplicativo for criado, insira também os dois comandos a seguir. Eles ativam o ambiente virtual Python do aplicativo e instalam as dependências AWS CDK principais.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
cdk init app --language java
```

Se você estiver usando um IDE, agora você pode abrir ou importar o projeto. No Eclipse, por exemplo, escolha Arquivo > Importar > Maven > Projetos Maven existentes. Certifique-se de que as configurações do projeto estejam definidas para usar o Java 8 (1.8).

C#

```
cdk init app --language csharp
```

Se você estiver usando o Visual Studio, abra o arquivo da solução no `src` diretório.

Go

```
cdk init app --language go
```

Depois que o aplicativo for criado, insira também o comando a seguir para instalar os módulos do AWS Construct Library que o aplicativo exige.

```
go get
```

> **Important**
>
> Certifique-se de enviar seus arquivos `cdk.json` e `cdk.context.json` arquivos para o controle de origem. As informações de contexto (como sinalizadores de recursos e valores em cache recuperados da sua AWS conta) fazem parte do estado do seu projeto. Os valores podem ser diferentes em outro ambiente, o que pode causar alterações inesperadas nos resultados. Para ter mais informações, consulte the section called “Contexto”.

Definir um pipeline

Seu aplicativo CDK Pipelines incluirá pelo menos duas pilhas: uma que representa o próprio pipeline e uma ou mais pilhas que representam o aplicativo implantado por meio dele. As pilhas também podem ser agrupadas em estágios, que você pode usar para implantar cópias das pilhas de

infraestrutura em diferentes ambientes. Por enquanto, consideraremos o pipeline e, posteriormente, nos aprofundaremos no aplicativo que ele implantará.

A construção `CodePipeline`é a construção que representa um pipeline de CDK usado AWS CodePipeline como mecanismo de implantação. Ao instanciar `CodePipeline` em uma pilha, você define o local de origem do pipeline (como um GitHub repositório). Você também define os comandos para criar o aplicativo.

Por exemplo, o seguinte define um pipeline cuja fonte é armazenada em um GitHub repositório. Também inclui uma etapa de criação para um aplicativo TypeScript CDK. Preencha as informações sobre seu GitHub repositório onde indicado.

> **Note**
>
> Por padrão, o pipeline se autentica GitHub usando um token de acesso pessoal armazenado no Secrets Manager sob o nome`github-token`.

Você também precisará atualizar a instanciação da pilha do pipeline para especificar a AWS conta e a região.

TypeScript

Em `lib/my-pipeline-stack.ts` (pode variar se a pasta do seu projeto não tiver um nome`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']
      })
    });
```

```
  }
}
```

Em `bin/my-pipeline.ts` (pode variar se a pasta do seu projeto não tiver um nome `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

Em `lib/my-pipeline-stack.js` (pode variar se a pasta do seu projeto não tiver um nome `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 }
```

Em `bin/my-pipeline.js` (pode variar se a pasta do seu projeto não tiver um nome `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

Em `my-pipeline/my-pipeline-stack.py` (pode variar se a pasta do seu projeto não tiver um nome `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"]
```

```
                        )
                    )
```

Em `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

Em `src/main/java/com/myorg/MyPipelineStack.java` (pode variar se a pasta do projeto não tiver um nome`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();
    }
}
```

Em `src/main/java/com/myorg/MyPipelineApp.java` (pode variar se a pasta do projeto não tiver um nome`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#

Em `src/MyPipeline/MyPipelineStack.cs` (pode variar se a pasta do projeto não tiver um nome`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" }
                })
            });
        }
    }
}
```

Em `src/MyPipeline/Program.cs` (pode variar se a pasta do projeto não tiver um nome`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();
        }
    }
}
```

Você deve implantar um pipeline manualmente uma vez. Depois disso, o pipeline se mantém atualizado a partir do repositório do código-fonte. Portanto, certifique-se de que o código no

repositório seja o código que você deseja implantar. Verifique suas alterações, envie para GitHub, em seguida, implante:

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

> **Tip**
>
> Agora que você fez a implantação inicial, sua AWS conta local não precisa mais de acesso administrativo. Isso ocorre porque todas as alterações em seu aplicativo serão implantadas por meio do pipeline. Tudo o que você precisa fazer é pressionar para GitHub.

Etapas de aplicação

Para definir um AWS aplicativo de várias pilhas que possa ser adicionado ao pipeline de uma só vez, defina uma subclasse de. `Stage` (Isso é diferente do `CdkStage` módulo CDK Pipelines.)

O estágio contém as pilhas que compõem seu aplicativo. Se houver dependências entre as pilhas, as pilhas serão adicionadas automaticamente ao pipeline na ordem correta. As pilhas que não dependem umas das outras são implantadas paralelamente. Você pode adicionar uma relação de dependência entre as pilhas chamando. `stack1.addDependency(stack2)`

Os estágios aceitam um `env` argumento padrão, que se torna o ambiente padrão para as pilhas dentro dele. (As pilhas ainda podem ter seu próprio ambiente especificado.).

Um aplicativo é adicionado ao pipeline chamando `addStage()` com instâncias de `Stage`. Um estágio pode ser instanciado e adicionado ao pipeline várias vezes para definir diferentes estágios do seu pipeline de aplicativos DTAP ou multirregional.

Criaremos uma pilha contendo uma função Lambda simples e colocaremos essa pilha em um estágio. Em seguida, adicionaremos o estágio ao pipeline para que ele possa ser implantado.

TypeScript

Crie o novo arquivo `lib/my-pipeline-lambda-stack.ts` para armazenar nossa pilha de aplicativos contendo uma função 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";')
      });
    }
}
```

Crie o novo arquivo `lib/my-pipeline-app-stage.ts` para manter nosso palco.

```
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');
    }
}
```

Edite `lib/my-pipeline-stack.ts` para adicionar o estágio ao nosso pipeline.

```
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

Crie o novo arquivo `lib/my-pipeline-lambda-stack.js` para armazenar nossa pilha de aplicativos contendo uma função 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 }
```

Crie o novo arquivo `lib/my-pipeline-app-stage.js` para manter nosso palco.

```
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 };
```

Edite `lib/my-pipeline-stack.ts` para adicionar o estágio ao nosso pipeline.

```
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

Crie o novo arquivo `my_pipeline/my_pipeline_lambda_stack.py` para armazenar nossa pilha de aplicativos contendo uma função 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';")
        )
```

Crie o novo arquivo `my_pipeline/my_pipeline_app_stage.py` para manter nosso palco.

```
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")
```

Edite `my_pipeline/my-pipeline-stack.py` para adicionar o estágio ao nosso pipeline.

```
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

Crie o novo arquivo `src/main/java/com.myorg/MyPipelineLambdaStack.java` para armazenar nossa pilha de aplicativos contendo uma função 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();

    }

}
```

Crie o novo arquivo `src/main/java/com.myorg/MyPipelineAppStage.java` para manter nosso palco.

```
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");
    }

}
```

Edite `src/main/java/com.myorg/MyPipelineStack.java` para adicionar o estágio ao nosso pipeline.

```
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#

Crie o novo arquivo `src/MyPipeline/MyPipelineLambdaStack.cs` para armazenar nossa pilha de aplicativos contendo uma função 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';")
            });
        }
    }
}
```

Crie o novo arquivo `src/MyPipeline/MyPipelineAppStage.cs` para manter nosso palco.

```
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");
        }
    }
}
```

Edite `src/MyPipeline/MyPipelineStack.cs` para adicionar o estágio ao nosso pipeline.

```
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"
                }
            }));
        }
    }
}
```

Cada estágio do aplicativo adicionado por `addStage()` resulta na adição de um estágio de pipeline correspondente, representado por uma StageDeploymentinstância retornada pela `addStage()` chamada. Você pode adicionar ações de pré-implantação ou pós-implantação ao estágio chamando seu método `addPre()` ou`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"));
```

Você pode adicionar estágios a um Wave para implantá-los paralelamente, por exemplo, ao implantar um estágio em várias contas ou regiões.

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"
    }
}));
```

Testando implantações

Você pode adicionar etapas a um pipeline de CDK para validar as implantações que você está realizando. Por exemplo, você pode usar a biblioteca CDK Pipeline `ShellStep` para realizar tarefas como as seguintes:

- Tentando acessar um Amazon API Gateway recém-implantado apoiado por uma função Lambda
- Verificando a configuração de um recurso implantado emitindo um comando AWS CLI

Em sua forma mais simples, adicionar ações de validação tem a seguinte aparência:

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'" }
}));
```

Muitas AWS CloudFormation implantações resultam na geração de recursos com nomes imprevisíveis. Por esse motivo, o CDK Pipelines fornece uma maneira de AWS CloudFormation ler as saídas após uma implantação. Isso possibilita passar (por exemplo) a URL gerada de um balanceador de carga para uma ação de teste.

Para usar saídas, exponha o `CfnOutput` objeto em que você está interessado. Em seguida, passe-o na `envFromCfnOutputs` propriedade de uma etapa para disponibilizá-lo como uma variável de ambiente nessa etapa.

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" }
}));
```

Você pode escrever testes de validação simples diretamente no`ShellStep`, mas essa abordagem se torna complicada quando o teste tem mais do que algumas linhas. Para testes mais complexos, você pode trazer arquivos adicionais (como scripts de shell completos ou programas em outras linguagens) para a propriedade `ShellStep` por meio da `inputs` propriedade. As entradas podem ser qualquer etapa que tenha uma saída, incluindo uma fonte (como um GitHub repositório) ou outra. `ShellStep`

Trazer arquivos do repositório de origem é apropriado se os arquivos puderem ser usados diretamente no teste (por exemplo, se eles próprios forem executáveis). Neste exemplo, declaramos nosso GitHub repositório como `source` (em vez de instanciá-lo em linha como parte do). `CodePipeline` Em seguida, passamos esse conjunto de arquivos para o pipeline e para o teste de validação.

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" }
}));
```

Obter os arquivos adicionais da etapa de sintetização é apropriado se seus testes precisarem ser compilados, o que é feito como parte da síntese.

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" }
}));
```

Observações de segurança

Qualquer forma de entrega contínua tem riscos de segurança inerentes. De acordo com o Modelo de Responsabilidade AWS Compartilhada, você é responsável pela segurança de suas informações na AWS nuvem. A biblioteca CDK Pipelines oferece uma vantagem inicial ao incorporar padrões seguros e melhores práticas de modelagem.

No entanto, por sua própria natureza, uma biblioteca que precisa de um alto nível de acesso para cumprir a finalidade pretendida não pode garantir segurança total. Há muitos vetores de ataque fora da AWS sua organização.

Em particular, tenha em mente o seguinte:

- Esteja atento ao software do qual você depende. Verifique todos os softwares de terceiros que você executa em seu pipeline, pois eles podem mudar a infraestrutura que é implantada.
- Use o bloqueio de dependências para evitar atualizações acidentais. A CDK Pipelines `package-lock.json` respeita `yarn.lock` e garante que suas dependências sejam as que você espera.
- O CDK Pipelines é executado em recursos criados em sua própria conta, e a configuração desses recursos é controlada pelos desenvolvedores que enviam o código pelo pipeline. Portanto, o CDK Pipelines por si só não pode se proteger contra desenvolvedores mal-intencionados que tentam contornar as verificações de conformidade. Se seu modelo de ameaça incluir desenvolvedores que escrevem código CDK, você deve ter mecanismos externos de conformidade, como AWS CloudFormation Hooks (preventivos) ou AWS Config (reativos), que a AWS CloudFormation Função de Execução não tenha permissão para desativar.

- As credenciais para ambientes de produção devem durar pouco. Após a inicialização e o provisionamento inicial, não é necessário que os desenvolvedores tenham as credenciais da conta. As mudanças podem ser implantadas por meio do pipeline. Reduza a possibilidade de vazamento de credenciais ao não precisar delas em primeiro lugar.

Solução de problemas

Os problemas a seguir são comumente encontrados ao começar a usar o CDK Pipelines.

Pipeline: falha interna

```
CREATE_FAILED  | AWS::CodePipeline::Pipeline | Pipeline/Pipeline
Internal Failure
```

Verifique seu token de GitHub acesso. Ele pode estar ausente ou pode não ter as permissões para acessar o repositório.

Chave: A política contém uma declaração com um ou mais princípios inválidos

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey
Policy contains a statement with one or more invalid principals.
```

Um dos ambientes de destino não foi inicializado com a nova pilha de bootstrap. Certifique-se de que todos os seus ambientes de destino estejam inicializados.

A pilha está no estado ROLLBACK_COMPLETE e não pode ser atualizada.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
ID: ...)
```

A pilha falhou em sua implantação anterior e está em um estado que não pode ser repetido. Exclua a pilha do AWS CloudFormation console e repita a implantação.

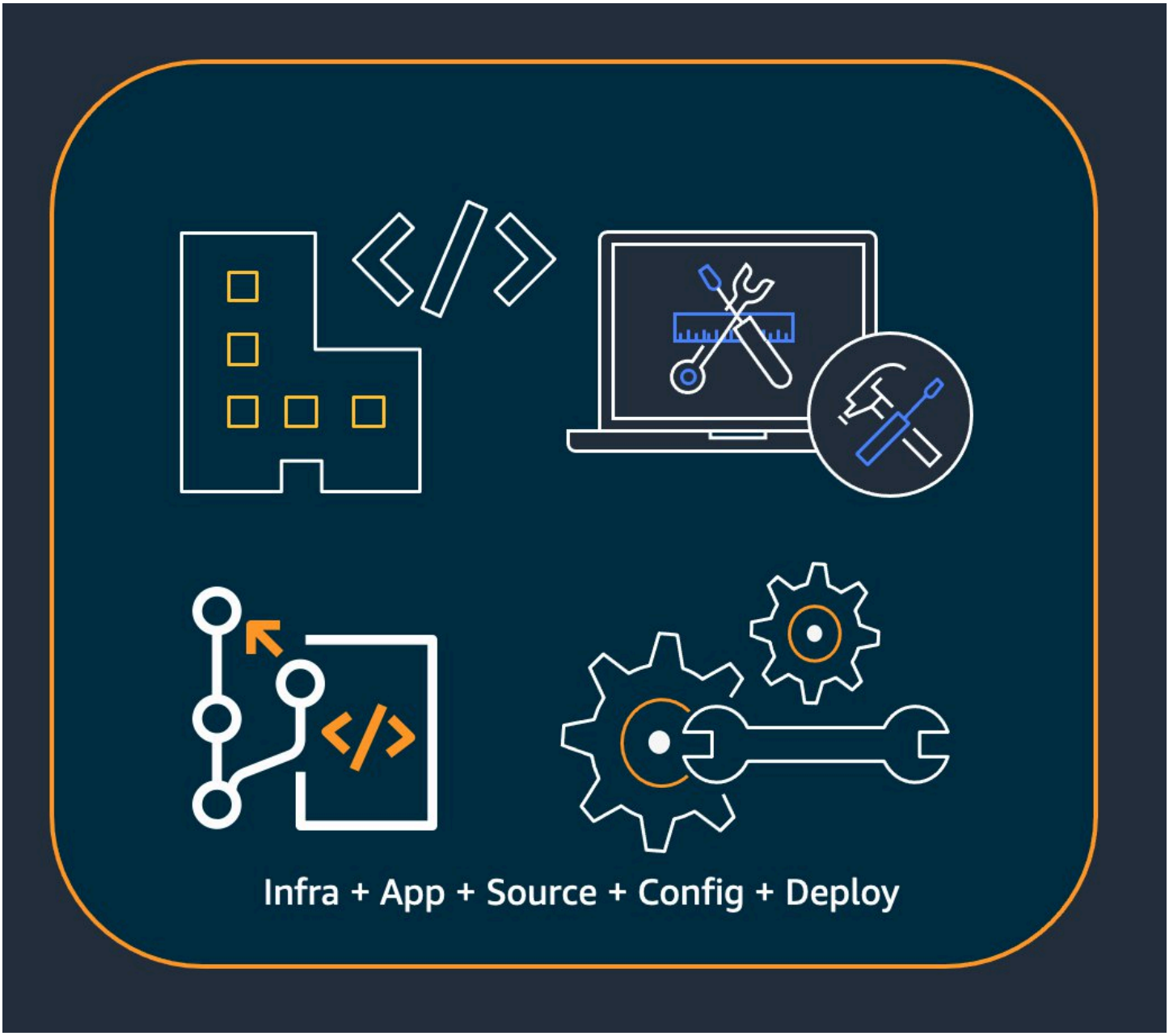
Melhores práticas para desenvolver e implantar infraestrutura em nuvem com o AWS CDK

Com o AWS CDK, desenvolvedores ou administradores podem definir sua infraestrutura de nuvem usando uma linguagem de programação compatível. Os aplicativos CDK devem ser organizados em unidades lógicas, como API, banco de dados e recursos de monitoramento, e, opcionalmente, ter um pipeline para implantações automatizadas. As unidades lógicas devem ser implementadas como construções, incluindo as seguintes:

- Infraestrutura (como buckets Amazon S3, bancos de dados Amazon RDS ou uma rede Amazon VPC)
- Código de tempo de execução (como AWS Lambda funções)
- Código de configuração

As pilhas definem o modelo de implantação dessas unidades lógicas. Para obter uma introdução mais detalhada aos conceitos por trás do CDK, consulteConceitos básicos.

AWS CDK Isso reflete uma análise cuidadosa das necessidades de nossos clientes e equipes internas e dos padrões de falha que geralmente surgem durante a implantação e a manutenção contínua de aplicativos complexos em nuvem. Descobrimos que as falhas geralmente estão relacionadas a out-of-band "" alterações em um aplicativo que não foram totalmente testadas, como alterações de configuração. Portanto, desenvolvemos em AWS CDK torno de um modelo no qual todo o seu aplicativo é definido em código, não apenas na lógica de negócios, mas também na infraestrutura e na configuração. Dessa forma, as mudanças propostas podem ser cuidadosamente analisadas, testadas de forma abrangente em ambientes semelhantes à produção em vários graus e totalmente revertidas se algo der errado.



No momento da implantação, ele AWS CDK sintetiza um conjunto de nuvem que contém o seguinte:

- AWS CloudFormation modelos que descrevem sua infraestrutura em todos os ambientes de destino
- Ativos de arquivo que contêm seu código de tempo de execução e seus arquivos de suporte

Com o CDK, cada confirmação na ramificação principal de controle de versão do seu aplicativo pode representar uma versão completa, consistente e implantável do seu aplicativo. Seu aplicativo pode então ser implantado automaticamente sempre que uma alteração for feita.

A filosofia por trás disso AWS CDK leva às nossas melhores práticas recomendadas, que dividimos em quatro grandes categorias.

- the section called “Práticas recomendadas para organizações”
- the section called “Práticas recomendadas de codificação”
- the section called “Crie as melhores práticas”
- the section called “Práticas recomendadas de aplicação”

> **Tip**
>
> Considere também as melhores práticas AWS CloudFormation e os AWS serviços individuais que você usa, quando aplicável à infraestrutura definida pela CDK.

Práticas recomendadas para organizações

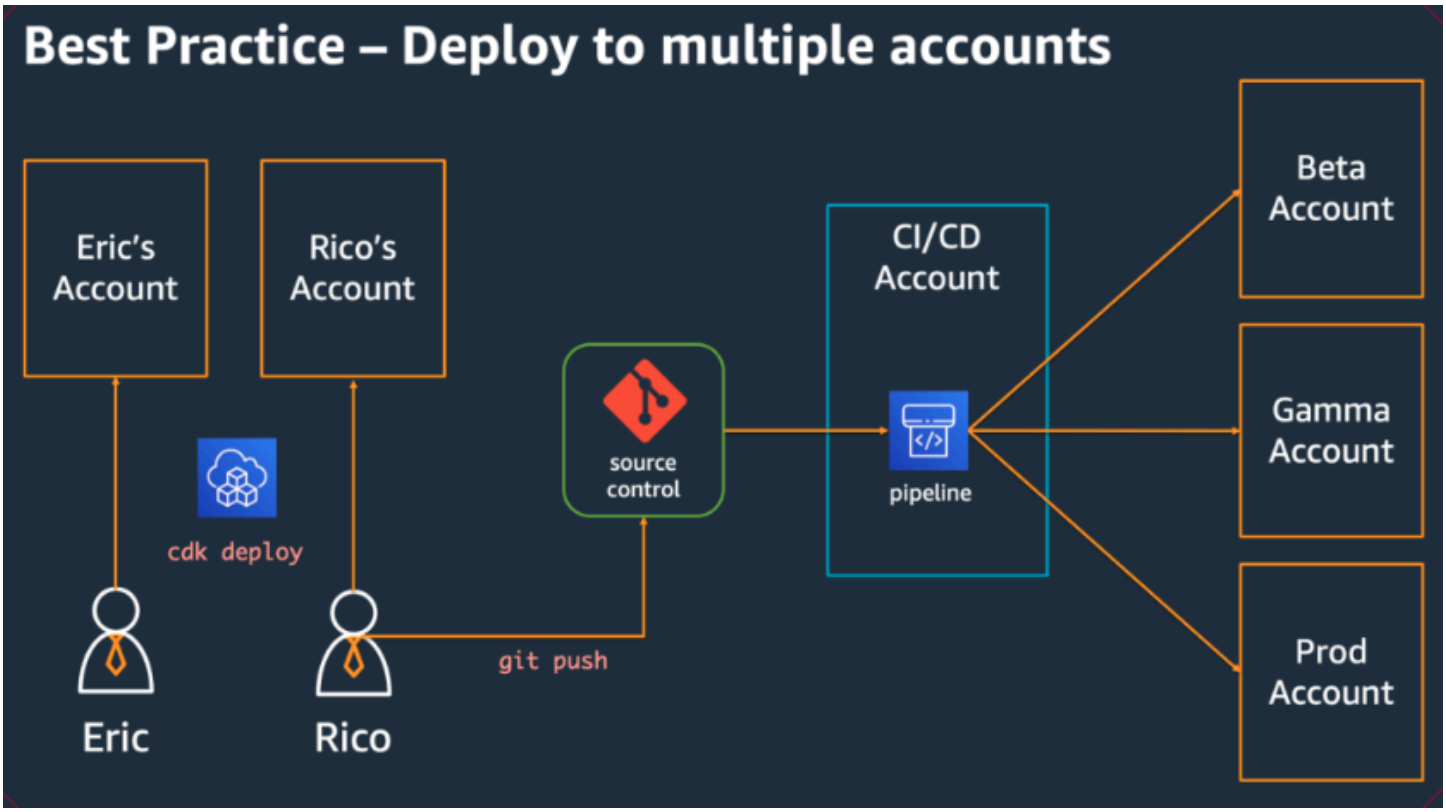
Nos estágios iniciais da AWS CDK adoção, é importante considerar como preparar sua organização para o sucesso. É uma boa prática ter uma equipe de especialistas responsável por treinar e orientar o resto da empresa à medida que eles adotam o CDK. O tamanho dessa equipe pode variar, desde uma ou duas pessoas em uma pequena empresa até um Cloud Center of Excellence (CCoE) completo em uma empresa maior. Essa equipe é responsável por definir padrões e políticas para a infraestrutura de nuvem em sua empresa e também por treinar e orientar desenvolvedores.

O CCoE pode fornecer orientação sobre quais linguagens de programação devem ser usadas para a infraestrutura de nuvem. Os detalhes variam de uma organização para outra, mas uma boa política ajuda a garantir que os desenvolvedores possam entender e manter a infraestrutura de nuvem da empresa.

O CCoE também cria uma “landing zone” que define suas unidades organizacionais internas. AWS Uma landing zone é um AWS ambiente pré-configurado, seguro, escalável e com várias contas baseado em esquemas de melhores práticas. Para unir os serviços que compõem sua landing zone, você pode usar AWS Control Tower, que configura e gerencia todo o seu sistema de várias contas a partir de uma única interface de usuário.

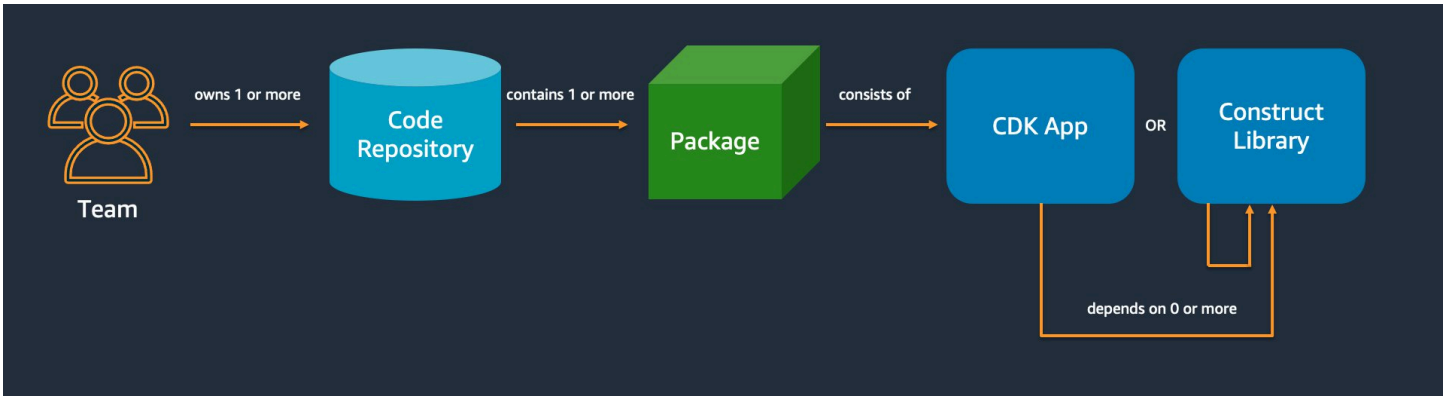
As equipes de desenvolvimento devem poder usar suas próprias contas para testar e implantar novos recursos nessas contas, conforme necessário. Desenvolvedores individuais podem tratar esses recursos como extensões de sua própria estação de trabalho de desenvolvimento. Usando o

CDK Pipelines, AWS CDK os aplicativos podem então ser implantados por meio de uma conta de CI/CD em ambientes de teste, integração e produção (cada um isolado em sua própria região ou conta). AWS Isso é feito mesclando o código dos desenvolvedores no repositório canônico da sua organização.



Práticas recomendadas de codificação

Esta seção apresenta as melhores práticas para organizar seu AWS CDK código. O diagrama a seguir mostra a relação entre uma equipe e os repositórios de código, pacotes, aplicativos e bibliotecas de construção dessa equipe.



Comece de forma simples e acrescente complexidade somente quando precisar

O princípio orientador da maioria das nossas melhores práticas é manter as coisas o mais simples possível, mas não mais simples. Adicione complexidade somente quando seus requisitos exigirem uma solução mais complicada. Com o AWS CDK, você pode refatorar seu código conforme necessário para suportar novos requisitos. Você não precisa arquitetar todos os cenários possíveis com antecedência.

Alinhe-se com o AWS Well-Architected Framework

O AWS Well-Architected Framework define um componente como o código, a configuração AWS e os recursos que, juntos, atendem a um requisito. Um componente geralmente é a unidade de propriedade técnica e é desacoplado de outros componentes. O termo carga de trabalho é usado para identificar um conjunto de componentes que, juntos, agregam valor comercial. Uma carga de trabalho geralmente é o nível de detalhes sobre o qual os líderes de negócios e tecnologia comunicam.

Um AWS CDK aplicativo mapeia para um componente conforme definido pelo AWS Well-Architected Framework. AWS CDK os aplicativos são um mecanismo para codificar e fornecer as melhores práticas de aplicativos em nuvem da Well-Architected. Você também pode criar e compartilhar componentes como bibliotecas de código reutilizáveis por meio de repositórios de artefatos, como. AWS CodeArtifact

Cada aplicativo começa com um único pacote em um único repositório

Um único pacote é o ponto de entrada do seu AWS CDK aplicativo. Aqui, você define como e onde implantar as diferentes unidades lógicas do seu aplicativo. Você também define o pipeline de CI/CD para implantar o aplicativo. As construções do aplicativo definem as unidades lógicas da sua solução.

Use pacotes adicionais para construções que você usa em mais de um aplicativo. (As construções compartilhadas também devem ter seu próprio ciclo de vida e estratégia de teste.) As dependências entre pacotes no mesmo repositório são gerenciadas pelas ferramentas de compilação do seu repositório.

Embora seja possível, não recomendamos colocar vários aplicativos no mesmo repositório, especialmente ao usar pipelines de implantação automatizados. Isso aumenta o “raio de explosão”

das mudanças durante a implantação. Quando há vários aplicativos em um repositório, as alterações em um aplicativo acionam a implantação dos outros (mesmo que os outros não tenham sido alterados). Além disso, uma interrupção em um aplicativo impede que os outros aplicativos sejam implantados.

Mova o código para repositórios com base no ciclo de vida do código ou na propriedade da equipe

Quando os pacotes começarem a ser usados em vários aplicativos, mova-os para seu próprio repositório. Dessa forma, os pacotes podem ser referenciados pelos sistemas de criação de aplicativos que os usam e também podem ser atualizados em cadências independentes dos ciclos de vida do aplicativo. No entanto, a princípio, pode fazer sentido colocar todas as construções compartilhadas em um repositório.

Além disso, mova os pacotes para seu próprio repositório quando equipes diferentes estiverem trabalhando neles. Isso ajuda a impor o controle de acesso.

Para consumir pacotes além dos limites do repositório, você precisa de um repositório de pacotes privado, semelhante ao NPM ou ao Maven Central PyPi, mas interno à sua organização. Você também precisa de um processo de lançamento que compile, teste e publique o pacote no repositório privado de pacotes. CodeArtifactpode hospedar pacotes para as linguagens de programação mais populares.

As dependências dos pacotes no repositório de pacotes são gerenciadas pelo gerenciador de pacotes da sua linguagem, como o NPM for TypeScript or applications. JavaScript Seu gerenciador de pacotes ajuda a garantir que as compilações sejam repetíveis. Isso é feito gravando as versões específicas de cada pacote do qual seu aplicativo depende. Ele também permite que você atualize essas dependências de forma controlada.

Pacotes compartilhados precisam de uma estratégia de teste diferente. Para um único aplicativo, pode ser suficiente implantá-lo em um ambiente de teste e confirmar se ele ainda funciona. Mas os pacotes compartilhados devem ser testados independentemente do aplicativo consumidor, como se estivessem sendo lançados ao público. (Sua organização pode optar por realmente lançar alguns pacotes compartilhados para o público.)

Lembre-se de que uma construção pode ser arbitrariamente simples ou complexa. `BucketA` é uma construção, mas também `CameraShopWebsite` pode ser uma construção.

A infraestrutura e o código de tempo de execução residem no mesmo pacote

Além de gerar AWS CloudFormation modelos para implantar a infraestrutura, ele AWS CDK também agrupa ativos de tempo de execução, como funções Lambda e imagens do Docker, e os implanta junto com sua infraestrutura. Isso possibilita combinar o código que define sua infraestrutura e o código que implementa sua lógica de tempo de execução em uma única construção. É uma boa prática fazer isso. Esses dois tipos de código não precisam estar em repositórios separados ou mesmo em pacotes separados.

Para desenvolver os dois tipos de código juntos, você pode usar uma construção independente que descreva completamente uma parte da funcionalidade, incluindo sua infraestrutura e lógica. Com uma construção independente, você pode testar os dois tipos de código isoladamente, compartilhar e reutilizar o código entre projetos e criar versões de todo o código em sincronia.

Crie as melhores práticas

Esta seção contém as melhores práticas para o desenvolvimento de construções. As construções são módulos reutilizáveis e combináveis que encapsulam recursos. Eles são os alicerces dos AWS CDK aplicativos.

Modele com construções, implante com pilhas

As pilhas são a unidade de implantação: tudo em uma pilha é implantado em conjunto. Portanto, ao criar as unidades lógicas de nível superior do seu aplicativo a partir de vários AWS recursos, represente cada unidade lógica como uma `Construct`, não como uma. `Stack` Use pilhas somente para descrever como suas construções devem ser compostas e conectadas para seus vários cenários de implantação.

Por exemplo, se uma de suas unidades lógicas for um site, as construções que a compõem (como um bucket do Amazon S3, API Gateway, funções Lambda ou tabelas do Amazon RDS) devem ser compostas em uma única construção de alto nível. Em seguida, essa construção deve ser instanciada em uma ou mais pilhas para implantação.

Ao usar construções para construção e pilhas para implantação, você melhora o potencial de reutilização de sua infraestrutura e oferece mais flexibilidade na forma como ela é implantada.

Configure com propriedades e métodos, não com variáveis de ambiente

Pesquisas de variáveis de ambiente dentro de construções e pilhas são um antipadrão comum. Tanto as construções quanto as pilhas devem aceitar um objeto de propriedades para permitir a configuração total totalmente no código. Fazer o contrário introduz uma dependência na máquina na qual o código será executado, o que cria ainda mais informações de configuração que você precisa rastrear e gerenciar.

Em geral, as pesquisas de variáveis de ambiente devem ser limitadas ao nível superior de um AWS CDK aplicativo. Eles também devem ser usados para transmitir informações necessárias para execução em um ambiente de desenvolvimento. Para ter mais informações, consulte the section called “Ambientes do”.

Teste unitário sua infraestrutura

Para executar consistentemente um conjunto completo de testes unitários no momento da construção em todos os ambientes, evite consultas na rede durante a síntese e modele todos os estágios de produção no código. (Essas melhores práticas serão abordadas posteriormente.) Se qualquer confirmação individual sempre resultar no mesmo modelo gerado, você pode confiar nos testes de unidade que você escreve para confirmar se os modelos gerados têm a aparência esperada. Para ter mais informações, consulte Testando construções.

Não altere a ID lógica dos recursos com estado

Alterar a ID lógica de um recurso resulta na substituição do recurso por um novo na próxima implantação. Para recursos com estado, como bancos de dados e buckets S3, ou infraestrutura persistente, como uma Amazon VPC, isso raramente é o que você deseja. Tenha cuidado com qualquer refatoração do AWS CDK código que possa fazer com que o ID seja alterado. Escreva testes de unidade que afirmem que os IDs lógicos de seus recursos com estado permanecem estáticos. O ID lógico é derivado do `id` que você especifica ao instanciar a construção e da posição da construção na árvore de construção. Para ter mais informações, consulte the section called “IDs lógicos”.

As construções não são suficientes para a conformidade

Muitos clientes corporativos escrevem seus próprios wrappers para construções L2 (as construções “selecionadas” que representam AWS recursos individuais com padrões sensatos integrados e melhores práticas). Esses wrappers aplicam as melhores práticas de segurança, como criptografia

estática e políticas específicas do IAM. Por exemplo, você pode criar um `MyCompanyBucket` para usar em seus aplicativos no lugar da construção usual do Amazon S3`Bucket`. Esse padrão é útil para apresentar diretrizes de segurança no início do ciclo de vida do desenvolvimento de software, mas não confie nelas como o único meio de fiscalização.

Em vez disso, use AWS recursos como políticas de controle de serviço e limites de permissão para reforçar suas barreiras de segurança no nível da organização. Use the section called “Aspectos” ferramentas como o CloudFormation Guard para fazer afirmações sobre as propriedades de segurança dos elementos da infraestrutura antes da implantação. Use AWS CDK para o que ele faz de melhor.

Por fim, lembre-se de que escrever suas próprias construções “L2+” pode impedir que seus desenvolvedores tirem proveito de AWS CDK pacotes como AWS Solutions Constructs ou construções de terceiros do Construct Hub. Esses pacotes geralmente são criados em AWS CDK construções padrão e não poderão usar suas construções de wrapper.

Práticas recomendadas de aplicação

Nesta seção, discutiremos como escrever seus AWS CDK aplicativos, combinando construções para definir como seus AWS recursos estão conectados.

Tome decisões na hora da síntese

Embora AWS CloudFormation permita que você tome decisões no momento da implantação (usando `Conditions{ Fn::If }`,, `eParameters`) e AWS CDK ofereça algum acesso a esses mecanismos, não recomendamos usá-los. Os tipos de valores que você pode usar e os tipos de operações que você pode realizar neles são limitados em comparação com o que está disponível em uma linguagem de programação de uso geral.

Em vez disso, tente tomar todas as decisões, como qual construção instanciar, em seu AWS CDK aplicativo usando as `if` instruções da sua linguagem de programação e outros recursos. Por exemplo, um idioma CDK comum, iterando em uma lista e instanciando uma construção com valores de cada item na lista, simplesmente não é possível usando expressões. AWS CloudFormation

Trate AWS CloudFormation como um detalhe de implementação que eles AWS CDK usam para implantações robustas em nuvem, não como um destino de linguagem. Você não está escrevendo AWS CloudFormation modelos em TypeScript Python, você está escrevendo um código CDK que por acaso é usado CloudFormation para implantação.

Use nomes de recursos gerados, não nomes físicos

Os nomes são um recurso precioso. Cada nome só pode ser usado uma vez. Portanto, se você codificar um nome de tabela ou um nome de bucket em sua infraestrutura e aplicativo, não poderá implantar essa parte da infraestrutura duas vezes na mesma conta. (O nome do qual estamos falando aqui é o nome especificado, por exemplo, pela `bucketName` propriedade em uma construção de bucket do Amazon S3.)

O pior é que você não pode fazer alterações no recurso que exijam sua substituição. Se uma propriedade só puder ser definida na criação do recurso, como a `KeySchema` de uma tabela do Amazon DynamoDB, essa propriedade será imutável. A alteração dessa propriedade requer um novo recurso. No entanto, o novo recurso deve ter o mesmo nome para ser um verdadeiro substituto. Mas ele não pode ter o mesmo nome enquanto o recurso existente ainda estiver usando esse nome.

Uma abordagem melhor é especificar o mínimo possível de nomes. Se você omitir os nomes dos recursos, eles os AWS CDK gerarão para você de uma forma que não cause problemas. Suponha que você tenha uma tabela como recurso. Em seguida, você pode passar o nome da tabela gerada como uma variável de ambiente para sua AWS Lambda função. Em seu AWS CDK aplicativo, você pode referenciar o nome da tabela com`otable.tableName`. Como alternativa, você pode gerar um arquivo de configuração na sua instância do Amazon EC2 na inicialização ou gravar o nome real da tabela no AWS Systems Manager Parameter Store para que seu aplicativo possa lê-lo a partir daí.

Se o local de que você precisa for outra AWS CDK pilha, isso é ainda mais simples. Supondo que uma pilha defina o recurso e outra pilha precise usá-lo, o seguinte se aplica:

- Se as duas pilhas estiverem no mesmo AWS CDK aplicativo, passe uma referência entre as duas pilhas. Por exemplo, salve uma referência à construção do recurso como um atributo da stack (`this.stack.uploadBucket = myBucket`) definidora. Em seguida, passe esse atributo para o construtor da pilha que precisa do recurso.
- Quando as duas pilhas estiverem em AWS CDK aplicativos diferentes, use um `from` método estático para usar um recurso definido externamente com base em seu ARN, nome ou outros atributos. (Por exemplo, use `Table.fromArn()` para uma tabela do DynamoDB). Use a `CfnOutput` construção para imprimir o ARN ou outro valor necessário na saída `decdk deploy`, ou consulte o. AWS Management Console Como alternativa, o segundo aplicativo pode ler o CloudFormation modelo gerado pelo primeiro aplicativo e recuperar esse valor da `Outputs` seção.

Defina políticas de remoção e retenção de registros

As AWS CDK tentativas de evitar que você perca dados adotando políticas que retêm tudo o que você cria. Por exemplo, a política de remoção padrão em recursos que contêm dados (como buckets do Amazon S3 e tabelas de banco de dados) é não excluir o recurso quando ele é removido da pilha. Em vez disso, o recurso fica órfão da pilha. Da mesma forma, o padrão do CDK é reter todos os registros para sempre. Em ambientes de produção, esses padrões podem resultar rapidamente no armazenamento de grandes quantidades de dados que você realmente não precisa e na fatura correspondente AWS .

Considere cuidadosamente o que você deseja que essas políticas sejam para cada recurso de produção e especifique-as adequadamente. Use the section called “Aspectos” para validar as políticas de remoção e registro em sua pilha.

Separe seu aplicativo em várias pilhas, conforme determinado pelos requisitos de implantação

Não há uma regra rígida e rápida de quantas pilhas seu aplicativo precisa. Normalmente, você acabará baseando a decisão em seus padrões de implantação. Lembre-se das seguintes diretrizes:

- Normalmente, é mais fácil manter o máximo possível de recursos na mesma pilha, então mantenha-os juntos, a menos que você saiba que quer separá-los.
- Considere manter os recursos com estado (como bancos de dados) em uma pilha separada dos recursos sem estado. Em seguida, você pode ativar a proteção contra rescisão na pilha com estado. Dessa forma, você pode destruir ou criar livremente várias cópias da pilha sem estado sem risco de perda de dados.
- Recursos com estado são mais sensíveis à renomeação de construções — renomear leva à substituição de recursos. Portanto, não agrupe recursos com estado em construções que provavelmente serão movidas ou renomeadas (a menos que o estado possa ser reconstruído se perdido, como um cache). Esse é outro bom motivo para colocar recursos com estado em sua própria pilha.

Comprometa-se **cdk.context.json** a evitar comportamentos não determinísticos

O determinismo é fundamental para AWS CDK implantações bem-sucedidas. Um AWS CDK aplicativo deve ter essencialmente o mesmo resultado sempre que for implantado em um determinado ambiente.

Como seu AWS CDK aplicativo é escrito em uma linguagem de programação de uso geral, ele pode executar código arbitrário, usar bibliotecas arbitrárias e fazer chamadas de rede arbitrárias. Por exemplo, você pode usar um AWS SDK para recuperar algumas informações da sua AWS conta enquanto sintetiza seu aplicativo. Reconheça que isso resultará em requisitos adicionais de configuração de credenciais, maior latência e uma chance, ainda que pequena, de falha toda vez que você correr. `cdk synth`

Nunca modifique sua AWS conta ou seus recursos durante a síntese. Sintetizar um aplicativo não deve ter efeitos colaterais. As alterações em sua infraestrutura devem ocorrer somente na fase de implantação, após a geração do AWS CloudFormation modelo. Dessa forma, se houver algum problema, AWS CloudFormation pode reverter automaticamente a alteração. Para fazer alterações que não podem ser feitas facilmente na AWS CDK estrutura, use recursos personalizados para executar código arbitrário no momento da implantação.

Mesmo chamadas estritamente somente para leitura não são necessariamente seguras. Considere o que acontece se o valor retornado por uma chamada de rede for alterado. Que parte da sua infraestrutura isso afetará? O que acontecerá com os recursos já implantados? A seguir estão dois exemplos de situações em que uma mudança repentina nos valores pode causar um problema.

- Se você provisionar uma Amazon VPC para todas as zonas de disponibilidade disponíveis em uma região específica e o número de AZs for dois no dia da implantação, seu espaço IP será dividido pela metade. Se AWS iniciar uma nova zona de disponibilidade no dia seguinte, a próxima implantação tentará dividir seu espaço IP em terços, exigindo que todas as sub-redes sejam recriadas. Isso provavelmente não será possível porque suas instâncias do Amazon EC2 ainda estão em execução e você precisará limpá-las manualmente.
- Se você consultar a imagem mais recente da máquina Amazon Linux e implantar uma instância do Amazon EC2 e, no dia seguinte, uma nova imagem for lançada, uma implantação subsequente selecionará a nova AMI e substituirá todas as suas instâncias. Isso pode não ser o que você esperava que acontecesse.

Essas situações podem ser perniciosas porque a mudança AWS lateral pode ocorrer após meses ou anos de implantações bem-sucedidas. De repente, suas implantações estão falhando “sem motivo” e você esqueceu há muito tempo o que fez e por quê.

Felizmente, isso AWS CDK inclui um mecanismo chamado provedores de contexto para registrar um instantâneo de valores não determinísticos. Isso permite que futuras operações de síntese produzam exatamente o mesmo modelo que produziam quando foram implantadas pela primeira vez. As únicas alterações no novo modelo são as alterações que você fez no seu código. Quando você usa o `.fromLookup()` método de uma construção, o resultado da chamada é armazenado em `cdk.context.json` cache. Você deve submeter isso ao controle de versão junto com o resto do seu código para garantir que futuras execuções do seu aplicativo CDK usem o mesmo valor. O CDK Toolkit inclui comandos para gerenciar o cache de contexto, para que você possa atualizar entradas específicas quando precisar. Para ter mais informações, consulte the section called “Contexto”.

Se você precisar de algum valor (de AWS ou de outro lugar) para o qual não haja um provedor de contexto CDK nativo, recomendamos escrever um script separado. O script deve recuperar o valor e gravá-lo em um arquivo e, em seguida, ler esse arquivo no seu aplicativo CDK. Execute o script somente quando quiser atualizar o valor armazenado, não como parte do seu processo de criação normal.

Deixe que eles AWS CDK gerenciem funções e grupos de segurança

Com os métodos `grant()` convenientes da biblioteca de construção do AWS CDK, você pode criar AWS Identity and Access Management funções que concedem acesso a um recurso por outro usando permissões com escopo mínimo. Por exemplo, considere uma linha como a seguinte:

```
myBucket.grantRead(myLambda)
```

Essa linha única adiciona uma política à função Lambda (que também é criada para você). Essa função e suas políticas são mais de uma dúzia de linhas CloudFormation que você não precisa escrever. Isso AWS CDK concede somente as permissões mínimas necessárias para que a função seja lida do bucket.

Se você exigir que os desenvolvedores sempre usem funções predefinidas criadas por uma equipe de segurança, a AWS CDK codificação se torna muito mais complicada. Suas equipes podem perder muita flexibilidade na forma como projetam seus aplicativos. Uma alternativa melhor é usar políticas de controle de serviços e limites de permissão para garantir que os desenvolvedores permaneçam dentro das barreiras.

Modele todas as etapas de produção em código

Em AWS CloudFormation cenários tradicionais, sua meta é produzir um único artefato parametrizado para que possa ser implantado em vários ambientes de destino após a aplicação de valores de configuração específicos a esses ambientes. No CDK, você pode e deve criar essa configuração em seu código-fonte. Crie uma pilha para seu ambiente de produção e crie uma pilha separada para cada um dos outros estágios. Em seguida, coloque os valores de configuração de cada pilha no código. Use serviços como Secrets Manager e Systems Manager Parameter Store para valores confidenciais que você não deseja verificar no controle de origem, usando os nomes ou ARNs desses recursos.

Quando você sintetiza seu aplicativo, o conjunto de nuvem criado na `cdk.out` pasta contém um modelo separado para cada ambiente. Toda a sua construção é determinística. Não há out-of-band alterações em seu aplicativo, e qualquer confirmação sempre gera exatamente o mesmo AWS CloudFormation modelo e os ativos que o acompanham. Isso torna o teste unitário muito mais confiável.

Meça tudo

Atingir a meta de implantação contínua completa, sem intervenção humana, requer um alto nível de automação. Essa automação só é possível com grandes quantidades de monitoramento. Para medir todos os aspectos dos recursos implantados, crie métricas, alarmes e painéis. Não pare de medir coisas como uso da CPU e espaço em disco. Além disso, registre suas métricas de negócios e use essas medidas para automatizar decisões de implantação, como reversões. A maioria das construções L2 AWS CDK tem métodos convenientes para ajudar você a criar métricas, como o `metricUserErrors()` método na classe DynamoDB.table.

AWS CDK referência

Esta seção contém informações de referência para o AWS Cloud Development Kit (AWS CDK).

Tópicos

Referência de API

A Referência da API contém informações sobre a AWS Construct Library e outras APIs fornecidas pela AWS Cloud Development Kit (AWS CDK). A maior parte da AWS Construct Library está contida em um único pacote chamado pelo TypeScript nome:`aws-cdk-lib`. O nome real do pacote varia de acordo com o idioma. Versões separadas da referência da API são fornecidas para cada linguagem de programação compatível.

A referência da API CDK é organizada em submódulos. Há um ou mais submódulos para cada um AWS service (Serviço da AWS).

Cada submódulo tem uma visão geral que inclui informações sobre como usar suas APIs. Por exemplo, a visão geral do S3 demonstra como definir a criptografia padrão em um bucket do Amazon Simple Storage Service (Amazon S3).

AWS CDK controle de versão

Este tópico fornece informações de referência sobre como o AWS Cloud Development Kit (AWS CDK) lida com o controle de versão.

Os números das versões consistem em três partes numéricas da versão: principal. menor. corrija e adira estritamente ao modelo de controle de versão semântico. Isso significa que as alterações significativas nas APIs estáveis estão limitadas às versões principais.

As versões secundárias e de patch são compatíveis com versões anteriores. O código escrito em uma versão anterior com a mesma versão principal pode ser atualizado para uma versão mais recente dentro da mesma versão principal. Ele também continuará a ser construído e executado, produzindo a mesma saída.

Tópicos

AWS CDKCLIcompatibilidade

O AWS CDK CLI é sempre compatível com bibliotecas de construção com um número de versão semanticamente menor ou igual. Portanto, é sempre seguro atualizá-lo AWS CDK CLI na mesma versão principal.

Nem sempre AWS CDK CLI é compatível com bibliotecas de construção de uma versão semanticamente superior. A compatibilidade depende se a mesma versão do esquema de montagem em nuvem é empregada pelos dois componentes. A AWS CDK estrutura gera uma montagem em nuvem durante a síntese e a AWS CDK CLI consome para implantação. O esquema que define o formato da montagem da nuvem é estritamente especificado e versionado.

AWS as bibliotecas de construção usando uma determinada versão do esquema de montagem em nuvem são compatíveis com AWS CDK CLI as versões que usam essa versão do esquema ou posterior. Isso pode incluir versões do AWS CDK CLI que são anteriores à versão de uma determinada biblioteca de construção.

Quando a versão de montagem em nuvem exigida pela biblioteca de construção não é compatível com a versão suportada pelo AWS CDK CLI, você recebe uma mensagem de erro como a seguinte:

```
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.
```

Para resolver esse erro, atualize o AWS CDK CLI para uma versão compatível com a versão de montagem em nuvem necessária ou para a versão mais recente disponível. A alternativa (rebaixar os módulos da biblioteca de construção que seu aplicativo usa) geralmente não é recomendada.

> **Note**
>
> Para obter mais detalhes sobre o esquema de montagem em nuvem, consulte Controle de versão do Cloud Assembly.

AWS Controle de versão da Construct Library

Os módulos na AWS Construct Library passam por vários estágios à medida que são desenvolvidos, do conceito à API madura. Estágios diferentes oferecem vários graus de estabilidade da API nas versões subsequentes do AWS CDK.

As APIs na AWS CDK biblioteca principal,`aws-cdk-lib`, são estáveis e a biblioteca é totalmente versionada semanticamente. Este pacote inclui construções AWS CloudFormation (L1) para todos os AWS serviços e todos os módulos estáveis de nível superior (L2 e L3). (Também inclui as classes principais do CDK, como `App` e`Stack`). As APIs não serão removidas desse pacote (embora possam estar obsoletas) até a próxima versão principal do CDK. Nenhuma API individual jamais terá alterações significativas. Quando uma alteração significativa for necessária, uma API totalmente nova será adicionada.

Novas APIs em desenvolvimento para um serviço já incorporado `aws-cdk-lib` são identificadas usando um `Beta`*`N`* sufixo, que `N` começa em 1 e é incrementado a cada alteração significativa na nova API. `Beta`*`N`* As APIs nunca são removidas, apenas obsoletas. Portanto, seu aplicativo existente continua funcionando com versões mais recentes do. `aws-cdk-lib` Quando a API é considerada estável, uma nova API sem o `Beta`*`N`* sufixo é adicionada.

Quando as APIs de nível superior (L2 ou L3) começam a ser desenvolvidas para um AWS serviço que antes tinha apenas APIs L1, essas APIs são inicialmente distribuídas em um pacote separado. O nome desse pacote tem um sufixo “Alpha” e sua versão corresponde à primeira versão compatível com, com uma `alpha` subversão. `aws-cdk-lib` Quando o módulo oferece suporte aos casos de uso pretendidos, suas APIs são adicionadas`aws-cdk-lib`.

Estabilidade de vinculação de linguagem

Com o tempo, poderemos adicionar suporte ao AWS CDK para linguagens de programação adicionais. Embora a API descrita em todos os idiomas seja a mesma, a forma como a API é expressa varia de acordo com o idioma e pode mudar à medida que o suporte ao idioma evolui. Por esse motivo, as vinculações de linguagem são consideradas experimentais por um tempo até serem consideradas prontas para uso em produção.

Language	Stability
TypeScript	Stable
JavaScript	Stable

Language	Stability
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK tutoriais

Esta seção contém tutoriais para o. AWS Cloud Development Kit (AWS CDK)

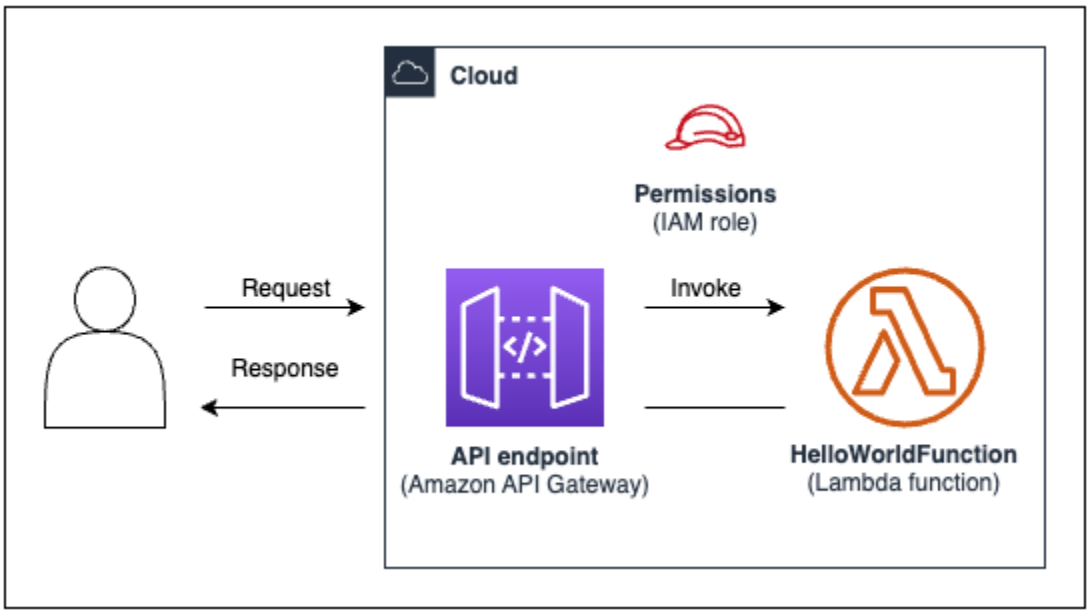
Tópicos

Crie um aplicativo Hello World sem servidor

Neste tutorial, você usa o AWS Cloud Development Kit (AWS CDK) para criar um Hello World aplicativo simples sem servidor que implementa um back-end de API básico criando o seguinte:

- Amazon API Gateway REST API — fornece um endpoint HTTP que é usado para invocar sua função por meio de uma HTTP GET solicitação.
- AWS Lambda function — Função que retorna uma `Hello World!` mensagem quando invocada com o HTTP endpoint.
- Integrações e permissões — Detalhes de configuração e permissões para que seus recursos interajam entre si e realizem ações, como gravar registros na Amazon CloudWatch.

O diagrama a seguir mostra os componentes deste aplicativo:



Para este tutorial, você concluirá o seguinte:

1. Crie um AWS CDK projeto.
2. Defina uma função Lambda e a API REST do API Gateway usando construções L2 da Construct Library. AWS
3. Implante seu aplicativo no Nuvem AWS.
4. Interaja com seu aplicativo no Nuvem AWS.
5. Exclua a amostra do aplicativo do Nuvem AWS.

Pré-requisitos

Antes de começar este tutorial, conclua as seguintes etapas:

- Crie um Conta da AWS e tenha o AWS Command Line Interface (AWS CLI) instalado e configurado.
- Instale Node.js npm e.
- Instale o CDK Toolkit globalmente, usando. `npm install -g aws-cdk`

Para ter mais informações, consulte Começando com o AWS CDK.

Também recomendamos uma compreensão básica do seguinte:

- O que é o AWS CDK?para uma introdução básica ao AWS CDK.
- AWS CDK conceitospara uma visão geral dos principais conceitos do AWS CDK.

Etapa 1: criar um projeto CDK

Nesta etapa, você cria um novo projeto CDK usando o AWS CDK CLI `cdk init` comando.

Para criar um projeto CDK

1. Em um diretório inicial de sua escolha, crie e navegue até um diretório de projeto chamado `cdk-hello-world` em sua máquina:

```
$ mkdir cdk-hello-world && cd cdk-hello-world
```

2. Use o `cdk init` comando para criar um novo projeto na linguagem de programação de sua preferência:

   TypeScript

   ```
   $ cdk init --language typescript
   ```

   Instale AWS CDK bibliotecas:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   JavaScript

   ```
   $ cdk init --language javascript
   ```

   Instale AWS CDK bibliotecas:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   Python

   ```
   $ cdk init --language python
   ```

   Ative o ambiente virtual:

   ```
   $ source .venv/bin/activate
   ```

   Instale AWS CDK bibliotecas e dependências do projeto:

   ```
   (.venv)$ python3 -m pip install -r requirements.txt
   ```

   Java

   ```
   $ cdk init --language java
   ```

   Instale AWS CDK bibliotecas e dependências do projeto:

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

Instale AWS CDK bibliotecas e dependências do projeto:

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

Instale as dependências do projeto:

```
$ go mod tidy
```

O CDK CLI cria um projeto com a seguinte estrutura:

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
```

O CDK cria CLI automaticamente um aplicativo CDK que contém uma única pilha. A instância do aplicativo CDK é criada a partir da App classe. A seguir está uma parte do seu arquivo de aplicativo CDK:

TypeScript

Localizado em`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

Localizado em`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

Localizado em`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

Localizado em`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#

Localizado em`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

Localizado em`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
}
```

Etapa 2: Crie sua função Lambda

Em seu projeto CDK, crie um `lambda` diretório que inclua um novo `hello.js` arquivo. Veja um exemplo a seguir:

TypeScript

Na raiz do seu projeto, execute o seguinte:

```
$ mkdir lambda && cd lambda
```

```
$ touch hello.js
```

Agora, o seguinte deve ser adicionado ao seu projeto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

JavaScript

Na raiz do seu projeto, execute o seguinte:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Agora, o seguinte deve ser adicionado ao seu projeto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

Na raiz do seu projeto, execute o seguinte:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Agora, o seguinte deve ser adicionado ao seu projeto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

Na raiz do seu projeto, execute o seguinte:

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

Agora, o seguinte deve ser adicionado ao seu projeto CDK:

```
cdk-hello-world
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

Na raiz do seu projeto, execute o seguinte:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Agora, o seguinte deve ser adicionado ao seu projeto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

Na raiz do seu projeto, execute o seguinte:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Agora, o seguinte deve ser adicionado ao seu projeto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

> **Note**
>
> Para manter este tutorial simples, usamos uma função JavaScript Lambda para todas as linguagens de programação CDK.

Defina sua função Lambda adicionando o seguinte ao arquivo recém-criado:

```
exports.handler = async (event) => {
    return {
        statusCode: 200,
        headers: { "Content-Type": "text/plain" },
        body: JSON.stringify({ message: "Hello, World!" }),
    };
};
```

Etapa 3: defina suas construções

Nesta etapa, você definirá seus recursos do Lambda e do API Gateway usando construções AWS CDK L2.

Abra o arquivo do projeto que define sua pilha de CDK. Você modificará esse arquivo para definir suas construções. Veja a seguir um exemplo do seu arquivo de pilha inicial:

TypeScript

Localizado em`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

Localizado em`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

Localizado em`cdk_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

Localizado em`src/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#

Localizado em`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

Localizado em`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
}
```

Nesse arquivo, o AWS CDK está fazendo o seguinte:

- Sua instância de pilha do CDK é instanciada a partir da classe. `Stack`
- A classe `Constructs` base é importada e fornecida como escopo ou pai da sua instância de pilha.

Defina seu recurso de função Lambda

Para definir seu recurso de função Lambda, você importa e usa a construção `aws-lambda` L2 da Construct Library. AWS

Modifique seu arquivo de pilha da seguinte forma:

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**
>
> Importamos o `aws_lambda` módulo `_lambda` porque `lambda` é um identificador embutido no. 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
}

// ...
```

Aqui, você cria um recurso de função Lambda e define as seguintes propriedades:

- `runtime`— O ambiente em que a função é executada. Aqui, usamos a Node.js versão20.x.
- `code`— O caminho para o código da função em sua máquina local.
- `handler`— O nome do arquivo específico que contém o código da função.

Defina seu REST API recurso do API Gateway

Para definir seu REST API recurso do API Gateway, você importa e usa a construção [`aws-apigateway`](#) L2 da AWS Construct Library.

Modifique seu arquivo de pilha da seguinte forma:

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
}

// ...
```

Aqui, você cria um REST API recurso do API Gateway, junto com o seguinte:

- Uma integração entre a REST API e sua função Lambda, permitindo que a API invoque sua função. Isso inclui a criação de um recurso de permissão do Lambda.
- Um novo recurso ou caminho chamado `hello` que é adicionado à raiz do endpoint da API. Isso cria um novo endpoint que é adicionado `/hello` à sua baseURL.
- Um método GET para o `hello` recurso. Quando uma solicitação GET é enviada ao `/hello` endpoint, a função Lambda é invocada e sua resposta é retornada.

Etapa 4: Preparar seu aplicativo para implantação

Nesta etapa, você prepara seu aplicativo para implantação criando, se necessário, e executando a validação básica com o AWS CDK CLI `cdk synth` comando.

Se necessário, crie seu aplicativo:

TypeScript

Na raiz do seu projeto, execute o seguinte:

```
$ npm run build
```

JavaScript

A construção não é necessária.

Python

A construção não é necessária.

Java

Na raiz do seu projeto, execute o seguinte:

```
$ mvn package
```

C#

Na raiz do seu projeto, execute o seguinte:

```
$ dotnet build src
```

Go

Na raiz do seu projeto, execute o seguinte:

```
$ go build
```

Execute `cdk synth` para sintetizar um AWS CloudFormation modelo a partir do seu código CDK. Ao usar construções L2, muitos dos detalhes de configuração necessários AWS CloudFormation para facilitar a interação entre sua função Lambda REST API são provisionados para você pelo. AWS CDK

Na raiz do seu projeto, execute o seguinte:

```
$ cdk synth
```

> **Note**
>
> Se você receber um erro como o seguinte, verifique se você está no `cdk-hello-world` diretório e tente novamente:
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

Se for bem-sucedido, o AWS CDK CLI exibirá o AWS CloudFormation modelo em YAML formato no prompt de comando. Um modelo JSON formatado também é salvo no `cdk.out` diretório.

Veja a seguir um exemplo de saída do AWS CloudFormation modelo:

AWS CloudFormation modelo

```
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.
```

Ao usar construções L2, você define algumas propriedades para configurar seus recursos e usa métodos auxiliares para integrá-los. O AWS CDK configura a maioria dos AWS CloudFormation recursos e propriedades necessários para provisionar seu aplicativo.

Etapa 5: implantar a aplicação

Nesta etapa, você usa o AWS CDK CLI `cdk deploy` comando para implantar seu aplicativo. Ele AWS CDK trabalha com o AWS CloudFormation serviço para provisionar seus recursos.

> **Important**
>
> Você deve executar uma única inicialização do seu AWS ambiente antes da implantação. Para obter instruções, consulte Bootstrap em seu ambiente.

Na raiz do seu projeto, execute o seguinte. Confirme as alterações, se solicitado:

```
$ cdk deploy

#  Synthesis time: 2.44s

...

Do you wish to deploy these changes (y/n)? y
```

Quando a implantação for concluída, ele AWS CDK CLI exibirá o URL do seu endpoint. Copie esse URL para a próxima etapa. Veja um exemplo a seguir:

```
...
#  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
...
```

Etapa 6: Interaja com seu aplicativo

Nesta etapa, você inicia uma solicitação GET para o endpoint da API e recebe a resposta da função Lambda.

Localize o URL do endpoint da etapa anterior e adicione o `/hello` caminho. Em seguida, usando seu navegador ou prompt de comando, envie uma solicitação GET para seu endpoint. Veja um exemplo a seguir:

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello
{"message":"Hello World!"}%
```

Parabéns, você criou, implantou e interagiu com sucesso com seu aplicativo usando o! AWS CDK

Etapa 7: excluir seu aplicativo

Nesta etapa, você usa o AWS CDK CLI para excluir seu aplicativo do Nuvem AWS.

Para excluir seu aplicativo, execute`cdk destroy`. Quando solicitado, confirme sua solicitação para excluir o aplicativo:

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
 #  CdkHelloWorldStack: destroyed
```

Solução de problemas

Erro: {“message”: “Erro interno do servidor”}%

Ao invocar a função Lambda implantada, você recebe esse erro. Esse erro pode ocorrer por vários motivos.

Para solucionar problemas adicionais

Use o AWS CLI para invocar sua função Lambda.

1. Modifique seu arquivo de pilha para capturar o valor de saída do nome da função Lambda implantada. Veja um exemplo a seguir:

```
...

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. Implante seu aplicativo novamente. AWS CDK CLIIsso exibirá o valor do nome da função Lambda implantada:

```
$ cdk deploy

#  Synthesis time: 0.29s
...
 #  CdkHelloWorldStack

#  Deployment time: 20.36s

Outputs:
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. Use o AWS CLI para invocar sua função Lambda no e enviar Nuvem AWS a resposta para um arquivo de texto:

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-identifier output.txt
```

4. Verifique `output.txt` para ver seus resultados.

Possível causa: o recurso API Gateway está definido incorretamente em seu arquivo de pilha.

Se `output.txt` mostrar uma resposta bem-sucedida da função Lambda, o problema pode estar na forma como você definiu sua API REST do API Gateway. Ele AWS CLI invoca seu Lambda diretamente, não por meio de seu endpoint. Verifique seu código para garantir que ele corresponda a este tutorial. Em seguida, implante novamente.

Possível causa: o recurso Lambda está definido incorretamente em seu arquivo de pilha.

Se `output.txt` retornar um erro, o problema pode estar na forma como você definiu sua função Lambda. Verifique seu código para garantir que ele corresponda a este tutorial. Em seguida, implante novamente.

Crie um aplicativo com várias pilhas

Você pode criar um AWS Cloud Development Kit (AWS CDK) aplicativo contendo várias pilhas. Quando você implanta o AWS CDK aplicativo, cada pilha se torna seu próprio AWS CloudFormation modelo. Você também pode sintetizar e implantar cada pilha individualmente usando o comando. AWS CDK CLI `cdk deploy`

Este tutorial aborda o seguinte:

- Como estender a `Stack` classe para aceitar novas propriedades ou argumentos.
- Como usar propriedades para determinar quais recursos a pilha contém e suas configurações.
- Como instanciar várias pilhas dessa classe.

O exemplo neste tópico usa uma propriedade booleana chamada `encryptBucket` (Python`encrypt_bucket`:). Ela indica se um bucket do Amazon S3 deve ser criptografado. Nesse caso, a pilha habilita a criptografia usando uma chave gerenciada por AWS Key Management Service (AWS KMS). O aplicativo cria duas instâncias dessa pilha, uma com criptografia e outra sem.

Tópicos

Antes de começar

Primeiro, instale o Node.js e as ferramentas de linha de AWS CDK comando, caso ainda não tenha feito isso. Para mais detalhes, consulte Começando com o AWS CDK.

Em seguida, crie um AWS CDK projeto inserindo os seguintes comandos na linha de comando.

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
```

Você pode importar o projeto Maven resultante para o seu Java IDE.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

Você pode abrir o arquivo `src/Pipeline.sln` no Visual Studio.

Adicionar parâmetro opcional

O `props` argumento do `Stack` construtor preenche a interface. `StackProps` Neste exemplo, queremos que a pilha aceite uma propriedade adicional para nos dizer se devemos criptografar o bucket do Amazon S3. Devemos criar uma interface ou classe que inclua a propriedade. Isso permite que o compilador garanta que a propriedade tenha um valor booleano e habilite o preenchimento automático para ela em seu IDE.

Então, abra o arquivo fonte indicado em seu IDE ou editor e adicione a nova interface, classe ou argumento. O código deve ficar assim após as alterações. As linhas que adicionamos são mostradas em negrito.

TypeScript

Arquivo: `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

Arquivo: `lib/multistack-stack.js`

JavaScript não tem um recurso de interface; não precisamos adicionar nenhum código.

```
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

Arquivo: `multistack/multistack_stack.py`

O Python não tem um recurso de interface, então ampliaremos nossa pilha para aceitar a nova propriedade adicionando um argumento de palavra-chave.

```
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

Arquivo: `src/main/java/com/myorg/MultistackStack.java`

É mais complicado do que realmente queremos estender um tipo de adereços em Java. Em vez disso, escreva o construtor da pilha para aceitar um parâmetro booleano opcional. Como `props` é um argumento opcional, escreveremos um construtor adicional que permite ignorá-lo. O padrão será`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#

Arquivo: `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
        }
    }
}
```

A nova propriedade é opcional. Se `encryptBucket` (Python:`encrypt_bucket`) não estiver presente, seu valor é`undefined`, ou o equivalente local. O bucket não será criptografado por padrão.

Defina a classe da pilha

Agora vamos definir nossa classe de pilha, usando nossa nova propriedade. Faça com que o código tenha a seguinte aparência. O código que você precisa adicionar ou alterar é mostrado em negrito.

TypeScript

Arquivo: `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

Arquivo: `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

Arquivo: `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

Arquivo: `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#

Arquivo: `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
                });
            }
        }
    }
}
```

Crie duas instâncias de pilha

Agora vamos adicionar o código para instanciar duas pilhas separadas. Como antes, as linhas de código mostradas em negrito são as que você precisa adicionar. Exclua a `MultistackStack` definição existente.

TypeScript

Arquivo: `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

Arquivo: `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

Arquivo: `./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

Arquivo: `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#

Arquivo: 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();
        }
    }
}
```

Esse código usa a nova propriedade `encryptBucket` (Python:`encrypt_bucket`) na `MultistackStack` classe para instanciar o seguinte:

- Uma pilha com um bucket criptografado do Amazon S3 na `us-east-1` AWS região.
- Uma pilha com um bucket Amazon S3 não criptografado na região. `us-west-1` AWS

Sintetize e implante a pilha

Agora você pode implantar pilhas a partir do aplicativo. Primeiro, sintetize um AWS CloudFormation modelo para `MyEastCdkStack` ——empilhe. `us-east-1` Essa é a pilha com o bucket S3 criptografado.

```
$ cdk synth MyEastCdkStack
```

Para implantar essa pilha em sua AWS conta, execute um dos comandos a seguir. O primeiro comando usa seu AWS perfil padrão para obter as credenciais para implantar a pilha. O segundo usa um perfil que você especifica. Para *`PROFILE_NAME`*, substitua o nome de um AWS CLI perfil que contém as credenciais apropriadas para implantação na região. `us-east-1` AWS

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

Limpeza

Para evitar cobranças pelos recursos que você implantou, destrua a pilha usando o comando a seguir.

```
cdk destroy MyEastCdkStack
```

A operação de destruição falhará se houver algo armazenado no bucket da pilha. Não deveria existir se você apenas seguiu as instruções deste tópico. Mas se você colocou algo no bucket, você deve excluir o conteúdo do bucket antes de destruir a pilha. (Não exclua o bucket em si.) Use o AWS Management Console ou o AWS CLI para excluir o conteúdo do bucket.

Exemplos

Este tópico contém os seguintes exemplos:

- Crie um aplicativo Hello World sem servidorCria um aplicativo sem servidor usando Lambda, API Gateway e Amazon S3.
- Criando um serviço AWS Fargate usando o AWS CDKCria um serviço Amazon ECS Fargate a partir de uma imagem em. DockerHub

Criando um serviço AWS Fargate usando o AWS CDK

Este exemplo mostra como criar um serviço AWS Fargate executado em um cluster do Amazon Elastic Container Service (Amazon ECS) que é liderado por um Application Load Balancer voltado para a Internet a partir de uma imagem no Amazon ECR.

O Amazon ECS é um serviço de gerenciamento de contêineres altamente escalável e rápido que facilita a execução, a interrupção e o gerenciamento de contêineres do Docker em um cluster. Você pode hospedar seu cluster em uma infraestrutura sem servidor gerenciada pelo Amazon ECS lançando seus serviços ou tarefas usando o tipo de execução Fargate. Para obter mais controle, você pode hospedar suas tarefas em um cluster de instâncias do Amazon Elastic Compute Cloud (Amazon EC2) que você gerencia usando o tipo de execução do Amazon EC2.

Este tutorial mostra como iniciar alguns serviços usando o tipo de inicialização Fargate. Se você usou o AWS Management Console para criar um serviço Fargate, sabe que há muitas etapas a serem seguidas para realizar essa tarefa. AWS tem vários tutoriais e tópicos de documentação que orientam você na criação de um serviço Fargate, incluindo:

- Como implantar contêineres Docker - AWS
- Configuração com o Amazon ECS
- Começando a usar o Amazon ECS usando o Fargate

Este exemplo cria um serviço Fargate semelhante em AWS CDK código.

A construção do Amazon ECS usada neste tutorial ajuda você a usar AWS serviços fornecendo os seguintes benefícios:

- Configura automaticamente um balanceador de carga.

- Abre automaticamente um grupo de segurança para balanceadores de carga. Isso permite que os balanceadores de carga se comuniquem com as instâncias sem que você crie explicitamente um grupo de segurança.
- Ordena automaticamente a dependência entre o serviço e o balanceador de carga vinculado a um grupo-alvo, onde ele AWS CDK impõe a ordem correta de criação do ouvinte antes da criação de uma instância.
- Configura automaticamente os dados do usuário em grupos de escalabilidade automática. Isso cria a configuração correta para associar um cluster às AMIs.
- Valida as combinações de parâmetros com antecedência. Isso expõe AWS CloudFormation problemas mais cedo, economizando tempo de implantação. Por exemplo, dependendo da tarefa, é fácil configurar incorretamente as configurações de memória. Anteriormente, você não encontrava um erro até implantar seu aplicativo. Mas agora eles AWS CDK podem detectar uma configuração incorreta e emitir um erro ao sintetizar seu aplicativo.
- Adiciona automaticamente permissões para o Amazon Elastic Container Registry (Amazon ECR) se você usar uma imagem do Amazon ECR.
- Dimensiona automaticamente. Ele AWS CDK fornece um método para que você possa escalar automaticamente as instâncias ao usar um cluster do Amazon EC2. Isso acontece automaticamente quando você usa uma instância em um cluster Fargate.

  Além disso, AWS CDK evita que uma instância seja excluída quando o escalonamento automático tenta eliminar uma instância, mas uma tarefa está em execução ou está programada nessa instância.

  Anteriormente, era necessário criar uma função Lambda para ter essa funcionalidade.
- Fornece suporte de ativos, para que você possa implantar uma fonte da sua máquina no Amazon ECS em uma única etapa. Anteriormente, para usar uma fonte de aplicativo, era necessário realizar várias etapas manuais, como fazer o upload para o Amazon ECR e criar uma imagem do Docker.

Consulte o ECS para obter detalhes.

> **Important**
>
> As `ApplicationLoadBalancedFargateService` construções que usaremos incluem vários AWS componentes, alguns dos quais têm custos não triviais se deixados

provisionados em sua AWS conta, mesmo que você não os use. Certifique-se de limpar (cdk destroy) depois de concluir este exemplo.

Criando o diretório e inicializando o AWS CDK

Vamos começar criando um diretório para armazenar o AWS CDK código e, em seguida, criando um AWS CDK aplicativo nesse diretório.

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
```

Agora você pode importar o projeto Maven para o seu IDE.

C#

```
mkdir MyEcsConstruct
```

```
cd MyEcsConstruct
cdk init --language csharp
```

Agora você pode abrir `src/MyEcsConstruct.sln` no Visual Studio.

Execute o aplicativo e confirme se ele cria uma pilha vazia.

```
cdk synth
```

Crie um serviço Fargate

Há duas maneiras diferentes de executar suas tarefas de contêiner com o Amazon ECS:

- Use o tipo de `Fargate` lançamento, em que o Amazon ECS gerencia as máquinas físicas nas quais seus contêineres estão sendo executados para você.
- Use o tipo de `EC2` lançamento, onde você faz o gerenciamento, como especificar o escalonamento automático.

Neste exemplo, criaremos um serviço Fargate executado em um cluster ECS liderado por um Application Load Balancer voltado para a Internet.

Adicione as seguintes importações do módulo AWS Construct Library ao arquivo indicado.

TypeScript

Arquivo: `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

Arquivo: `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

Arquivo: `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

Arquivo: `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#

Arquivo: `src/MyEcsConstruct/MyEcsConstructStack.cs`

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

Substitua o comentário no final do construtor pelo código a seguir.

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 ec2.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
                }
            );
```

Salve-o e certifique-se de que ele seja executado e crie uma pilha.

```
cdk synth
```

A pilha tem centenas de linhas, então não a mostraremos aqui. A pilha deve conter uma instância padrão, uma sub-rede privada e uma sub-rede pública para as três zonas de disponibilidade e um grupo de segurança.

Implante a pilha.

```
cdk deploy
```

AWS CloudFormation exibe informações sobre as dezenas de etapas necessárias ao implantar seu aplicativo.

É muito fácil criar um serviço Amazon ECS baseado em Fargate para executar uma imagem do Docker.

Limpeza

Para evitar AWS cobranças inesperadas, destrua sua AWS CDK pilha depois de concluir este exercício.

```
cdk destroy
```

AWS CDK exemplos

Para ver mais exemplos de AWS CDK pilhas e aplicativos em sua linguagem de programação compatível favorita, consulte o repositório AWS CDK Examples em. GitHub

AWS CDK ferramentas

Esta seção contém informações sobre as AWS CDK ferramentas listadas abaixo.

Tópicos

AWS CDK Kit de ferramentas (**cdk**comando)

O AWS CDK Toolkit, o `cdk` comando da CLI, é a principal ferramenta para interagir com seu aplicativo. AWS CDK Ele executa seu aplicativo, interroga o modelo de aplicativo que você definiu e produz e implanta os AWS CloudFormation modelos gerados pelo. AWS CDK Ele também fornece outros recursos úteis para criar e trabalhar com AWS CDK projetos. Este tópico contém informações sobre casos de uso comuns do CDK Toolkit.

O AWS CDK kit de ferramentas é instalado com o Node Package Manager. Na maioria dos casos, recomendamos instalá-lo globalmente.

```
npm install -g aws-cdk             # install latest version
npm install -g aws-cdk@X.YY.Z      # install specific version
```

> **Tip**
>
> Se você trabalha regularmente com várias versões do AWS CDK, considere instalar uma versão correspondente do AWS CDK Toolkit em projetos individuais do CDK. Para fazer isso, omita `-g` do `npm install` comando. Em seguida, use `npx aws-cdk` para invocá-lo. Isso executa a versão local, se houver, voltando para uma versão global, se não existir.

Comandos do kit de ferramentas

Todos os comandos do CDK Toolkit começam com`cdk`, seguido por um subcomando (`list`,, `synthesize``deploy`, etc.). Alguns subcomandos têm uma versão mais curta (`ls`,`synth`, etc.) que é

equivalente. As opções e os argumentos seguem o subcomando em qualquer ordem. Os comandos disponíveis estão resumidos aqui.

Command	Função
`cdk list` (`ls`)	Lista as pilhas no aplicativo
`cdk synthesize` (`synth`)	Sintetiza e imprime o CloudFormation modelo para uma ou mais pilhas especificadas
`cdk bootstrap`	Implanta a pilha de preparação do CDK Toolkit; consulte the section called “Bootstrapping”
`cdk deploy`	Implanta uma ou mais pilhas especificadas
`cdk destroy`	Destrói uma ou mais pilhas especificadas
`cdk diff`	Compara a pilha especificada e suas dependências com as pilhas implantadas ou com um modelo local CloudFormation
`cdk import`	Usa importações CloudFormation de recursos para colocar os recursos existentes em uma pilha gerenciada pelo CDK
`cdk metadata`	Exibe metadados sobre a pilha especificada
`cdk init`	Cria um novo projeto CDK no diretório atual a partir de um modelo especificado
`cdk context`	Gerencia valores de contexto em cache
`cdk docs` (`doc`)	Abre a referência da API CDK em seu navegador
`cdk doctor`	Verifica se há possíveis problemas em seu projeto CDK

Para ver as opções disponíveis para cada comando, consultethe section called “Ajuda integrada”.

Especificando opções e seus valores

As opções da linha de comando começam com dois hífens ()`--`. Algumas opções usadas com frequência têm sinônimos de uma única letra que começam com um único hífen (por exemplo, `--app` tem um sinônimo`-a`). A ordem das opções em um comando do AWS CDK Toolkit não é importante.

Todas as opções aceitam um valor, que deve seguir o nome da opção. O valor pode ser separado do nome por espaço em branco ou por um sinal de igual. = As duas opções a seguir são equivalentes.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

Algumas opções são bandeiras (booleanos). Você pode especificar `true` ou `false` como seu valor. Se você não fornecer um valor, o valor será consideradotrue. Você também pode prefixar o nome da opção com `no-` para `false` sugerir.

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

Algumas opções, a saber`--context`,`--parameters`,`--plugin`, e `--tags``--trust`, podem ser especificadas mais de uma vez para especificar vários valores. Eles são indicados como sendo `[array]` digitados na ajuda do CDK Toolkit. Por exemplo: .

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

Ajuda integrada

O AWS CDK kit de ferramentas tem ajuda integrada. Você pode ver a ajuda geral sobre o utilitário e uma lista dos subcomandos fornecidos emitindo:

```
cdk --help
```

Para ver a ajuda de um subcomando específico, por exemplo`deploy`, especifique-o antes do `--help` sinalizador.

```
cdk deploy --help
```

Problema `cdk version` para exibir a versão do AWS CDK kit de ferramentas. Forneça essas informações ao solicitar suporte.

Relatórios de versão

Para obter informações sobre como o AWS CDK é usado, as construções usadas pelos AWS CDK aplicativos são coletadas e relatadas usando um recurso identificado como`AWS::CDK::Metadata`. Esse recurso é adicionado aos AWS CloudFormation modelos e pode ser facilmente revisado. Essas informações também podem ser usadas AWS para identificar pilhas usando uma construção com problemas conhecidos de segurança ou confiabilidade. Também pode ser usado para contatar seus usuários com informações importantes.

> **Note**
>
> Antes da versão 1.93.0, eles AWS CDK relatavam os nomes e versões dos módulos carregados durante a síntese, em vez das construções usadas na pilha.

Por padrão, ele AWS CDK relata o uso de construções nos seguintes módulos NPM que são usados na pilha:

- AWS CDK módulo principal
- AWS Construir módulos de biblioteca
- AWS Módulo Solutions Constructs
- AWS Módulo do kit de implantação do Render Farm

O `AWS::CDK::Metadata` recurso se parece com o seguinte.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
  Properties:
    Analytics:
 "v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/
```

```
XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz
+AqqLDt2Y3YMnLYWwAAAA="
```

A `Analytics` propriedade é uma lista compactada com gzip, codificada em base64 e codificada por prefixo das construções na pilha.

Para optar por não receber relatórios de versão, use um dos seguintes métodos:

- Use o cdk comando com o --no-version-reporting argumento para optar por não receber um único comando.

  ```
  cdk --no-version-reporting synth
  ```

  Lembre-se de que o AWS CDK kit de ferramentas sintetiza novos modelos antes da implantação, portanto, você também deve adicionar comandos. `--no-version-reporting` `cdk deploy`
- versionReportingDefina como false em `./cdk.json` ou`~/.cdk.json`. Isso é desativado, a menos que você opte por meio de uma especificação `--version-reporting` em um comando individual.

  ```
  {
    "app": "...",
    "versionReporting": false
  }
  ```

Autenticação com AWS

Há diferentes maneiras de configurar o acesso programático aos AWS recursos, dependendo do ambiente e do AWS acesso disponível para você.

Para escolher seu método de autenticação e configurá-lo para o CDK Toolkit, consulte Autenticação e acesso no Guia de referência de AWS SDKs e ferramentas.

A abordagem recomendada para novos usuários que se desenvolvem localmente, que não recebem um método de autenticação por parte do empregador, é configurar AWS IAM Identity Center. Esse método inclui a instalação do AWS CLI para facilitar a configuração e entrar regularmente no portal de AWS acesso. Se você escolher esse método, seu ambiente deverá conter os seguintes elementos depois de concluir o procedimento de autenticação do IAM Identity Center no Guia de referência de ferramentas e SDKs da AWS :

- O AWS CLI, que você usa para iniciar uma sessão do portal de AWS acesso antes de executar seu aplicativo.
- Um AWS`config`arquivo compartilhado com um `[default]` perfil com um conjunto de valores de configuração que podem ser referenciados a AWS CDK partir do. Para encontrar a localização desse arquivo, consulte Localização dos arquivos compartilhados no Guia de referência de ferramentas e SDKs da AWS .
- O arquivo `config` compartilhado define a configuração do `region`. Isso define o padrão que o CDK Toolkit AWS CDK e Região da AWS o CDK Toolkit usam para AWS solicitações.
- O CDK Toolkit usa a configuração do provedor de token SSO do perfil para adquirir credenciais antes de enviar solicitações para. AWS O `sso_role_name` valor, que é uma função do IAM conectada a um conjunto de permissões do IAM Identity Center, deve permitir o acesso ao Serviços da AWS usado em seu aplicativo.

  O arquivo `config` de amostra a seguir mostra um perfil padrão configurado com o provedor de token de SSO. A configuração `sso_session` do perfil se refere à seção do `sso-session`. A `sso-session` seção contém configurações para iniciar uma sessão do portal de AWS acesso.

```
[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
```

Iniciar uma sessão do portal de AWS acesso

Antes de acessar Serviços da AWS, você precisa de uma sessão ativa do portal de AWS acesso para que o CDK Toolkit use a autenticação do IAM Identity Center para resolver as credenciais. Dependendo da duração da sessão configurada, seu acesso acabará expirando e o CDK Toolkit encontrará um erro de autenticação. Execute o seguinte comando no AWS CLI para entrar no portal de AWS acesso.

```
aws sso login
```

Se a configuração do seu provedor de token SSO estiver usando um perfil nomeado em vez do perfil padrão, o comando seráaws sso login --profile *NAME*. Especifique também esse perfil ao emitir cdk comandos usando a --profile opção ou a variável de AWS_PROFILE ambiente.

Para testar se você já tem uma sessão ativa, execute o AWS CLI comando a seguir.

```
aws sts get-caller-identity
```

A resposta a esse comando deve relatar a conta do Centro de Identidade do IAM e o conjunto de permissões configurados no arquivo compartilhado config.

> **Note**
>
> Se você já tiver uma sessão ativa do portal de AWS acesso e executá-laaws sso login, não será necessário fornecer credenciais.
> O processo de login pode solicitar que você permita o AWS CLI acesso aos seus dados. Como o AWS CLI é construído sobre o SDK para Python, as mensagens de permissão podem conter variações do botocore nome.

Especificando a região e outras configurações

O CDK Toolkit precisa conhecer a AWS região em que você está implantando e como se autenticar. AWS Isso é necessário para operações de implantação e para recuperar valores de contexto durante a síntese. Juntas, sua conta e sua região compõem o ambiente.

A região pode ser especificada usando variáveis de ambiente ou em arquivos de configuração. Essas são as mesmas variáveis e arquivos usados por outras AWS ferramentas, como o AWS CLI e os vários AWS SDKs. O CDK Toolkit procura essas informações na seguinte ordem.

- A variável de ambiente AWS_DEFAULT_REGION.
- Um perfil nomeado definido no AWS config arquivo padrão e especificado usando a --profile opção nos cdk comandos.
- A [default] seção do AWS config arquivo padrão.

Além de especificar a AWS autenticação e uma região na `[default]` seção, você também pode adicionar uma ou mais `[profile NAME]` seções, onde *NOME* é o nome do perfil. Para obter mais informações sobre perfis nomeados, consulte Arquivos de configuração e credenciais compartilhados no Guia de referência de AWS SDKs e ferramentas.

O AWS `config` arquivo padrão está localizado em `~/.aws/config` (macOS/Linux) ou `%USERPROFILE%\.aws\config` (Windows). Para obter detalhes e localizações alternativas, consulte Localização dos arquivos compartilhados de configuração e credenciais no Guia de referência de AWS SDKs e ferramentas

O ambiente que você especifica no seu AWS CDK aplicativo usando a `env` propriedade da pilha é usado durante a síntese. Ele é usado para gerar um AWS CloudFormation modelo específico do ambiente e, durante a implantação, substitui a conta ou a região especificada por um dos métodos anteriores. Para ter mais informações, consulte the section called “Ambientes do”.

> **Note**
>
> O AWS CDK usa credenciais dos mesmos arquivos de origem de outras AWS ferramentas e SDKs, incluindo o. AWS Command Line Interface No entanto, eles AWS CDK podem se comportar de forma um pouco diferente dessas ferramentas. Ele usa a parte de AWS SDK for JavaScript baixo do capô. Para obter detalhes completos sobre a configuração de credenciais para o AWS SDK for JavaScript, consulte Configuração de credenciais.

Opcionalmente, você pode usar a opção `--role-arn` (ou`-r`) para especificar o ARN de uma função do IAM que deve ser usada para implantação. Essa função deve ser assumida pela AWS conta que está sendo usada.

Especificando o comando do aplicativo

Muitos recursos do CDK Toolkit exigem que um ou mais AWS CloudFormation modelos sejam sintetizados, o que, por sua vez, exige a execução do aplicativo. O AWS CDK suporta programas escritos em vários idiomas. Portanto, ele usa uma opção de configuração para especificar o comando exato necessário para executar seu aplicativo. Essa opção pode ser especificada de duas maneiras.

Primeiro, e mais comumente, ele pode ser especificado usando a `app` chave dentro do arquivo`cdk.json`. Isso está no diretório principal do seu AWS CDK projeto. O CDK Toolkit fornece

um comando apropriado ao criar um novo projeto com. `cdk init` Aqui está o `cdk.json` de um novo TypeScript projeto, por exemplo.

```
{
  "app": "npx ts-node bin/hello-cdk.ts"
}
```

O CDK Toolkit procura `cdk.json` no diretório de trabalho atual ao tentar executar seu aplicativo. Por isso, você pode manter um shell aberto no diretório principal do seu projeto para emitir comandos do CDK Toolkit.

O CDK Toolkit também procura a chave do aplicativo `~/.cdk.json` (ou seja, em seu diretório inicial) se não conseguir encontrá-la. `./cdk.json` Adicionar o comando do aplicativo aqui pode ser útil se você costuma trabalhar com o código CDK no mesmo idioma.

Se você estiver em algum outro diretório ou quiser executar seu aplicativo usando um comando diferente daquele em`cdk.json`, use a opção `--app` (ou`-a`) para especificá-lo.

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

Ao implantar, você também pode especificar um diretório contendo assemblies de nuvem sintetizados, como`cdk.out`, como o valor de. --app As pilhas especificadas são implantadas a partir desse diretório; o aplicativo não é sintetizado.

Especificando pilhas

Muitos comandos do CDK Toolkit (por exemplo,`cdk deploy`) funcionam em pilhas definidas em seu aplicativo. Se seu aplicativo contiver apenas uma pilha, o CDK Toolkit presume que você se refere a essa pilha se não especificar uma pilha explicitamente.

Caso contrário, você deverá especificar a pilha ou pilhas com as quais deseja trabalhar. Você pode fazer isso especificando as pilhas desejadas por ID individualmente na linha de comando. Lembre-se de que o ID é o valor especificado pelo segundo argumento quando você instancia a pilha.

```
cdk synth PipelineStack LambdaStack
```

Você também pode usar curingas para especificar IDs que correspondam a um padrão.

- `?`corresponde a qualquer caractere único
- `*`corresponde a qualquer número de caracteres (`*`sozinho corresponde a todas as pilhas)

- **corresponde a tudo em uma hierarquia

Você também pode usar a --all opção para especificar todas as pilhas.

Se seu aplicativo usa CDK Pipelines, o CDK Toolkit entende suas pilhas e estágios como uma hierarquia. Além disso, a --all opção e o * curinga correspondem apenas às pilhas de nível superior. Para combinar todas as pilhas, use**. Use também ** para indicar todas as pilhas em uma determinada hierarquia.

Ao usar curingas, coloque o padrão entre aspas ou escape dos curingas com. \ Caso contrário, seu shell pode tentar expandir o padrão para os nomes dos arquivos no diretório atual. Na melhor das hipóteses, isso não fará o que você espera; na pior das hipóteses, você pode implantar pilhas que não pretendia. Isso não é estritamente necessário no Windows porque `cmd.exe` não expande os curingas, mas ainda assim é uma boa prática.

```
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**
>
> A ordem na qual você especifica as pilhas não é necessariamente a ordem em que elas serão processadas. O AWS CDK kit de ferramentas considera as dependências entre as pilhas ao decidir a ordem na qual processá-las. Por exemplo, digamos que uma pilha usa um valor produzido por outra (como o ARN de um recurso definido na segunda pilha). Nesse caso, a segunda pilha é sintetizada antes da primeira devido a essa dependência. Você pode adicionar dependências entre as pilhas manualmente usando o método da pilha. `addDependency()`

Inicializando seu ambiente AWS

A implantação de pilhas com o CDK exige que AWS CDK recursos dedicados especiais sejam provisionados. O `cdk bootstrap` comando cria os recursos necessários para você. Você só

precisa inicializar se estiver implantando uma pilha que exija esses recursos dedicados. Para mais detalhes, consulte the section called “Bootstrapping”.

```
cdk bootstrap
```

Se emitido sem argumentos, conforme mostrado aqui, o `cdk bootstrap` comando sintetiza o aplicativo atual e inicializa os ambientes nos quais suas pilhas serão implantadas. Se o aplicativo contiver pilhas independentes do ambiente, que não especificam explicitamente um ambiente, a conta e a região padrão serão inicializadas ou o ambiente especificado usando. `--profile`

Fora de um aplicativo, você deve especificar explicitamente o ambiente a ser inicializado. Você também pode fazer isso para inicializar um ambiente que não esteja especificado em seu aplicativo ou AWS perfil local. As credenciais devem ser configuradas (por exemplo, in`~/.aws/credentials`) para a conta e a região especificadas. Você pode especificar um perfil que contenha as credenciais necessárias.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.
cdk bootstrap 1111111111/us-east-1
cdk bootstrap --profile test 1111111111/us-east-1
```

Important

Cada ambiente (combinação de conta/região) no qual você implanta essa pilha deve ser inicializado separadamente.

Você pode incorrer em AWS cobranças pelo que eles AWS CDK armazenam nos recursos inicializados. Além disso, se você usar`-bootstrap-customer-key`, uma chave do AWS KMS será criada, o que também incorrerá em cobranças por ambiente.

Note

As versões anteriores do modelo de bootstrap criavam uma chave KMS por padrão. Para evitar cobranças, reinicialize usando. `--no-bootstrap-customer-key`

> **Note**
>
> O CDK Toolkit v2 não suporta o modelo de bootstrap original, apelidado de modelo legado, usado por padrão com o CDK v1.

> **Important**
>
> O modelo de bootstrap moderno concede efetivamente as permissões implícitas no `--cloudformation-execution-policies` para qualquer AWS conta na `--trust` lista. Por padrão, isso estende as permissões de leitura e gravação em qualquer recurso na conta inicializada. Certifique-se de configurar a pilha de inicialização com políticas e contas confiáveis com as quais você se sinta confortável.

Criação de um novo aplicativo

Para criar um novo aplicativo, crie um diretório para ele e, dentro do diretório, emita`cdk init`.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

Os idiomas suportados (*IDIOMA*) são:

Código	Idioma
`typescript`	TypeScript
`javascript`	JavaScript
`python`	Python
`java`	Java
`csharp`	C#

TEMPLATE é um modelo opcional. Se o modelo desejado for app, o padrão, você poderá omiti-lo. Os modelos disponíveis são:

Modelo	Descrição
`app`(padrão)	Cria um AWS CDK aplicativo vazio.
`sample-app`	Cria um AWS CDK aplicativo com uma pilha contendo uma fila do Amazon SQS e um tópico do Amazon SNS.

Os modelos usam o nome da pasta do projeto para gerar nomes para arquivos e classes dentro do seu novo aplicativo.

Listando pilhas

Para ver uma lista dos IDs das pilhas em seu AWS CDK aplicativo, insira um dos seguintes comandos equivalentes:

```
cdk list
cdk ls
```

Se seu aplicativo contiver pilhas do CDK Pipelines, o CDK Toolkit exibirá os nomes das pilhas como caminhos de acordo com sua localização na hierarquia do pipeline. (Por exemplo`PipelineStack`,`PipelineStack/Prod`, `PipelineStack/Prod/MyService` e.)

Se seu aplicativo contiver muitas pilhas, você poderá especificar IDs de pilha completas ou parciais das pilhas a serem listadas. Para ter mais informações, consulte the section called “Especificando pilhas”.

Adicione a `--long` bandeira para ver mais informações sobre as pilhas, incluindo os nomes das pilhas e seus ambientes (AWS conta e região).

Sintetizando pilhas

O `cdk synthesize` comando (quase sempre abreviado`synth`) sintetiza uma pilha definida no seu aplicativo em um modelo. CloudFormation

```
cdk synth         # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"     # all stacks in app
```

> **Note**
>
> Na verdade, o CDK Toolkit executa seu aplicativo e sintetiza novos modelos antes da maioria das operações (como ao implantar ou comparar pilhas). Esses modelos são armazenados por padrão no `cdk.out` diretório. O `cdk synth` comando simplesmente imprime os modelos gerados para uma ou mais pilhas especificadas.

Veja `cdk synth --help` todas as opções disponíveis. Algumas das opções usadas com mais frequência são abordadas na seção a seguir.

Especificando valores de contexto

Use a `-c` opção `--context` ou para passar valores de contexto de tempo de execução para seu aplicativo CDK.

```
# 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
```

Ao implantar várias pilhas, os valores de contexto especificados normalmente são passados para todas elas. Se quiser, você pode especificar valores diferentes para cada pilha prefixando o nome da pilha no valor do contexto.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

Especificando o formato de exibição

Por padrão, o modelo sintetizado é exibido no formato YAML. Em vez disso, adicione o `--json` sinalizador para exibi-lo no formato JSON.

```
cdk synth --json MyStack
```

Especificando o diretório de saída

Adicione a opção `--output` (`-o`) para gravar os modelos sintetizados em um diretório diferente de. `cdk.out`

```
cdk synth --output=~/templates
```

Implantação de pilhas

O `cdk deploy` subcomando implanta uma ou mais pilhas especificadas em sua conta. AWS

```
cdk deploy        # if app contains only one stack
cdk deploy MyStack
cdk deploy Stack1 Stack2
cdk deploy "*"    # all stacks in app
```

> **Note**
>
> O CDK Toolkit executa seu aplicativo e sintetiza novos AWS CloudFormation modelos antes de implantar qualquer coisa. Portanto, a maioria das opções de linha de comando que você pode usar com `cdk synth` (por exemplo,`--context`) também pode ser usada com`cdk deploy`.

Veja `cdk deploy --help` todas as opções disponíveis. Algumas das opções mais úteis são abordadas na seção a seguir.

Ignorando a síntese

O cdk deploy comando normalmente sintetiza as pilhas do seu aplicativo antes da implantação para garantir que a implantação reflita a versão mais recente do seu aplicativo. Se você sabe que não alterou seu código desde a última vezcdk synth, você pode suprimir a etapa de síntese redundante durante a implantação. Para fazer isso, especifique o `cdk.out` diretório do seu projeto na --app opção.

```
cdk deploy --app cdk.out StackOne StackTwo
```

Desativando a reversão

AWS CloudFormation tem a capacidade de reverter as alterações para que as implantações sejam atômicas. Isso significa que eles são bem-sucedidos ou fracassam como um todo. O AWS CDK herda esse recurso porque sintetiza e implanta modelos. AWS CloudFormation

A reversão garante que seus recursos estejam sempre em um estado consistente, o que é vital para as pilhas de produção. No entanto, enquanto você ainda está desenvolvendo sua infraestrutura, algumas falhas são inevitáveis, e a reversão de implantações malsucedidas pode atrasá-lo.

Por esse motivo, o CDK Toolkit permite que você desative a reversão adicionando `--no-rollback` ao seu comando. `cdk deploy` Com esse sinalizador, implantações com falha não são revertidas. Em vez disso, os recursos implantados antes do recurso com falha permanecem em vigor e a próxima implantação começa com o recurso com falha. Você passará muito menos tempo esperando por implantações e muito mais tempo desenvolvendo sua infraestrutura.

Troca a quente

Use o `--hotswap` sinalizador with `cdk deploy` para tentar atualizar seus AWS recursos diretamente em vez de gerar um conjunto de AWS CloudFormation alterações e implantá-lo. A implantação volta à AWS CloudFormation implantação se a troca dinâmica não for possível.

Atualmente, o hot swapping suporta funções Lambda, máquinas de estado Step Functions e imagens de contêineres do Amazon ECS. O `--hotswap` sinalizador também desativa a reversão (ou seja, implica). `--no-rollback`

> **Important**
>
> A troca dinâmica não é recomendada para implantações de produção.

Modo de relógio

O modo de observação do CDK Toolkit (cdk deploy --watchou abreviadamente) monitora continuamente os arquivos de origem e os ativos do seu aplicativo CDK em cdk watch busca de alterações. Ele executa imediatamente uma implantação das pilhas especificadas quando uma alteração é detectada.

Por padrão, essas implantações usam a `--hotswap` bandeira, que acelera a implantação de alterações nas funções do Lambda. Ele também volta à implantação AWS CloudFormation se

você tiver alterado a configuração da infraestrutura. Para `cdk watch` sempre realizar AWS CloudFormation implantações completas, adicione o `--no-hotswap` sinalizador a. `cdk watch`

Todas as alterações feitas enquanto `cdk watch` você já está executando uma implantação são combinadas em uma única implantação, que começa assim que a implantação em andamento é concluída.

O modo de observação usa a `"watch"` tecla do projeto `cdk.json` para determinar quais arquivos monitorar. Por padrão, esses arquivos são os arquivos e ativos do seu aplicativo, mas isso pode ser alterado modificando as `"exclude"` entradas `"include"` e na `"watch"` chave. O `cdk.json` arquivo a seguir mostra um exemplo dessas entradas.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

`cdk watch`executa o `"build"` comando de `cdk.json` para criar seu aplicativo antes da síntese. Se sua implantação exigir algum comando para criar ou empacotar seu código Lambda (ou qualquer outra coisa que não esteja em seu aplicativo CDK), adicione-o aqui.

Os curingas no estilo Git, tanto quanto * e**, podem ser usados nas teclas e. `"watch"` `"build"` Cada caminho é interpretado em relação ao diretório pai do`cdk.json`. O valor padrão de `include` é`**/*`, ou seja, todos os arquivos e diretórios no diretório raiz do projeto. `exclude`é opcional.

> **Important**
>
> O modo de observação não é recomendado para implantações de produção.

Especificando parâmetros AWS CloudFormation

O AWS CDK kit de ferramentas suporta a especificação de AWS CloudFormation parâmetros na implantação. Você pode fornecê-los na linha de comando seguindo o `--parameters` sinalizador.

```
cdk deploy MyStack --parameters uploadBucketName=UploadBucket
```

Para definir vários parâmetros, use vários `--parameters` sinalizadores.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
 downloadBucketName=DownBucket
```

Se você estiver implantando várias pilhas, poderá especificar um valor diferente de cada parâmetro para cada pilha. Para fazer isso, prefixe o nome do parâmetro com o nome da pilha e dois pontos. Caso contrário, o mesmo valor será passado para todas as pilhas.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --
parameters YourStack:uploadBucketName=UpBucket
```

Por padrão, o AWS CDK retém os valores dos parâmetros de implantações anteriores e os usa em implantações posteriores, caso não sejam especificados explicitamente. Use o `--no-previous-parameters` sinalizador para exigir que todos os parâmetros sejam especificados.

Especificando o arquivo de saídas

Se sua pilha declarar AWS CloudFormation saídas, elas normalmente são exibidas na tela no final da implantação. Para gravá-los em um arquivo no formato JSON, use o `--outputs-file` sinalizador.

```
cdk deploy --outputs-file outputs.json MyStack
```

Mudanças relacionadas à segurança

Para protegê-lo contra alterações não intencionais que afetam sua postura de segurança, o AWS CDK kit de ferramentas solicita que você aprove as alterações relacionadas à segurança antes de implantá-las. Você pode especificar o nível de alteração que requer aprovação:

```
cdk deploy --require-approval LEVEL
```

O *NÍVEL* pode ser um dos seguintes:

Prazo	Significado
`never`	A aprovação nunca é necessária

Prazo	Significado
`any-change`	Requer aprovação em qualquer IAM ou security-group-related alteração
`broadening` (padrão)	Requer aprovação quando declarações do IAM ou regras de trânsito são adicionadas; remoções não exigem aprovação

A configuração também pode ser configurada no `cdk.json` arquivo.

```
{
  "app": "...",
  "requireApproval": "never"
}
```

Comparando pilhas

O `cdk diff` comando compara a versão atual de uma pilha (e suas dependências) definida em seu aplicativo com as versões já implantadas ou com um AWS CloudFormation modelo salvo e exibe uma lista de alterações.

```
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
```

Para comparar as pilhas do seu aplicativo com a implantação existente:

```
cdk diff MyStack
```

Para comparar as pilhas do seu aplicativo com um CloudFormation modelo salvo:

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

Importar recursos existentes para uma pilha

Você pode usar o `cdk import` comando para colocar os recursos sob o gerenciamento CloudFormation de uma AWS CDK pilha específica. Isso é útil se você estiver migrando ou movendo recursos entre pilhas ou alterando sua identificação lógica. `cdk import` Usa importações de CloudFormation recursos. AWS CDK Veja a lista de recursos que podem ser importados aqui.

Para importar um recurso existente em uma AWS CDK pilha, siga as etapas a seguir:

- Certifique-se de que o recurso não esteja sendo gerenciado atualmente por nenhuma outra CloudFormation pilha. Se estiver, primeiro defina a política de remoção para a pilha `RemovalPolicy.RETAIN` em que o recurso está atualmente e execute uma implantação. Em seguida, remova o recurso da pilha e execute outra implantação. Esse processo garantirá que o recurso não seja mais gerenciado CloudFormation , mas não o exclua.
- Execute a `cdk diff` para garantir que não haja alterações pendentes na AWS CDK pilha para a qual você deseja importar recursos. As únicas alterações permitidas em uma operação de "importação" são a adição de novos recursos que você deseja importar.
- Adicione construções para os recursos que você deseja importar para sua pilha. Por exemplo, se você quiser importar um bucket do Amazon S3, adicione algo como. `new s3.Bucket(this, 'ImportedS3Bucket', {});` Não faça nenhuma modificação em nenhum outro recurso.

Você também deve se certificar de modelar exatamente o estado que o recurso tem atualmente na definição. Para o exemplo do bucket, não se esqueça de incluir AWS KMS chaves, políticas de ciclo de vida e qualquer outra coisa que seja relevante sobre o bucket. Caso contrário, as operações de atualização subsequentes podem não fazer o que você espera.

Você pode escolher se deseja ou não incluir o nome do bucket físico. Geralmente, recomendamos não incluir nomes de recursos em suas definições de AWS CDK recursos para que seja mais fácil implantar seus recursos várias vezes.

- Executar `cdk import` *`STACKNAME`*.
- Se os nomes dos recursos não estiverem em seu modelo, a CLI solicitará que você informe os nomes reais dos recursos que você está importando. Depois disso, a importação é iniciada.
- Quando `cdk import` relata o sucesso, o recurso agora é gerenciado por AWS CDK CloudFormation e. Qualquer alteração subsequente que você fizer nas propriedades do recurso em seu AWS CDK aplicativo, a configuração de construção será aplicada na próxima implantação.
- Para confirmar se a definição do recurso no seu AWS CDK aplicativo corresponde ao estado atual do recurso, você pode iniciar uma operação de detecção de CloudFormation desvios.

Atualmente, esse recurso não oferece suporte à importação de recursos para pilhas aninhadas.

Configuração (**`cdk.json`**)

Os valores padrão de muitos sinalizadores de linha de comando do CDK Toolkit podem ser armazenados no arquivo de um projeto ou no `cdk.json` `.cdk.json` arquivo do seu diretório de usuário. A seguir está uma referência alfabética às configurações suportadas.

Chave	Observações	Opção CDK Toolkit
`app`	O comando que executa o aplicativo CDK.	--app
`assetMetadata`	Se`false`, o CDK não adiciona metadados aos recursos que usam ativos.	--no-asset-metadata
`bootstrapKmsKeyId`	Substitui o ID da AWS KMS chave usada para criptografar	--bootstrap-kms-key-id

Chave	Observações	Opção CDK Toolkit
	o bucket de implantação do Amazon S3.	
`build`	O comando que compila ou constrói o aplicativo CDK antes da síntese. Não é permitido entrar`~/.cdk.json` on .	--build
`browser`	O comando para iniciar um navegador da Web para o `cdk docs` subcomando.	--browser
`context`	Consulte the section called "Contexto". Os valores de contexto em um arquivo de configuração não serão apagados pelo`cdk context --clear`. (O CDK Toolkit coloca valores de contexto em cache.) `cdk.context.json`	--context
`debug`	Se`true`, o CDK Toolkit emite informações mais detalhadas úteis para depuração.	--debug
`language`	A linguagem a ser usada para inicializar novos projetos.	--language
`lookups`	Se`false`, nenhuma pesquisa de contexto for permitida. A síntese falhará se alguma pesquisa de contexto precisar ser realizada.	--no-lookups

Chave	Observações	Opção CDK Toolkit
`notices`	Se `false`, suprime a exibição de mensagens sobre vulnerabilidades de segurança, regressões e versões não suportadas.	--no-notices
`output`	O nome do diretório no qual o conjunto de nuvem sintetizado será emitido (padrão). `"cdk.out"`	--output
`outputsFile`	O arquivo no qual as AWS CloudFormation saídas das pilhas implantadas serão gravadas (no formato JSON).	--outputs-file
`pathMetadata`	Se `false`, os metadados do caminho do CDK não forem adicionados aos modelos sintetizados.	--no-path-metadata
`plugin`	Matriz JSON especificando os nomes dos pacotes ou os caminhos locais dos pacotes que estendem o CDK	--plugin
`profile`	Nome do AWS perfil padrão usado para especificar as credenciais da região e da conta.	--profile
`progress`	Se definido como `"events"`, o CDK Toolkit exibe todos os AWS CloudFormation eventos durante a implantação, em vez de uma barra de progresso.	--progress

Chave	Observações	Opção CDK Toolkit
`requireApproval`	Nível de aprovação padrão para alterações de segurança . Consulte the section called “Mudanças relacionadas à segurança”	--require-approval
`rollback`	`false`Em caso afirmativo, as implantações com falha não serão revertidas.	--no-rollback
`staging`	S`false`, os ativos não forem copiados para o diretório de saída (use para depuração local dos arquivos de origem com). AWS SAM	--no-staging
`tags`	Objeto JSON contendo tags (pares de valores-chave) para a pilha.	--tags
`toolkitBucketName`	O nome do bucket do Amazon S3 usado para implantar ativos como funções Lambda e imagens de contêineres (consulte. the section called “Inicializando seu ambiente AWS”	--toolkit-bucket-name
`toolkitStackName`	O nome da pilha de bootstrap (consulte. the section called “Inicializando seu ambiente AWS”	--toolkit-stack-name
`versionReporting`	S`false`, opta por não receber relatórios de versão.	--no-version-reporting

Chave	Observações	Opção CDK Toolkit
`watch`	Objeto JSON contendo `"include" "exclude"` chaves que indicam quais arquivos devem (ou não) acionar uma reconstrução do projeto quando alterados . Consulte the section called “Modo de relógio”.	--watch

referência de comando da cdk migrate

Referência para o AWS Cloud Development Kit (AWS CDK) comando Command Line Interface (CLI)`cdk migrate`. Para obter mais informações sobre o uso`cdk migrate`, consulteMigre recursos e AWS CloudFormation modelos existentes para o AWS CDK.

O `cdk migrate` comando migra AWS recursos, AWS CloudFormation pilhas e modelos locais AWS CloudFormation implantados para o. AWS CDK

Tópicos

- Uso
- Opções

Uso

```
$ cdk migrate <options>
```

Opções

Opções obrigatórias

`--stack-name` *STRING*

O nome da AWS CloudFormation pilha que será criada no aplicativo CDK após a migração.

Obrigatório: Sim

Opções condicionais

`--from-path` *PATH*

O caminho para o AWS CloudFormation modelo a ser migrado. Forneça essa opção para especificar um modelo local.

Obrigatório: condicional. Obrigatório ao migrar de um AWS CloudFormation modelo local.

`--from-scan` *STRING*

Ao migrar recursos implantados de um AWS ambiente, use essa opção para especificar se um novo escaneamento deve ser iniciado ou se AWS CDK CLI deve usar o último escaneamento bem-sucedido.

Obrigatório: condicional. Necessário ao migrar dos recursos implantados. AWS

Valores aceitos:`most-recent`, `new`

`--from-stack`

Forneça essa opção para migrar de uma pilha implantada. AWS CloudFormation Use `--stack-name` para especificar o nome da AWS CloudFormation pilha implantada.

Obrigatório: condicional. Necessário ao migrar de uma pilha implantada. AWS CloudFormation

Opções opcionais

`--account` *STRING*

A conta da qual recuperar o modelo de AWS CloudFormation pilha.

Obrigatório: não

Padrão: AWS CDK CLI Obtém as informações da conta de fontes padrão.

`--compress`

Forneça essa opção para compactar o projeto CDK gerado em um ZIP arquivo.

Obrigatório: não

`--filter` *ARRAY*

Use ao migrar recursos implantados de uma AWS conta e. Região da AWS Essa opção especifica um filtro para determinar quais recursos implantados devem ser migrados.

Essa opção aceita uma matriz de pares de valores-chave, em que chave representa o tipo de filtro e valor representa o valor a ser filtrado.

As seguintes chaves são aceitas:

- `resource-identifier`— Um identificador para o recurso. O valor pode ser o ID lógico ou físico do recurso. Por exemplo, `resource-identifier="ClusterName"`.
- `resource-type-prefix`— O prefixo do tipo de AWS CloudFormation recurso. Por exemplo, especifique `resource-type-prefix="AWS::DynamoDB::"` para filtrar todos os recursos do Amazon DynamoDB.
- `tag-key`— A chave de uma tag de recurso. Por exemplo, `tag-key="myTagKey"`.
- `tag-value`— O valor de uma tag de recurso. Por exemplo, `tag-value="myTagValue"`.

Forneça vários pares de valores-chave para lógica `AND` condicional. O exemplo a seguir filtra qualquer recurso do DynamoDB marcado `myTagKey` com a chave da tag:. `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

Forneça a `--filter` opção várias vezes em um único comando para lógica `OR` condicional. O exemplo a seguir filtra qualquer recurso que seja um recurso do DynamoDB ou esteja marcado `myTagKey` com a chave da tag:. `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

Obrigatório: não

`--language` *`STRING`*

A linguagem de programação a ser usada no projeto CDK criado durante a migração.

Obrigatório: não

Valores aceitos:`typescript`,`python`,`java`,`csharp`,`go`.

Padrão: `typescript`

`--output-path` *`PATH`*

O caminho de saída para o projeto CDK migrado.

Obrigatório: não

Padrão: Por padrão, o AWS CDK CLI usará seu diretório de trabalho atual.

`--region` *`STRING`*

O Região da AWS para recuperar o modelo de AWS CloudFormation pilha.

Obrigatório: não

Padrão: AWS CDK CLI Obtém Região da AWS informações de fontes padrão.

AWS Kit de ferramentas para Visual Studio Code

O AWS Toolkit for Visual Studio Code é um plug-in de código aberto para o Visual Studio Code que facilita a criação, depuração e implantação de aplicativos. AWS O kit de ferramentas fornece uma experiência integrada para o desenvolvimento de AWS CDK aplicativos. Ele inclui o recurso AWS CDK Explorer para listar seus AWS CDK projetos e navegar pelos vários componentes do aplicativo CDK. Instale o AWS Toolkit e saiba mais sobre como usar o AWS CDK Explorer.

AWS SAM integração

O AWS CDK e o AWS Serverless Application Model (AWS SAM) podem trabalhar juntos para permitir que você crie e teste localmente aplicativos sem servidor definidos no CDK. Para obter informações completas, consulte AWS Cloud Development Kit (AWS CDK)o Guia do AWS SAM desenvolvedor. Para instalar a CLI do SAM, consulte Instalando a AWS SAM CLI.

Testando construções

Com o AWS CDK, sua infraestrutura pode ser tão testável quanto qualquer outro código que você escrever. A abordagem padrão para testar AWS CDK aplicativos usa o módulo AWS CDK de asserções do e estruturas de teste populares, como Jest for TypeScript and JavaScript ou Pytest for Python.

Há duas categorias de testes que você pode escrever para AWS CDK aplicativos.

- Asserções refinadas testam aspectos específicos do AWS CloudFormation modelo gerado, como “esse recurso tem essa propriedade com esse valor”. Esses testes podem detectar regressões. Eles também são úteis quando você desenvolve novos recursos usando o desenvolvimento orientado a testes. (Você pode escrever um teste primeiro e depois fazê-lo passar escrevendo uma implementação correta.) Afirmações refinadas são os testes usados com mais frequência.
- Os testes instantâneos testam o modelo sintetizado em relação a um AWS CloudFormation modelo de linha de base armazenado anteriormente. Os testes instantâneos permitem que você refatore livremente, pois você pode ter certeza de que o código refatorado funciona exatamente da mesma forma que o original. Se as alterações foram intencionais, é possível aceitar uma nova linha de base para futuros testes. No entanto, as atualizações do CDK também podem fazer com que os modelos sintetizados sejam alterados, portanto, você não pode confiar apenas em instantâneos para garantir que sua implementação esteja correta.

> **Note**
>
> Versões completas dos TypeScript aplicativos Python e Java usados como exemplos neste tópico estão disponíveis em. GitHub

Conceitos básicos

Para ilustrar como escrever esses testes, criaremos uma pilha que contém uma máquina de AWS Step Functions estados e uma AWS Lambda função. A função Lambda está inscrita em um tópico do Amazon SNS e simplesmente encaminha a mensagem para a máquina de estado.

Primeiro, crie um projeto de aplicativo CDK vazio usando o CDK Toolkit e instalando as bibliotecas de que precisaremos. As construções que usaremos estão todas no pacote CDK principal, que é

uma dependência padrão em projetos criados com o CDK Toolkit. No entanto, você deve instalar sua estrutura de teste.

TypeScript

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

Crie um diretório para seus testes.

```
mkdir test
```

Edite os projetos `package.json` para dizer ao NPM como executar o Jest e para dizer ao Jest quais tipos de arquivos coletar. As mudanças necessárias são as seguintes.

- Adicione uma nova `test` chave à `scripts` seção
- Adicione Jest e seus tipos à seção `devDependencies`
- Adicione uma nova chave `jest` de nível superior com uma declaração `moduleFileExtensions`

Essas mudanças são mostradas no esboço a seguir. Coloque o novo texto onde indicado em`package.json`. Os espaços reservados “...” indicam partes existentes do arquivo que não devem ser alteradas.

```
{
  ...
  "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
```

Crie um diretório para seus testes.

```
mkdir test
```

Edite os projetos `package.json` para dizer ao NPM como executar o Jest e para dizer ao Jest quais tipos de arquivos coletar. As mudanças necessárias são as seguintes.

- Adicione uma nova `test` chave à `scripts` seção
- Adicione Jest à seção `devDependencies`
- Adicione uma nova chave `jest` de nível superior com uma declaração `moduleFileExtensions`

Essas mudanças são mostradas no esboço a seguir. Coloque o novo texto onde indicado em`package.json`. Os espaços reservados “...” indicam partes existentes do arquivo que não devem ser alteradas.

```
{
  ...
  "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
```

Abra o projeto no seu IDE Java preferido. (No Eclipse, use Arquivo > Importar > Projetos existentes do Maven.)

C#

```
mkdir state-machine && cd-state-machine
cdk init --language=csharp
```

Abra a `src\StateMachine.sln` no Visual Studio.

Clique com o botão direito do mouse na solução no Solution Explorer e escolha Adicionar > Novo projeto. Pesquise por MSTest C# e adicione um projeto de teste MSTest para C#. (O nome padrão `TestProject1` é bom.)

Clique com o botão direito do mouse **`TestProject1`** e escolha Adicionar > Referência do projeto e adicione o `StateMachine` projeto como referência.

A pilha de exemplos

Aqui está a pilha que será testada neste tópico. Como descrevemos anteriormente, ele contém uma função Lambda e uma máquina de estado Step Functions e aceita um ou mais tópicos do Amazon SNS. A função Lambda está inscrita nos tópicos do Amazon SNS e os encaminha para a máquina de estado.

Você não precisa fazer nada de especial para tornar o aplicativo testável. Na verdade, essa pilha de CDK não é diferente em nada importante das outras pilhas de exemplo neste Guia.

TypeScript

Insira o seguinte código em`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

Insira o seguinte código em`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

Insira o seguinte código em`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);
            }

        }
    }
}
```

Modificaremos o ponto de entrada principal do aplicativo para que não instanciemos realmente nossa pilha. Não queremos implantá-lo acidentalmente. Nossos testes criarão um aplicativo e uma instância da pilha para testes. Essa é uma tática útil quando combinada com o desenvolvimento orientado a testes: certifique-se de que a pilha passe em todos os testes antes de ativar a implantação.

TypeScript

Em `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

Em `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

Em `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();
        }
    }
}
```

A função Lambda

Nossa pilha de exemplos inclui uma função Lambda que inicia nossa máquina de estado. Precisamos fornecer o código-fonte dessa função para que o CDK possa agrupá-la e implantá-la como parte da criação do recurso da função Lambda.

- Crie a pasta `start-state-machine` no diretório principal do aplicativo.
- Nessa pasta, crie pelo menos um arquivo. Por exemplo, você pode salvar o código a seguir em`start-state-machines/index.js`.

  ```
  exports.handler = async function (event, context) {
   return 'hello world';
  };
  ```

  No entanto, qualquer arquivo funcionará, já que não vamos realmente implantar a pilha.

Execução de testes

Para referência, aqui estão os comandos que você usa para executar testes no seu AWS CDK aplicativo. Esses são os mesmos comandos que você usaria para executar os testes em qualquer projeto usando a mesma estrutura de teste. Para linguagens que exigem uma etapa de construção, inclua-a para garantir que seus testes tenham sido compilados.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

Crie sua solução (F6) para descobrir os testes e, em seguida, execute-os (Teste > Executar todos os testes). Para escolher quais testes executar, abra o Test Explorer (Test > Test Explorer).

Ou:

```
dotnet test src
```

Afirmações refinadas

A primeira etapa para testar uma pilha com afirmações refinadas é sintetizar a pilha, porque estamos escrevendo afirmações com base no modelo gerado. AWS CloudFormation

Nossa `StateMachineStackStack` exigência é que passemos o tópico do Amazon SNS para ser encaminhado para a máquina estadual. Então, em nosso teste, criaremos uma pilha separada para conter o tópico.

Normalmente, ao escrever um aplicativo CDK, você pode criar uma subclasse `Stack` e instanciar o tópico do Amazon SNS no construtor da pilha. Em nosso teste, instanciamos `Stack` diretamente e, em seguida, passamos essa pilha como escopo, anexando-a à `Topic` pilha. Isso é funcionalmente equivalente e menos detalhado. Também ajuda a fazer com que as pilhas usadas somente em testes “pareçam diferentes” das pilhas que você pretende implantar.

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
        }
```

```
    }
}
```

Agora podemos afirmar que a função Lambda e a assinatura do Amazon SNS foram criadas.

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);
```

Nosso teste de função Lambda afirma que duas propriedades específicas do recurso da função têm valores específicos. Por padrão, o `hasResourceProperties` método executa uma correspondência parcial nas propriedades do recurso, conforme indicado no modelo sintetizado CloudFormation . Esse teste exige que as propriedades fornecidas existam e tenham os valores especificados, mas o recurso também pode ter outras propriedades que não foram testadas.

Nossa afirmação do Amazon SNS afirma que o modelo sintetizado contém uma assinatura, mas nada sobre a assinatura em si. Incluímos essa afirmação principalmente para ilustrar como afirmar a contagem de recursos. A `Template` classe oferece métodos mais específicos para escrever afirmações nas `Mapping` seções `ResourcesOutputs`, e do CloudFormation modelo.

Combinadores

O comportamento padrão de correspondência parcial de `hasResourceProperties` pode ser alterado usando matchers da `Match`classe.

Os matchers variam de tolerantes (`Match.anyValue`) a estritos (). `Match.objectEquals` Eles podem ser agrupados para aplicar diferentes métodos de correspondência a diferentes partes das propriedades do recurso. Usando `Match.objectEquals` e `Match.anyValue` juntos, por exemplo, podemos testar a função do IAM da máquina de estado de forma mais completa, sem exigir valores específicos para propriedades que possam mudar.

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" }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }));
```

Muitos CloudFormation recursos incluem objetos JSON serializados representados como strings. O `Match.serializedJson()` matcher pode ser usado para combinar propriedades dentro desse JSON.

Por exemplo, as máquinas de estado do Step Functions são definidas usando uma string na Amazon States Language baseada em JSON. Usaremos `Match.serializedJson()` para garantir que nosso estado inicial seja a única etapa. Novamente, usaremos combinadores aninhados para aplicar diferentes tipos de correspondência a diferentes partes do objeto.

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() }
                            }}
                        }}
                    })
                )}});
```

Capturando

Geralmente, é útil testar propriedades para garantir que elas sigam formatos específicos ou tenham o mesmo valor de outra propriedade, sem precisar saber seus valores exatos com antecedência. O `assertions` módulo fornece esse recurso em sua `Capture`classe.

Ao especificar uma `Capture` instância no lugar de um valor em`hasResourceProperties`, esse valor é retido no `Capture` objeto. O valor real capturado pode ser recuperado usando os `as` métodos do objeto, incluindo`asNumber()`, e `asString()``asObject`, e submetido a testes. Use `Capture` com um matcher para especificar a localização exata do valor a ser capturado nas propriedades do recurso, incluindo propriedades JSON serializadas.

O exemplo a seguir testa para garantir que o estado inicial de nossa máquina de estado tenha um nome que comece com`Start`. Também testa se esse estado está presente na lista de estados da máquina.

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()));
```

Testes instantâneos

No teste de instantâneo, você compara todo o modelo sintetizado com um CloudFormation modelo de linha de base armazenado anteriormente (geralmente chamado de “mestre”). Ao contrário das afirmações refinadas, o teste instantâneo não é útil para capturar regressões. Isso ocorre porque o teste de instantâneo se aplica a todo o modelo, e coisas além das alterações no código podem

causar pequenas (ou not-so-small) diferenças nos resultados da síntese. Essas alterações podem nem mesmo afetar sua implantação, mas ainda assim farão com que um teste de snapshot falhe.

Por exemplo, você pode atualizar uma construção de CDK para incorporar uma nova prática recomendada, que pode causar alterações nos recursos sintetizados ou na forma como eles são organizados. Como alternativa, você pode atualizar o CDK Toolkit para uma versão que relata metadados adicionais. Alterações nos valores de contexto também podem afetar o modelo sintetizado.

No entanto, os testes instantâneos podem ser de grande ajuda na refatoração, desde que você mantenha constantes todos os outros fatores que possam afetar o modelo sintetizado. Você saberá imediatamente se uma alteração que você fez alterou involuntariamente o modelo. Se a alteração for intencional, basta aceitar o novo modelo como linha de base.

Por exemplo, se tivermos essa `DeadLetterQueue` construção:

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()
            });
        }
    }
}
```

Podemos testá-lo assim:

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());
        }
    }
}
```

Dicas para testes

Lembre-se de que seus testes durarão tanto quanto o código testado e serão lidos e modificados com a mesma frequência. Portanto, vale a pena considerar a melhor forma de escrevê-las.

Não copie e cole linhas de configuração ou afirmações comuns. Em vez disso, refatore essa lógica em fixtures ou funções auxiliares. Use bons nomes que reflitam o que cada teste realmente testa.

Não tente fazer muita coisa em um teste. De preferência, um teste deve testar um e somente um comportamento. Se você quebrar acidentalmente esse comportamento, exatamente um teste falhará e o nome do teste deverá indicar o que falhou. No entanto, esse é mais um ideal a ser buscado; às vezes, você inevitavelmente (ou inadvertidamente) escreve testes que testam mais de um comportamento. Os testes instantâneos são, por motivos que já descrevemos, especialmente propensos a esse problema, portanto, use-os com moderação.

Segurança para o AWS Cloud Development Kit (AWS CDK)

A segurança da nuvem na Amazon Web Services (AWS) é a nossa maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança. A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como a Segurança da nuvem e a Segurança na nuvem.

Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa todos os serviços oferecidos na AWS nuvem e fornecer serviços que você possa usar com segurança. Nossa responsabilidade de segurança é a maior prioridade em AWS, e a eficácia de nossa segurança é regularmente testada e verificada por auditores terceirizados como parte dos Programas de AWS Conformidade.

Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você está usando e por outros fatores, incluindo a sensibilidade de seus dados, os requisitos da sua organização e as leis e regulamentos aplicáveis.

Ele AWS CDK segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Tópicos

Gerenciamento de identidade e acesso para o AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar

AWS os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz AWS.

Usuário do serviço — Se você Serviços da AWS costuma fazer seu trabalho, seu administrador fornece as credenciais e as permissões de que você precisa. À medida que você usa mais AWS recursos para fazer seu trabalho, talvez precise de permissões adicionais. Entender como o acesso é gerenciado pode ajudá-lo a solicitar as permissões corretas ao seu administrador.

Administrador de serviços — Se você é responsável pelos AWS recursos da sua empresa, provavelmente tem acesso total aos AWS recursos. É seu trabalho determinar quais recursos Serviços da AWS e quais recursos seus usuários do serviço devem acessar. Assim, você deve enviar solicitações ao administrador do IAM para alterar as permissões dos usuários de seu serviço. Revise as informações nesta página para entender os Introdução ao IAM.

Administrador do IAM:Se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar o acesso ao Serviços da AWS.

Autenticando com identidades

A autenticação é a forma como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como o Usuário raiz da conta da AWS, como usuário do IAM ou assumindo uma função do IAM.

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Usuários (IAM Identity Center), a autenticação de login único da sua empresa e suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como uma identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Ao acessar AWS usando a federação, você está assumindo indiretamente uma função.

Dependendo do tipo de usuário que você é, você pode entrar no AWS Management Console ou no portal de AWS acesso. Para obter mais informações sobre como fazer login AWS, consulte Como fazer login Conta da AWS no Guia do Início de Sessão da AWS usuário.

Para acessar AWS programaticamente, AWS fornece kits de desenvolvimento de software (SDKs) e uma interface de linha de comando (CLI) para assinar criptograficamente suas solicitações usando suas credenciais. AWS CDK Se você não usa AWS ferramentas, você mesmo deve assinar as solicitações. Para obter mais informações sobre o uso do método recomendado para você assinar as solicitações por conta própria, consulte Signature Version 4 signing process (Processo de assinatura do Signature Versão 4) no Referência geral da AWS.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, AWS recomenda que você use a autenticação multifator (MFA) para aumentar a segurança da sua conta. Para saber mais, consulte Autenticação multifator no Guia AWS IAM Identity Center do usuário. Usar a autenticação multifator (MFA) na AWS no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos Serviços da AWS os recursos da conta. Essa identidade é chamada de usuário Conta da AWS raiz e é acessada fazendo login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não usar o usuário raiz para tarefas diárias. Proteja as credenciais do usuário raiz e use-as para executar as tarefas que somente ele pode executar. Para obter a lista completa das tarefas que exigem login como usuário raiz, consulte Tarefas que exigem credenciais de usuário raiz no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que usuários humanos, incluindo usuários que precisam de acesso de administrador, usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório de usuários corporativo, de um provedor de identidade da web AWS Directory Service, do diretório do Identity Center ou de qualquer usuário que acesse usando credenciais fornecidas Serviços da AWS por meio de uma fonte de identidade. Quando as identidades federadas são acessadas Contas da AWS, elas assumem funções, e as funções fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos usar o AWS IAM Identity Center. Você pode criar usuários e grupos no IAM Identity Center ou pode se conectar e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todos os seus Contas

da AWS aplicativos. Para mais informações sobre o Centro de Identidade do IAM, consulte “O que é o Centro de Identidade do IAM?” no Guia do usuário do AWS IAM Identity Center .

Grupos e usuários do IAM

Um usuário do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, recomendamos depender de credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, recomendamos alternar as chaves de acesso. Para mais informações, consulte Alterne as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e atribuir a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis.. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Quando criar um usuário do IAM (em vez de um perfil) no Guia do usuário do IAM.

Perfis do IAM

Uma função do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas. Ele se assemelha a um usuário do IAM, entretanto, não está associado a uma pessoa específica. Você pode assumir temporariamente uma função do IAM no AWS Management Console trocando de funções. Você pode assumir uma função chamando uma operação de AWS API AWS CLI ou usando uma URL personalizada. Para mais informações sobre métodos para o uso de perfis, consulte Usar perfis do IAM no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas pelo mesmo. Para mais

informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidades de terceiros no Guia do usuário do IAM. Se você usar o IAM Identity Center, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o IAM Identity Center correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de permissões no Guia do Usuário do AWS IAM Identity Center .

- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode assumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, com alguns Serviços da AWS, você pode anexar uma política diretamente a um recurso (em vez de usar uma função como proxy). Para saber a diferença entre funções e políticas baseadas em recurso para acesso entre contas, consulte Como os perfis do IAM diferem das políticas baseadas em recurso no Guia do usuário do IAM.
- Acesso entre serviços — Alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.
  - Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para realizar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um no Guia do usuário do IAM.
  - Função vinculada ao serviço — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode assumir o perfil para executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para perfis vinculados ao serviço.
- Aplicativos em execução no Amazon EC2 — Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma instância do EC2 e fazendo AWS CLI solicitações de API. AWS É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir uma AWS função a uma instância do EC2 e disponibilizá-la para todos os seus aplicativos, você cria um perfil de instância anexado à instância. Um perfil de instância contém o perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para obter mais informações, consulte Usar uma função do IAM para

conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Para saber se deseja usar os perfis do IAM, consulte Quando criar um perfil do IAM (em vez de um usuário) no Guia do usuário do IAM.

Validação de conformidade para o AWS Cloud Development Kit (AWS CDK)

Ele AWS CDK segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

A segurança e a conformidade dos AWS serviços são avaliadas por auditores terceirizados como parte de vários programas de AWS conformidade. Isso inclui SOC, PCI, FedRAMP, HIPAA e outros. AWS fornece uma lista frequentemente atualizada de AWS serviços no escopo de programas de conformidade específicos em AWS Serviços no escopo por programa de conformidade.

Relatórios de auditoria de terceiros estão disponíveis para você baixar usando o AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact.

Para obter mais informações sobre programas de AWS conformidade, consulte Programas de AWS conformidade.

Sua responsabilidade de conformidade ao usar o AWS CDK para acessar um AWS serviço é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua organização e pelas leis e regulamentações aplicáveis. Se o uso de um AWS serviço estiver sujeito à conformidade com padrões como HIPAA, PCI ou FedRAMP, fornece recursos para ajudar a: AWS

- Guias de início rápido sobre segurança e conformidade — guias de implantação que discutem considerações arquitetônicas e fornecem etapas para a implantação de ambientes básicos focados na segurança e na conformidade em. AWS
- AWS Recursos de conformidade — Uma coleção de pastas de trabalho e guias que podem ser aplicados ao seu setor e localização.
- AWS Config: um serviço que avalia até que ponto suas configurações de recursos estão compatíveis com práticas internas, diretrizes do setor e regulamentações.

- AWS Security Hub— Uma visão abrangente do seu estado de segurança interno AWS que ajuda você a verificar sua conformidade com os padrões e as melhores práticas do setor de segurança.

Resiliência para o AWS Cloud Development Kit (AWS CDK)

A infraestrutura global da Amazon Web Services (AWS) é construída em torno de AWS regiões e zonas de disponibilidade.

AWS As regiões fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância.

Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que executam o failover automaticamente entre as zonas de disponibilidade sem interrupção. As Zonas de Disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenter tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Ele AWS CDK segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Segurança de infraestrutura para o AWS Cloud Development Kit (AWS CDK)

Ele AWS CDK segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Solução de AWS CDK problemas comuns

Este tópico descreve como solucionar os seguintes problemas com o. AWS CDK

Depois de atualizar o AWS CDK, o AWS CDK Toolkit (CLI) relata uma incompatibilidade com a Construct Library AWS

A versão do AWS CDK kit de ferramentas (que fornece o `cdk` comando) deve ser pelo menos igual à versão do módulo principal da AWS Construct Library,`aws-cdk-lib`. O kit de ferramentas foi projetado para ser compatível com versões anteriores. A versão 2.x mais recente do kit de ferramentas pode ser usada com qualquer versão 1.x ou 2.x da biblioteca. Por esse motivo, recomendamos que você instale esse componente globalmente e o mantenha atualizado.

```
npm update -g aws-cdk
```

Se você precisar trabalhar com várias versões do AWS CDK Toolkit, instale uma versão específica do kit de ferramentas localmente na pasta do seu projeto.

Se você estiver usando TypeScript ou JavaScript, o diretório do seu projeto já contém uma cópia local versionada do CDK Toolkit.

Se você estiver usando outro idioma, use `npm` para instalar o AWS CDK Toolkit, omitindo o `-g` sinalizador e especificando a versão desejada. Por exemplo: .

```
npm install aws-cdk@2.0
```

Para executar um AWS CDK kit de ferramentas instalado localmente, use o comando `npx aws-cdk` em vez de somente`cdk`. Por exemplo: .

```
npx aws-cdk deploy MyStack
```

`npx aws-cdk`executa a versão local do AWS CDK Toolkit, se houver. Ele volta para a versão global quando um projeto não tem uma instalação local. Talvez seja conveniente configurar um alias de shell para garantir que ele `cdk` seja sempre invocado dessa forma.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

(voltar à lista)

Ao implantar minha AWS CDK pilha, recebo um erro **`NoSuchBucket`**

Seu AWS ambiente não foi inicializado e, portanto, não tem um bucket Amazon S3 para armazenar recursos durante a implantação. Você pode criar o bucket de teste e outros recursos necessários com o seguinte comando:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

Para evitar a geração de AWS cobranças inesperadas, o AWS CDK não inicializa automaticamente nenhum ambiente. Você deve inicializar explicitamente cada ambiente no qual será implantado.

Por padrão, os recursos de bootstrap são criados na região ou regiões que são usadas por pilhas no aplicativo atual AWS CDK . Como alternativa, eles são criados na região especificada em seu AWS

perfil local (definida por`aws configure`), usando a conta desse perfil. Você pode especificar uma conta e uma região diferentes na linha de comando da seguinte maneira. (Você deve especificar a conta e a região se não estiver no diretório de um aplicativo.)

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

Para ter mais informações, consulte the section called “Bootstrapping”.

(voltar à lista)

Ao implantar minha AWS CDK pilha, recebo uma mensagem **`forbidden: null`**

Você está implantando uma pilha que requer recursos de bootstrap, mas está usando uma função ou conta do IAM que não tem permissão para gravar nela. (O intervalo de teste é usado ao implantar pilhas que contêm ativos ou que sintetizam um AWS CloudFormation modelo maior que 50K.) Use uma conta ou função que tenha permissão para realizar a `s3:*` ação no bucket mencionado na mensagem de erro.

(voltar à lista)

Ao sintetizar uma AWS CDK pilha, recebo a mensagem **`--app is required either in command-line, in cdk.json or in ~/.cdk.json`**

Essa mensagem geralmente significa que você não está no diretório principal do seu AWS CDK projeto quando você emite`cdk synth`. O arquivo `cdk.json` nesse diretório, criado pelo `cdk init` comando, contém a linha de comando necessária para executar (e, assim, sintetizar) seu AWS CDK aplicativo. Para um TypeScript aplicativo, por exemplo, o padrão `cdk.json` é mais ou menos assim:

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

Recomendamos emitir `cdk` comandos somente no diretório principal do seu projeto, para que o AWS CDK kit de ferramentas possa `cdk.json` encontrá-lo e executar seu aplicativo com sucesso.

Se isso não for prático por algum motivo, o AWS CDK Toolkit procura a linha de comando do aplicativo em dois outros locais:

- `cdk.json`Em seu diretório inicial

- No próprio `cdk synth` comando usando a `-a` opção

Por exemplo, você pode sintetizar uma pilha de um TypeScript aplicativo da seguinte maneira.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

(voltar à lista)

Ao sintetizar uma AWS CDK pilha, recebo um erro porque o AWS CloudFormation modelo contém muitos recursos

O AWS CDK gera e implanta AWS CloudFormation modelos. AWS CloudFormation tem um limite rígido no número de recursos que uma pilha pode conter. Com o AWS CDK, você pode atingir esse limite mais rapidamente do que imagina.

> **Note**
>
> O limite AWS CloudFormation de recursos é 500 no momento em que este artigo foi escrito. Veja as AWS CloudFormation cotas para o limite atual de recursos.

As AWS construções de alto nível baseadas em intenção da Construct Library provisionam automaticamente quaisquer recursos auxiliares necessários para registro, gerenciamento de chaves, autorização e outros fins. Por exemplo, conceder acesso a um recurso a outro gera todos os objetos do IAM necessários para que os serviços relevantes se comuniquem.

Em nossa experiência, o uso real de construções baseadas em intenção resulta em 1 a 5 AWS CloudFormation recursos por construção, embora isso possa variar. Para aplicativos sem servidor, é normal ter de 5 a 8 AWS recursos por endpoint de API.

Os padrões, que representam um nível mais alto de abstração, permitem definir ainda mais AWS recursos com ainda menos código. O AWS CDK código emthe section called “ECS”, por exemplo, gera mais de 50 AWS CloudFormation recursos enquanto define apenas três construções!

Exceder o limite AWS CloudFormation de recursos é um erro durante a AWS CloudFormation síntese. Ele AWS CDK emite um aviso se sua pilha exceder 80% do limite. Você pode usar um limite diferente definindo a `maxResources` propriedade em sua pilha ou desabilitar a validação definindo como `maxResources` 0.

> **Tip**
>
> Você pode obter uma contagem exata dos recursos em sua saída sintetizada usando o script utilitário a seguir. (Como todo AWS CDK desenvolvedor precisa do Node.js, o script é escrito em 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");
```

À medida que a contagem de recursos da sua pilha se aproxima do limite, considere a rearquitetura para reduzir o número de recursos que sua pilha contém: por exemplo, combinando algumas funções do Lambda ou dividindo sua pilha em várias pilhas. O CDK oferece suporte a referências entre pilhas, para que você possa separar a funcionalidade do seu aplicativo em pilhas diferentes da maneira que fizer mais sentido para você.

> **Note**
>
> AWS CloudFormation especialistas geralmente sugerem o uso de pilhas aninhadas como uma solução para o limite de recursos. O AWS CDK apóia essa abordagem por meio da `NestedStack`construção.

(voltar à lista)

Eu especifiquei três (ou mais) zonas de disponibilidade para meu grupo de Auto Scaling ou VPC, mas ela só foi implantada em duas

Para obter o número de zonas de disponibilidade que você solicita, especifique a conta e a região na `env` propriedade da pilha. Se você não especificar ambos, o, por padrão AWS CDK, sintetiza a pilha como independente do ambiente. Em seguida, você pode implantar a pilha em uma região específica usando AWS CloudFormation. Como algumas regiões têm apenas duas zonas de disponibilidade, um modelo independente do ambiente não usa mais do que duas.

> **Note**
>
> No passado, ocasionalmente, as regiões eram lançadas com apenas uma zona de disponibilidade. AWS CDK Pilhas independentes do ambiente não podem ser implantadas nessas regiões. No momento em que este artigo foi escrito, no entanto, todas as AWS regiões tinham pelo menos duas AZs.

Você pode alterar esse comportamento substituindo a propriedade `availability_zones` (`availablilityZones`Python:) da sua pilha para especificar explicitamente as zonas que você deseja usar.

Para obter mais informações sobre como especificar a conta e a região de uma pilha no momento da síntese, mantendo a flexibilidade de implantação em qualquer região, consulte. the section called “Ambientes do”

(voltar à lista)

Meu bucket do S3, tabela do DynamoDB ou outro recurso não é excluído quando eu emito **`cdk destroy`**

Por padrão, os recursos que podem conter dados do usuário têm uma propriedade `removalPolicy` (Python:`removal_policy`) de`RETAIN`, e o recurso não é excluído quando a pilha é destruída. Em vez disso, o recurso fica órfão da pilha. Em seguida, você deve excluir o recurso manualmente depois que a pilha for destruída. Até que você faça isso, a reimplantação da pilha falhará. Isso ocorre porque o nome do novo recurso que está sendo criado durante a implantação está em conflito com o nome do recurso órfão.

Se você definir a política de remoção de um recurso como`DESTROY`, esse recurso será excluído quando a pilha for destruída.

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 não é possível excluir um bucket do Amazon S3 que não esteja vazio. Se você definir a política de remoção de um bucket do Amazon S3 como “e ele contiver dados”, a tentativa de destruir a pilha falhará porque o bucket não pode ser excluído. `DESTROY` Você pode AWS CDK excluir os objetos no bucket antes de tentar destruí-lo definindo a `autoDeleteObjects` prop do bucket como. `true`

(voltar à lista)

Chaves OpenPGP para o e jsii AWS CDK

Este tópico contém chaves OpenPGP atuais e históricas para o AWS CDK e jsii.

Teclas atuais

Essas chaves devem ser usadas para validar as versões atuais do AWS CDK e do jsii.

AWS CDK Chave OpenPGP

ID da chave:	0x42B9CF2286CD987A
Digite:	RSA
Tamanho:	4096/4096
Criado:	2022-07-05
Expira:	2026-07-04
ID do usuário:	Kit de desenvolvimento de nuvem da AWS < aws-cdk@amazon.com >
Impressão digital da chave:	69B5 2D5B A295 1D11 FA65 413B 42B9 CF22 86CD 987A

Selecione o ícone “Copiar” para copiar a seguinte chave OpenPGP:

```
-----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-----
```

chave jsii OpenPGP

ID da chave:	0x056C4E15DAE3D8D9
Digite:	RSA
Tamanho:	4096/4096
Criado:	2022-07-05
Expira:	2026-07-04
ID do usuário:	Equipe AWS JSII < aws-jsii@amazon.com >
Impressão digital da chave:	1E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

Selecione o ícone “Copiar” para copiar a seguinte chave OpenPGP:

```
-----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-----
```

Chaves históricas

Essas chaves podem ser usadas para validar as versões do AWS CDK e do jsii antes de 2022-07-05.

> **Important**
>
> Novas chaves são criadas antes que as anteriores expirem. Como resultado, a qualquer momento, mais de uma chave pode ser válida. As chaves são usadas para assinar artefatos a partir do dia em que são criadas, então use a chave emitida mais recentemente em que a validade das chaves se sobrepõe.

AWS CDK Chave OpenPGP (2022-04-07)

> **Note**
>
> Essa chave não foi usada para assinar AWS CDK artefatos após 2022-07-05.

ID da chave:	0x015584281F44A3C3
Digite:	RSA
Tamanho:	4096/4096
Criado:	2022-04-07
Expira:	2026-04-06
ID do usuário:	Kit de desenvolvimento de nuvem da AWS < aws-cdk@amazon.com >
Impressão digital da chave:	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

Selecione o ícone “Copiar” para copiar a seguinte chave OpenPGP:

```
-----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-----
```

chave jsii OpenPGP (2022-04-07)

> **Note**
>
> Essa chave não foi usada para assinar artefatos jsii após 2022-07-05.

ID da chave:	0x985F5BC974B79356
Digite:	RSA
Tamanho:	4096/4096
Criado:	2022-04-07
Expira:	2026-04-06
ID do usuário:	Equipe AWS JSII < aws-jsii@amazon.com >
Impressão digital da chave:	35A7 1785 8FA6 282D C5AC CD95 985F 5BC9 74B7 9356

Selecione o ícone “Copiar” para copiar a seguinte chave OpenPGP:

```
-----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 Chave OpenPGP (2018-06-19)

ID da chave:	0x0566A784E17F3870
Digite:	RSA
Tamanho:	4096/4096
Criado:	2018-06-19
Expira:	2022-06-18
ID do usuário:	Equipe do AWS CDK < aws-cdk@amazon.com >

Impressão digital da chave:	E88B E3B6 F0B1 E350 9E36 4F96 0566 A784 E17F 3870

Selecione o ícone “Copiar” para copiar a seguinte chave OpenPGP:

```
-----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-----
```

chave jsii OpenPGP (2018-08-06)

ID da chave:	0x1C7ACE4CB2A1B93A
Digite:	RSA

Tamanho:	4096/4096
Criado:	2018-08-06
Expira:	2022-08-05
ID do usuário:	Equipe AWS JSII < aws-jsii@amazon.com >
Impressão digital da chave:	85EF 6522 4CE2 18EC 72DB 28EC 17CA CE4C B2A1 B93A

Selecione o ícone “Copiar” para copiar a seguinte chave OpenPGP:

```
-----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 Histórico do Developer Guide

Consulte Lançamentos para obter informações sobre AWS CDK lançamentos. AWS CDK É atualizado aproximadamente uma vez por semana. As versões de manutenção podem ser lançadas entre lançamentos semanais para resolver problemas críticos. Cada versão inclui um AWS CDK kit de ferramentas (CDK CLI) AWS , uma biblioteca de construção e uma referência de API correspondentes. As atualizações deste Guia geralmente não são sincronizadas com as AWS CDK versões.

> **Note**
>
> A tabela abaixo representa marcos significativos da documentação. Corrigimos erros e melhoramos o conteúdo continuamente.

Alteração	Descrição	Data
Adicionar documentação para o recurso CDK Migrate	Use o AWS CDK CLI `cdk migrate` comando para migrar AWS recursos implantados, AWS CloudFormation pilhas implantadas e modelos locais para o. AWS CloudFormation AWS CDK Para obter mais informações, consulte Migrar para AWS CDK.	2 de fevereiro de 2022
Atualizações de práticas recomendadas do IAM	Guia atualizado para alinhamento com as práticas recomendadas do IAM. Para obter mais informações, consulte Práticas recomendadas de segurança no IAM.	23 de março de 2023

Documento `cdk.json`	Adicione a documentação dos valores de `cdk.json` configuração.	20 de abril de 2022
Gerenciamento de dependências	Adicione um tópico sobre como gerenciar dependências com o. AWS CDK	7 de abril de 2022
Remova os colchetes duplos dos exemplos de Java	Substitua esse antipadrão pelo Java 9 `Map.of` por toda parte.	9 de março de 2022
AWS CDK versão v2	A versão 2 do Guia do AWS CDK Desenvolvedor foi lançada. Histórico de documentos do CDK v1.	4 de dezembro de 2021

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.