Guida per gli sviluppatori

AWS Cloud Development Kit (AWS CDK) v2

Version 2

AWS Cloud Development Kit (AWS CDK) v2: Guida per gli sviluppatori

Table of Contents

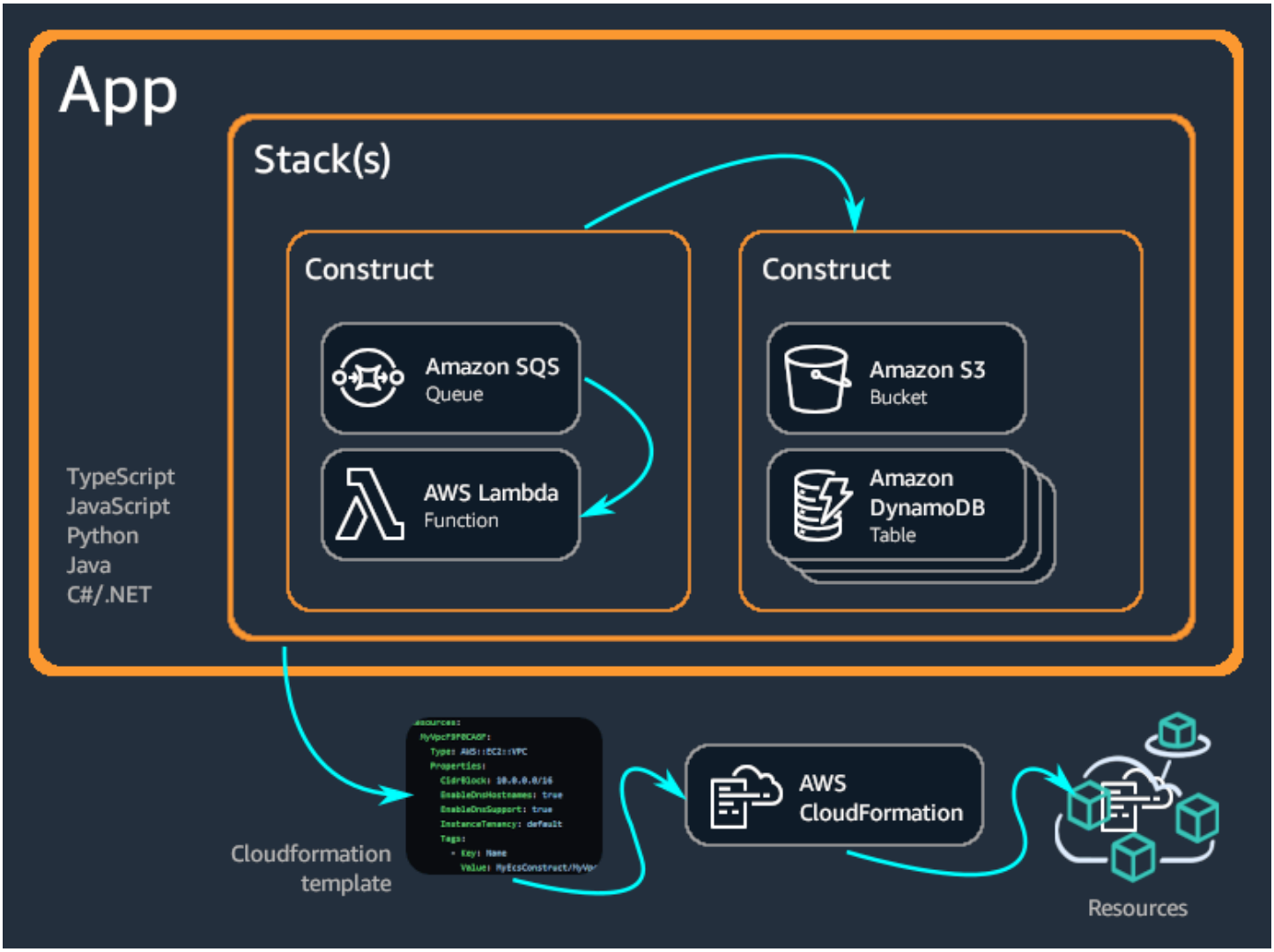
Che cos'è il AWS CDK?

AWS Cloud Development Kit (AWS CDK) È un framework di sviluppo software open source per definire l'infrastruttura cloud in codice e fornirla tramite. AWS CloudFormation

AWS CDK Si compone di due parti principali:

- AWS CDK Construct Library: una raccolta di parti di codice modulari e riutilizzabili prescritte, chiamate costrutti, che è possibile utilizzare, modificare e integrare per sviluppare rapidamente l'infrastruttura. L'obiettivo di AWS CDK Construct Library è ridurre la complessità richiesta per definire e integrare i AWS servizi durante la creazione di applicazioni. AWS
- AWS CDK Toolkit: uno strumento a riga di comando per interagire con le app CDK. Usa il AWS CDK Toolkit per creare, gestire e distribuire i tuoi progetti. AWS CDK

I AWS CDK supportiTypeScript,JavaScript,, Python JavaC#/.Net, e. Go Puoi utilizzare uno qualsiasi di questi linguaggi di programmazione supportati per definire componenti cloud riutilizzabili noti come costrutti. Li componi insieme in pile e app. Quindi, distribuisci le tue applicazioni CDK per fornire o AWS CloudFormation aggiornare le tue risorse.



Argomenti

Vantaggi della AWS CDK

Utilizzalo AWS CDK per sviluppare applicazioni affidabili, scalabili ed economiche nel cloud con la notevole potenza espressiva di un linguaggio di programmazione. Questo approccio offre molti vantaggi, tra cui:

Sviluppa e gestisci la tua infrastruttura come codice (IaC)

Pratica l'infrastruttura come codice per creare, implementare e gestire l'infrastruttura in modo programmatico, descrittivo e dichiarativo. Con IaC, trattate l'infrastruttura nello stesso modo in cui gli sviluppatori trattano il codice. Ciò si traduce in un approccio scalabile e strutturato alla gestione dell'infrastruttura. Per ulteriori informazioni su IaC, consulta Infrastructure as code nell'Introduzione a DevOps on AWS Whitepaper.

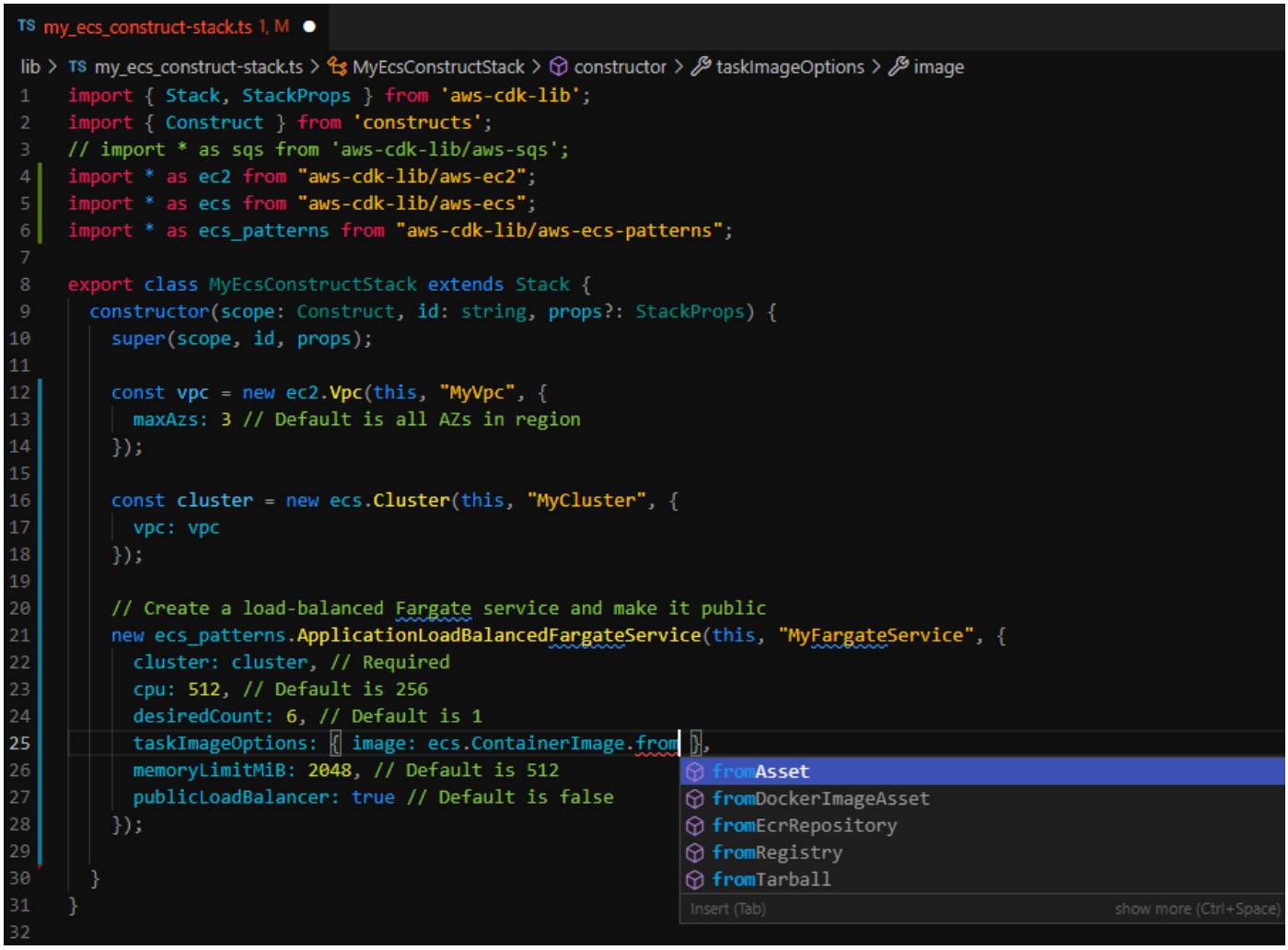
Con AWS CDK, puoi mettere l'infrastruttura, il codice dell'applicazione e la configurazione in un unico posto, assicurandoti di avere un sistema completo e implementabile sul cloud in ogni fase fondamentale. Utilizza le migliori pratiche di ingegneria del software come revisioni del codice, test unitari e controllo del codice sorgente per rendere la tua infrastruttura più robusta.

Definisci la tua infrastruttura cloud utilizzando linguaggi di programmazione generici

Con AWS CDK, puoi utilizzare uno dei seguenti linguaggi di programmazione per definire la tua infrastruttura cloud:TypeScript,,JavaScript, Python JavaC#/.Net, e. Go Scegli il tuo linguaggio preferito e usa elementi di programmazione come parametri, condizionali, loop, composizione ed ereditarietà per definire il risultato desiderato della tua infrastruttura.

Utilizza lo stesso linguaggio di programmazione per definire l'infrastruttura e la logica dell'applicazione.

Sfrutta i vantaggi dello sviluppo dell'infrastruttura nel tuo IDE (Integrated Development Environment) preferito, come l'evidenziazione della sintassi e il completamento intelligente del codice.



Implementa l'infrastruttura tramite AWS CloudFormation

AWS CDK si integra con AWS CloudFormation per implementare e fornire l'infrastruttura su. AWS AWS CloudFormation è un servizio gestito Servizio AWS che offre un supporto completo per le configurazioni di risorse e proprietà per i servizi di provisioning su. AWS Con AWS CloudFormation, è possibile eseguire le implementazioni dell'infrastruttura in modo prevedibile e ripetuto, con ripristino in caso di errore. Se lo conosci già AWS CloudFormation, non devi imparare a usare un nuovo servizio di gestione IaC quando inizi a usare. AWS CDK

Inizia a sviluppare rapidamente la tua applicazione con i costrutti

Sviluppa più velocemente utilizzando e condividendo componenti riutilizzabili chiamati costrutti. Usa costrutti di basso livello per definire le singole AWS CloudFormation risorse e le relative proprietà. Utilizza costrutti di alto livello per definire rapidamente componenti più grandi della tua applicazione, con impostazioni predefinite ragionevoli e sicure per AWS le tue risorse, definendo una maggiore infrastruttura con meno codice.

Crea i tuoi costrutti personalizzati per i tuoi casi d'uso unici e condividili all'interno dell'organizzazione o anche con il pubblico.

Esempio di AWS CDK

Di seguito è riportato un esempio di utilizzo della libreria AWS CDK Constructs per creare un servizio Amazon Elastic Container Service (Amazon ECS) con tipo di avvio. AWS Fargate (Fargate) Per maggiori dettagli su questo esempio, consulta. the 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

}
```

Questa classe produce un AWS CloudFormation modello di oltre 500 righe. La distribuzione dell' AWS CDK app produce più di 50 risorse dei seguenti tipi.

- 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

Il AWS CDKGitHub repository

Per il AWS CDKGitHub repository ufficiale, vedi aws-cdk. Qui puoi segnalare problemi, visualizzare la nostra licenza, tenere traccia dei rilasci e altro ancora.

Poiché AWS CDK è open source, il team ti incoraggia a contribuire per renderlo uno strumento ancora migliore. Per i dettagli, consulta Contributing to. AWS Cloud Development Kit (AWS CDK)

Il riferimento all' AWS CDK API

La AWS CDK Construct Library fornisce API per definire l'applicazione CDK e aggiungere costrutti CDK all'applicazione. Per ulteriori informazioni, consulta la Documentazione di riferimento delle API di AWS CDK.

Il modello di programmazione Construct

Il Construct Programming Model (CPM) estende i concetti alla base AWS CDK in altri domini. Altri strumenti che utilizzano il CPM includono:

- CDK per Terraform (CDKtf)
- CDK per Kubernetes (CDK8s)
- Projen, per creare configurazioni di progetto

Il Construct Hub

Construct Hub è un registro online in cui è possibile trovare, pubblicare e condividere librerie open source AWS CDK .

Passaggi successivi

Per iniziare a usare il AWS CDK, consulta. Iniziare con AWS CDK

Ulteriori informazioni

Per continuare a conoscere il AWS CDK, consulta quanto segue:

- AWS CDK concetti— Concetti e termini importanti per AWS CDK.
- AWS CDK Workshop: workshop pratico per imparare e utilizzare il. AWS CDK
- AWS CDK Patterns: raccolta open source di modelli di architettura AWS serverless, creata per te da esperti. AWS CDK AWS
- AWS CDK esempi di codice: GitHub archivio di progetti di esempio. AWS CDK
- cdk.dev — Hub gestito dalla community per, incluso uno spazio di lavoro comunitario. AWS CDKSlack
- Awesome CDK: GitHub repository contenente un elenco curato di progetti AWS CDK open source, guide, blog e altre risorse.
- AWS Solutions Constructs: modelli di configurazione IaC (Infrastructure as Code) verificati che possono essere facilmente assemblati in applicazioni pronte per la produzione.
- AWS Developer Tools Blog: post del blog filtrati per. AWS CDK
- AWS CDK on Stack Overflow - Domande contrassegnate con aws-cdk on. Stack Overflow
- AWS CDK tutorial per AWS Cloud9 — Tutorial sull'utilizzo di AWS CDK con l' AWS Cloud9 ambiente di sviluppo.

Per ulteriori informazioni sugli argomenti correlati a AWS CDK, consulta quanto segue:

- AWS CloudFormation concetti: poiché AWS CDK è progettato per funzionare AWS CloudFormation, ti consigliamo di apprendere e comprendere AWS CloudFormation i concetti chiave.
- AWS Glossario: definizioni dei termini chiave utilizzati in tutto AWS.

Per ulteriori informazioni sugli strumenti correlati AWS CDK che possono essere utilizzati per semplificare lo sviluppo e la distribuzione di applicazioni serverless, consulta quanto segue:

- AWS Serverless Application Model— Uno strumento di sviluppo open source che semplifica e migliora l'esperienza di creazione ed esecuzione di applicazioni serverless su. AWS
- AWSChalice— Un framework per la scrittura di app serverless. Python

AWS CDK concetti

Scopri i concetti fondamentali alla base di AWS Cloud Development Kit (AWS CDK).

AWS CDK e IaC

AWS CDK È un framework open source che puoi utilizzare per gestire la tua AWS infrastruttura tramite codice. Questo approccio è noto come infrastructure as code (IaC). Gestendo e fornendo l'infrastruttura come codice, trattate l'infrastruttura nello stesso modo in cui gli sviluppatori trattano il codice. Ciò offre molti vantaggi, come il controllo delle versioni e la scalabilità. Per ulteriori informazioni su IaC, consulta What is Infrastructure as Code?

AWS CDK e AWS CloudFormation

AWS CDK È strettamente integrato con. AWS CloudFormation AWS CloudFormation è un servizio completamente gestito che puoi utilizzare per gestire e fornire la tua infrastruttura. AWS Con AWS CloudFormation, definisci la tua infrastruttura in modelli e li distribuisci su AWS CloudFormation. Il AWS CloudFormation servizio effettua quindi il provisioning dell'infrastruttura in base alla configurazione definita nei modelli.

AWS CloudFormation i modelli sono dichiarativi, ossia dichiarano lo stato o il risultato desiderato dell'infrastruttura. Utilizzando JSON o YAML, dichiarate la vostra AWS infrastruttura definendo risorse e proprietà. AWS Le risorse rappresentano i numerosi servizi disponibili AWS e le proprietà rappresentano la configurazione desiderata di tali servizi. Quando si distribuisce il modello su AWS CloudFormation, le risorse e le relative proprietà configurate vengono fornite come descritto nel modello.

Con AWS CDK, è possibile gestire l'infrastruttura in modo imperativo, utilizzando linguaggi di programmazione generici. Invece di limitarsi a definire lo stato desiderato in modo dichiarativo, è possibile definire la logica o la sequenza necessaria per raggiungere lo stato desiderato. Ad esempio, è possibile utilizzare `if` istruzioni o loop condizionali che determinano come raggiungere lo stato finale desiderato per l'infrastruttura.

L'infrastruttura creata con il AWS CDK viene infine tradotta o sintetizzata in AWS CloudFormation modelli e distribuita utilizzando il servizio. AWS CloudFormation Pertanto, sebbene AWS CDK offra un approccio diverso alla creazione dell'infrastruttura, ne usufruirete comunque dei vantaggi AWS CloudFormation, come un ampio supporto per la configurazione AWS delle risorse e solidi processi di implementazione.

Per saperne di più AWS CloudFormation, vedi Cos'è AWS CloudFormation? nella Guida AWS CloudFormation per l'utente.

AWS CDK e astrazioni

Con AWS CloudFormation, è necessario definire ogni dettaglio della configurazione delle risorse. Ciò offre il vantaggio di avere il controllo completo sull'infrastruttura. Tuttavia, ciò richiede l'apprendimento, la comprensione e la creazione di modelli affidabili che contengano dettagli sulla configurazione delle risorse e le relazioni tra le risorse, come le autorizzazioni e le interazioni basate sugli eventi.

Con AWS CDK, puoi avere lo stesso controllo sulle configurazioni delle risorse. Tuttavia, offre AWS CDK anche potenti astrazioni, che possono accelerare e semplificare il processo di sviluppo dell'infrastruttura. Ad esempio, AWS CDK include costrutti che forniscono configurazioni predefinite ragionevoli e metodi di supporto che generano codice standard per l'utente. Offre AWS CDK anche strumenti, come AWS CDK Command Line Interface (AWS CDK CLI), che eseguono azioni di gestione dell'infrastruttura al posto vostro.

Scopri di più sui AWS CDK concetti fondamentali

Interagire con AWS CDK

Quando si utilizza con AWS CDK, interagirai principalmente con la AWS Construct Library e il. AWS CDK CLI

Sviluppando con AWS CDK

AWS CDK Può essere scritto in qualsiasi linguaggio di programmazione supportato. Si inizia con un progetto CDK, che contiene una struttura di cartelle e file, comprese le risorse. All'interno del progetto, create un'applicazione CDK. All'interno dell'app, definisci uno stack, che rappresenta direttamente uno CloudFormation stack. All'interno dello stack, definisci AWS le tue risorse e proprietà utilizzando costrutti.

Implementazione con AWS CDK

Implementate le app CDK in un ambiente. AWS Prima della distribuzione, è necessario eseguire un bootstrap una tantum per preparare l'ambiente.

Ulteriori informazioni

Per ulteriori informazioni sui concetti AWS CDK fondamentali, consulta gli argomenti di questa sezione.

Linguaggi di programmazione compatibili

Offre un supporto AWS Cloud Development Kit (AWS CDK) di prima classe per i seguenti linguaggi di programmazione generici:

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

In teoria possono essere utilizzati anche altri .NET CLR linguaggi JVM e, ma al momento non offriamo supporto ufficiale.

> **Note**
>
> Al momento questa guida non include istruzioni o esempi di codice a Go partethe section called “In Go”.

AWS CDK È sviluppato in una lingua,TypeScript. Per supportare le altre lingue, AWS CDK utilizza uno strumento chiamato JSIIa generare associazioni linguistiche.

Cerchiamo di offrire le consuete convenzioni di ogni lingua per rendere lo sviluppo il AWS CDK più naturale e intuitivo possibile. Ad esempio, distribuiamo i moduli di AWS Construct Library utilizzando l'archivio standard della lingua preferita e voi li installate utilizzando il gestore di pacchetti standard del linguaggio. I metodi e le proprietà vengono inoltre denominati utilizzando gli schemi di denominazione consigliati nella lingua in uso.

Di seguito sono riportati alcuni esempi di codice:

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**
>
> Questi frammenti di codice sono solo a scopo illustrativo. Sono incompleti e non funzioneranno così come sono.

La AWS Construct Library è distribuita utilizzando gli strumenti di gestione dei pacchetti standard di ogni lingua, tra cui NPMPyPi,Maven, eNuGet. Forniamo anche una versione dell'AWS CDK API Reference per ogni lingua.

Per aiutarti a utilizzarlo AWS CDK nella tua lingua preferita, questa guida include i seguenti argomenti per le lingue supportate:

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

TypeScriptè stata la prima lingua supportata da AWS CDK, e gran parte del codice di AWS CDK esempio è scritto inTypeScript. Questa guida include un argomento specifico per mostrare come adattare il TypeScript AWS CDK codice per l'uso con le altre lingue supportate. Per ulteriori informazioni, consulta Confronto AWS CDKTypeScript con altri linguaggi.

AWS CDK progetti

Un AWS Cloud Development Kit (AWS CDK) progetto rappresenta i file e le cartelle che contengono il codice CDK. I contenuti varieranno in base al linguaggio di programmazione.

È possibile creare il AWS CDK progetto manualmente o con il AWS CDK comando Command Line Interface (AWS CDK CLI)`cdk init`. In questo argomento, faremo riferimento alla struttura del progetto e alle convenzioni di denominazione di file e cartelle create dalla CLI di AWS CDK. Puoi personalizzare e organizzare i tuoi progetti CDK in base alle tue esigenze.

> **Note**
>
> La struttura del progetto creata da AWS CDK CLI può variare tra le versioni nel tempo.

Argomenti

File e cartelle universali

.git

Se l'hai `git` installato, inizializza AWS CDK CLI automaticamente un Git repository per il tuo progetto. La `.git` directory contiene informazioni sul repository.

.gitignore

File di testo utilizzato da Git per specificare file e cartelle da ignorare.

README.md

File di testo che fornisce indicazioni di base e informazioni importanti per la gestione AWS CDK del progetto. Se necessario, modificate questo file per documentare informazioni importanti relative al progetto CDK.

cdk.json

File di configurazione per. AWS CDK Questo file fornisce istruzioni AWS CDK CLI su come eseguire l'app.

File e cartelle specifici della lingua

I file e le cartelle seguenti sono unici per ogni linguaggio di programmazione supportato.

TypeScript

Di seguito è riportato un esempio di progetto creato nella `my-cdk-ts-project` directory utilizzando il `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
```

.npmignore

File che specifica quali file e cartelle ignorare quando si pubblica un pacchetto nel registro. npm Questo file è simile a `.gitignore`, ma è specifico per npm i pacchetti.

bin/.ts my-cdk-ts-project

Il file dell'applicazione definisce l'app CDK. I progetti CDK possono contenere uno o più file di applicazione. I file dell'applicazione vengono memorizzati nella `bin` cartella.

Di seguito è riportato un esempio di file applicativo di base che definisce un'app CDK:

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

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

jest.config.js

File di configurazione per. Jest Jestè un framework JavaScript di test popolare.

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

Il file stack definisce lo stack CDK. All'interno dello stack, definisci AWS risorse e proprietà utilizzando costrutti.

Di seguito è riportato un esempio di file stack di base che definisce uno stack CDK:

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

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

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

node_modules

Cartella comune nei Node.js progetti che contengono dipendenze per il progetto.

package-lock.json

File di metadati che funziona con il file per gestire le versioni delle dipendenze`package.json`.

package.json

File di metadati comunemente usato nei progetti. Node.js Questo file contiene informazioni sul progetto CDK, come il nome del progetto, le definizioni degli script, le dipendenze e altre informazioni di importazione a livello di progetto.

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

Viene creata una cartella di test per organizzare i test per il progetto CDK. Viene inoltre creato un file di test di esempio.

È possibile scrivere test TypeScript e utilizzarli Jest per compilare il TypeScript codice prima di eseguire i test.

tsconfig.json

File di configurazione utilizzato nei TypeScript progetti che specifica le opzioni del compilatore e le impostazioni del progetto.

JavaScript

Di seguito è riportato un esempio di progetto creato nella `my-cdk-js-project` directory utilizzando il `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
```

.npmignore

File che specifica quali file e cartelle ignorare quando si pubblica un pacchetto nel registro. npm Questo file è simile a `.gitignore`, ma è specifico per npm i pacchetti.

bin/.js my-cdk-js-project

Il file dell'applicazione definisce l'app CDK. I progetti CDK possono contenere uno o più file di applicazione. I file dell'applicazione vengono memorizzati nella `bin` cartella.

Di seguito è riportato un esempio di file applicativo di base che definisce un'app CDK:

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

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

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

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

jest.config.js

File di configurazione per. Jest Jestè un framework JavaScript di test popolare.

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

Il file stack definisce lo stack CDK. All'interno dello stack, definisci AWS risorse e proprietà utilizzando costrutti.

Di seguito è riportato un esempio di file stack di base che definisce uno stack CDK:

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

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

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

module.exports = { MyCdkJsProjectStack }
```

node_modules

Cartella comune nei Node.js progetti che contengono dipendenze per il progetto.

package-lock.json

File di metadati che funziona con il file per gestire le versioni delle dipendenze`package.json`.

package.json

File di metadati comunemente usato nei progetti. Node.js Questo file contiene informazioni sul progetto CDK, come il nome del progetto, le definizioni degli script, le dipendenze e altre informazioni di importazione a livello di progetto.

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

Viene creata una cartella di test per organizzare i test per il progetto CDK. Viene inoltre creato un file di test di esempio.

È possibile scrivere test JavaScript e utilizzarli Jest per compilare il JavaScript codice prima di eseguire i test.

Python

Di seguito è riportato un esempio di progetto creato nella `my-cdk-py-project` directory utilizzando il `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

Il CDK crea CLI automaticamente un ambiente virtuale per il tuo progetto. La `.venv` directory si riferisce a questo ambiente virtuale.

app.py

Il file dell'applicazione definisce l'app CDK. I progetti CDK possono contenere uno o più file di applicazione.

Di seguito è riportato un esempio di file applicativo di base che definisce un'app 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

Directory che contiene i file dello stack. Il CDK CLI crea quanto segue qui:

- `__init__.py` — Un file di definizione Python del pacchetto vuoto.
- `my_cdk_py_project`— File che definisce lo stack CDK. Quindi definisci AWS risorse e proprietà all'interno dello stack utilizzando i costrutti.

Di seguito è riportato un esempio di file stack:

```
from aws_cdk import Stack

from constructs import Construct

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

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

requirements-dev.txt

File simile `arequirements.txt`, ma utilizzato per gestire le dipendenze specificamente per scopi di sviluppo piuttosto che di produzione.

requirements.txt

File comune utilizzato nei Python progetti per specificare e gestire le dipendenze dei progetti.

source.bat

Il file Batch viene utilizzato per configurare l'ambiente Python virtuale. Windows

test

Directory che contiene i test per il tuo progetto CDK.

Di seguito è riportato un esempio di test unitario:

```
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

Di seguito è riportato un esempio di progetto creato nella `my-cdk-java-project` directory utilizzando il `cdk init --language java` comando:

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

pom.xml

File che contiene informazioni di configurazione e metadati relativi al progetto CDK. Questo file fa parte di. Maven

src/main

Directory contenente i file dell'applicazione e dello stack.

Di seguito è riportato un esempio di file di applicazione:

```
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();
 }
}
```

Di seguito è riportato un esempio di file stack:

```
package com.myorg;

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

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

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

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

src/test

Directory contenente i file di test. Di seguito è riportato un esempio:

```
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#

Di seguito è riportato un esempio di progetto creato nella `my-cdk-csharp-project` directory utilizzando il `cdk init --language csharp` comando:

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

src/ MyCdkCsharpProject

Directory contenente i file dell'applicazione e dello stack.

Di seguito è riportato un esempio di file di applicazione:

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

Di seguito è riportato un esempio di file stack:

```
using Amazon.CDK;
using Constructs;

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

Questa directory contiene anche quanto segue:

- `GlobalSuppressions.cs`— File utilizzato per sopprimere avvisi o errori specifici del compilatore in tutto il progetto.
- `.csproj`— File basato su XML utilizzato per definire le impostazioni del progetto, le dipendenze e creare configurazioni.

MyCdkCsharpProjectsrc/ .sln

Microsoft Visual Studio Solution Fileutilizzato per organizzare e gestire progetti correlati.

Go

Di seguito è riportato un esempio di progetto creato nella `my-cdk-go-project` directory utilizzando il `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

File che contiene informazioni sul modulo e viene utilizzato per gestire le dipendenze e il controllo delle versioni del progetto. Go

my-cdk-go-project.vai

File che definisce l'applicazione CDK e gli stack.

Di seguito è riportato un esempio:

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

type MyCdkGoProjectStackProps struct {
 awscdk.StackProps
}

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

```
 var sprops awscdk.StackProps
 if props != nil {
  sprops = props.StackProps
 }
 stack := awscdk.NewStack(scope, &id, &sprops)
 // The code that defines your resources and properties go here

  return stack
}

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

func env() *awscdk.Environment {

 return nil
}
```

my-cdk-go-project_test.go

File che definisce un test di esempio.

Di seguito è riportato un esempio:

```
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 app

L' AWS Cloud Development Kit (AWS CDK) applicazione o l'app è una raccolta di uno o più stack CDK. Gli stack sono una raccolta di uno o più costrutti, che definiscono AWS risorse e proprietà. Pertanto, il raggruppamento complessivo degli stack e dei costrutti è noto come app CDK.

Argomenti

- Definizione delle app
- L'albero di costruzione
- Il ciclo di vita dell'app

Definizione delle app

Puoi creare un'app definendo un'istanza dell'app nel file dell'applicazione del tuo progetto. A tale scopo, importate e utilizzate il Appcostrutto dalla AWS Construct Library. Il App costrutto non richiede alcun argomento di inizializzazione. È l'unico costrutto che può essere usato come radice.

Le `Stack` classi `App` e della AWS Construct Library sono costrutti unici. Rispetto ad altri costrutti, non configurano AWS le risorse da soli. Vengono invece utilizzati per fornire un contesto agli altri costrutti. Tutti i costrutti che rappresentano AWS risorse devono essere definiti, direttamente o indirettamente, nell'ambito di un costrutto. `Stack` `Stack`i costrutti sono definiti nell'ambito di un costrutto. `App`

Le app vengono quindi sintetizzate per creare AWS CloudFormation modelli per gli stack. Di seguito è riportato un esempio:

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)
```

Gli stack all'interno di una singola app possono facilmente fare riferimento alle risorse e alle proprietà reciproche. AWS CDK Deduce le dipendenze tra gli stack in modo che possano essere distribuiti nell'ordine corretto. Puoi distribuire uno o tutti gli stack all'interno di un'app con un solo comando. `cdk deploy`

L'albero di costruzione

I costrutti vengono definiti all'interno di altri costrutti utilizzando l'`scope`argomento passato a ogni costrutto, con la App classe come radice. In questo modo, un' AWS CDK app definisce una gerarchia di costrutti nota come albero dei costrutti.

La radice di questo albero è la tua app, che è un'istanza della classe. App All'interno dell'app, crei un'istanza di uno o più stack. All'interno degli stack, si creano delle istanze di costrutti, che possono a loro volta creare istanze di risorse o altri costrutti, e così via lungo l'albero.

I costrutti sono sempre definiti esplicitamente nell'ambito di un altro costrutto, che crea relazioni tra i costrutti. Quasi sempre, dovresti passare `this` (in Python`self`) come ambito, indicando che il nuovo costrutto è figlio del costrutto corrente. Lo schema previsto è quello da `Construct`cui si ricava il costrutto e poi si istanziano i costrutti che utilizza nel suo costruttore.

Il passaggio esplicito dell'ambito consente a ciascun costrutto di aggiungersi all'albero, con questo comportamento interamente contenuto nella classe base. `Construct` Funziona allo stesso modo in tutte le lingue supportate da AWS CDK e non richiede personalizzazioni aggiuntive.

> **Important**
>
> Tecnicamente, è possibile passare a un ambito diverso da `this` quando si istanzia un costrutto. Puoi aggiungere costrutti ovunque nell'albero o anche in un altro stack nella stessa app. Ad esempio, potete scrivere una funzione in stile mixin che aggiunga costrutti a un ambito passato come argomento. La difficoltà pratica qui è che non potete facilmente assicurarvi che gli ID che scegliete per i vostri costrutti siano unici nell'ambito di qualcun altro. Questa pratica rende inoltre il codice più difficile da comprendere, gestire e riutilizzare. È quasi sempre meglio trovare un modo per esprimere le proprie intenzioni senza ricorrere all'abuso dell'argomento. `scope`

AWS CDK Utilizza gli ID di tutti i costrutti nel percorso dalla radice dell'albero a ogni costrutto figlio per generare gli ID univoci richiesti da. AWS CloudFormation Questo approccio significa che gli ID di

costruzione devono essere univoci solo all'interno del loro ambito, anziché all'interno dell'intero stack come in lingua nativa. AWS CloudFormation Tuttavia, se spostate un costrutto in un ambito diverso, l'ID univoco dello stack generato cambia e AWS CloudFormation non lo considererà la stessa risorsa.

L'albero dei costrutti è separato dai costrutti definiti nel codice. AWS CDK Tuttavia, è accessibile tramite l'`node`attributo di qualsiasi costrutto, che è un riferimento al nodo che rappresenta quel costrutto nell'albero. Ogni nodo è un'`Node`istanza, i cui attributi forniscono l'accesso alla radice dell'albero e agli ambiti principali e secondari del nodo.

1. `node.children`— I figli diretti del costrutto.
2. `node.id`— L'identificatore del costrutto all'interno del suo ambito.
3. `node.path`— Il percorso completo del costrutto, inclusi gli ID di tutti i suoi genitori.
4. `node.root`— La radice dell'albero di costruzione (l'app).
5. `node.scope`— L'ambito (genitore) del costrutto o non definito se il nodo è la radice.
6. `node.scopes`— Tutti i genitori del costrutto, fino alla radice.
7. `node.uniqueId`— L'identificatore alfanumerico univoco per questo costrutto all'interno dell'albero (per impostazione predefinita, generato da `node.path` e un hash).

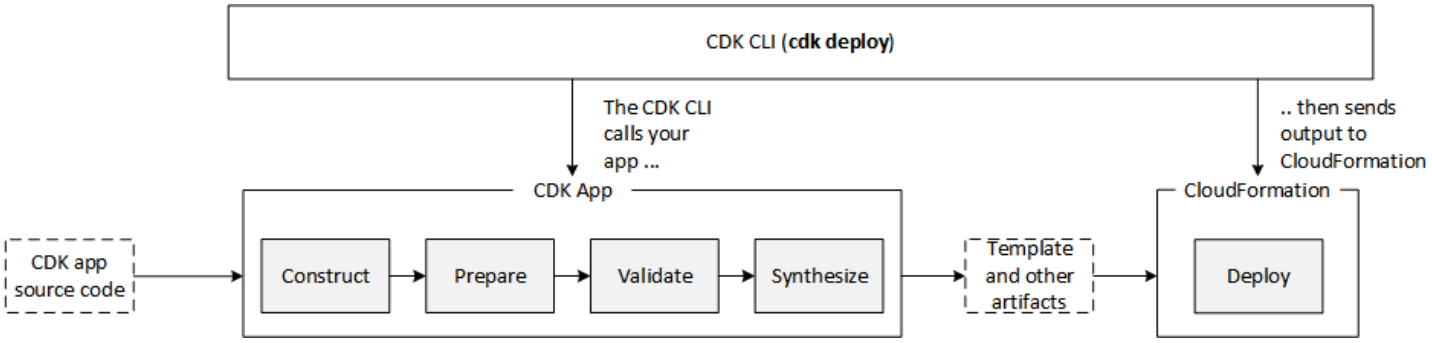
L'albero dei costrutti definisce un ordine implicito in cui i costrutti vengono sintetizzati nelle risorse del modello finale. AWS CloudFormation Dove una risorsa deve essere creata prima di un'altra, AWS CloudFormation oppure la AWS Construct Library generalmente deduce la dipendenza. Quindi si assicurano che le risorse vengano create nell'ordine giusto.

È inoltre possibile aggiungere una dipendenza esplicita tra due nodi utilizzando. `node.addDependency()` Per ulteriori informazioni, consulta Dipendenze nell'API Reference.AWS CDK

AWS CDK Fornisce un modo semplice per visitare ogni nodo dell'albero di costruzione ed eseguire un'operazione su ciascuno di essi. Per ulteriori informazioni, consulta the section called “Aspetti”.

Il ciclo di vita dell'app

Quando si distribuisce l'app CDK, si svolgono le seguenti fasi. Questo è noto come ciclo di vita dell'app:



Un' AWS CDK app attraversa le seguenti fasi del suo ciclo di vita.

- Costruzione (o inizializzazione): il codice crea un'istanza di tutti i costrutti definiti e poi li collega tra loro. In questa fase, tutti i costrutti (app, stack e i relativi costrutti secondari) vengono istanziati e la catena di costruttori viene eseguita. La maggior parte del codice dell'app viene eseguita in questa fase.
- Preparazione: tutti i costrutti che hanno implementato il `prepare` metodo partecipano a un ciclo finale di modifiche, per impostare il loro stato finale. La fase di preparazione avviene automaticamente. Come utente, non ricevi alcun feedback da questa fase. È raro che sia necessario utilizzare il gancio «prepare» e generalmente non è consigliato. Fate molta attenzione quando modificate l'albero di costruzione durante questa fase, perché l'ordine delle operazioni potrebbe influire sul comportamento.
- Convalida: tutti i costrutti che hanno implementato il `validate` metodo possono convalidarsi per garantire che si trovino in uno stato che venga distribuito correttamente. Riceverai una notifica di eventuali errori di convalida che si verificano durante questa fase. In genere, consigliamo di eseguire la convalida il prima possibile (di solito non appena si riceve un input) e di generare le eccezioni il prima possibile. L'esecuzione precoce della convalida migliora l'affidabilità poiché le tracce dello stack saranno più accurate e garantisce che il codice possa continuare a essere eseguito in modo sicuro.
- Sintesi: questa è la fase finale dell'esecuzione della tua AWS CDK app. Viene attivato da una chiamata a`app.synth()`, attraversa l'albero dei costrutti e richiama il metodo su tutti i costrutti. `synthesize` I costrutti che implementano `synthesize` possono partecipare alla sintesi ed emettere artefatti di implementazione nell'assemblaggio cloud risultante. Questi artefatti includono AWS CloudFormation modelli, bundle di AWS Lambda applicazioni, risorse di file e Docker immagini e altri elementi di distribuzione. the section called “Assemblaggi cloud”descrive l'output di questa fase. Nella maggior parte dei casi, non è necessario implementare il `synthesize` metodo.
- Implementazione: in questa fase, AWS CDK CLI prende gli artefatti di implementazione prodotti dall'assemblaggio cloud prodotti dalla fase di sintesi e li distribuisce in un ambiente. AWS Carica

le risorse su Amazon S3 e Amazon ECR o ovunque debbano andare. Quindi, avvia una AWS CloudFormation distribuzione per distribuire l'applicazione e creare le risorse.

Quando inizia la fase di AWS CloudFormation distribuzione, AWS CDK l'app è già terminata ed è uscita. Ciò comporta quanto segue:

- L' AWS CDK app non è in grado di rispondere agli eventi che si verificano durante la distribuzione, ad esempio la creazione di una risorsa o il completamento dell'intera distribuzione. Per eseguire il codice durante la fase di distribuzione, è necessario inserirlo nel AWS CloudFormation modello come risorsa personalizzata. Per ulteriori informazioni sull'aggiunta di una risorsa personalizzata all'app, consulta il AWS CloudFormation modulo o l'esempio di risorsa personalizzata.
- L' AWS CDK app potrebbe dover funzionare con valori che non possono essere conosciuti al momento dell'esecuzione. Ad esempio, se l' AWS CDK app definisce un bucket Amazon S3 con un nome generato automaticamente e recuperi l'attributo (`bucket.bucketName`Python:`bucket_name`), quel valore non è il nome del bucket distribuito. Invece, ottieni un valore. `Token` Per determinare se un particolare valore è disponibile, chiama `cdk.isUnresolved(value)` (Python:`is_unresolved`). Per informazioni dettagliate, vedi the section called “Gettoni”.

Assemblaggi cloud

La chiamata a `app.synth()` è ciò che dice loro di AWS CDK sintetizzare un assemblaggio cloud da un'app. In genere non si interagisce direttamente con gli assembly cloud. Sono file che includono tutto il necessario per distribuire l'app in un ambiente cloud. Ad esempio, include un AWS CloudFormation modello per ogni stack dell'app. Include anche una copia di qualsiasi file, risorsa o immagine Docker a cui fai riferimento nell'app.

Consulta le specifiche degli assemblaggi cloud per i dettagli sulla formattazione degli assembly cloud.

Per interagire con l'assembly cloud creato dall' AWS CDK app, in genere si utilizza il. AWS CDK CLI Tuttavia, qualsiasi strumento in grado di leggere il formato di assemblaggio cloud può essere utilizzato per distribuire l'app.

Esecuzione dell'app

Il CDK CLI deve sapere come eseguire la tua AWS CDK app. Se hai creato il progetto da un modello utilizzando il `cdk init` comando, il `cdk.json` file dell'app include una `app` chiave. Questa chiave

specifica il comando necessario per la lingua in cui è scritta l'app. Se la tua lingua richiede la compilazione, la riga di comando esegue questo passaggio prima di eseguire l'app, quindi non puoi dimenticare di farlo.

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 non hai creato il tuo progetto utilizzando il CDK CLI o se desideri sovrascrivere la riga di comando fornita in`cdk.json`, puoi usare l'--appopzione quando esegui il comando. `cdk`

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

La parte *`eseguibile`* del comando indica il comando da eseguire per eseguire l'applicazione CDK. Utilizzate le virgolette come illustrato, poiché tali comandi contengono spazi. Il *`comando cdk`* è un sottocomando simile a synth or deploy che indica al CDK CLI cosa vuoi fare con la tua app. Segui questa operazione con tutte le opzioni aggiuntive necessarie per quel sottocomando.

AWS CDK CLIPossono anche interagire direttamente con un assemblaggio cloud già sintetizzato. Per farlo, passa la directory in cui è archiviato l'assembly cloud. --app L'esempio seguente elenca gli stack definiti nell'assembly cloud archiviato in`./my-cloud-assembly`.

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

Stack

Uno AWS Cloud Development Kit (AWS CDK) stack è una raccolta di uno o più costrutti, che definiscono le risorse. AWS Ogni stack CDK rappresenta uno AWS CloudFormation stack nell'app CDK. Al momento dell'implementazione, i costrutti all'interno di uno stack vengono forniti come una singola unità, chiamata stack. AWS CloudFormation Per ulteriori informazioni sugli AWS CloudFormation stack, consulta Working with stacks nella Guida per l'utente.AWS CloudFormation

Poiché gli stack CDK vengono implementati tramite AWS CloudFormation stack, AWS CloudFormation si applicano quote e limitazioni. Per saperne di più, consulta le quote.AWS CloudFormation

Argomenti

- Definizione degli stack
- Utilizzo degli stack di

Definizione degli stack

Gli stack sono definiti nel contesto di un'app. Definite uno stack utilizzando la `Stack` classe della AWS Construct Library. Gli stack possono essere definiti in uno dei seguenti modi:

- Direttamente nell'ambito dell'app.
- Indirettamente da qualsiasi costrutto all'interno dell'albero.

L'esempio seguente definisce un'app CDK che contiene due stack:

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();
```

L'esempio seguente è un modello comune per definire uno stack su un file separato. Qui estendiamo o ereditiamo la `Stack` classe e definiamo un costruttore che accetta`scope`, `id` e. `props` Quindi, invochiamo il costruttore della `Stack` classe base utilizzando `super` con il comando received`scope`, e. `id` `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
}
```

L'esempio seguente dichiara una classe stack denominata `MyFirstStack` che include un singolo bucket Amazon S3.

TypeScript

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

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

JavaScript

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

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

Python

```
class MyFirstStack(Stack):

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

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

Java

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

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

```
        super(scope, id, props);

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

C#

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

Go

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

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

Tuttavia, questo codice ha dichiarato solo uno stack. Affinché lo stack venga effettivamente sintetizzato in un AWS CloudFormation modello e distribuito, deve essere istanziato. E, come tutti i costrutti CDK, deve essere istanziato in qualche contesto. Questo `App` è quel contesto.

Se utilizzi il modello di AWS CDK sviluppo standard, i tuoi stack vengono istanziati nello stesso file in cui crei l'istanza dell'oggetto. `App`

TypeScript

Il file con il nome del progetto (ad esempio,`hello-cdk.ts`) nella cartella del progetto. `bin`

JavaScript

Il file con il nome del progetto (ad esempio,`hello-cdk.js`) nella `bin` cartella del progetto.

Python

Il file `app.py` nella directory principale del progetto.

Java

Il file denominato*`ProjectName`*`App.java`, ad esempio`HelloCdkApp.java`, è annidato in profondità nella `src/main` directory.

C#

Il file denominato `Program.cs` sotto`src\`*`ProjectName`*, ad esempio`src\HelloCdk\Program.cs`.

L'API dello stack

L'oggetto Stack fornisce un'API ricca, che include quanto segue:

- `Stack.of(construct)`— Un metodo statico che restituisce lo Stack in cui è definito un costrutto. Ciò è utile se è necessario interagire con uno stack dall'interno di un costrutto riutilizzabile. La chiamata fallisce se non è possibile trovare uno stack nell'ambito.
- `stack.stackName`(Python:`stack_name`) — Restituisce il nome fisico dello stack. Come accennato in precedenza, tutti gli AWS CDK stack hanno un nome fisico che AWS CDK possono risolvere durante la sintesi.
- `stack.regione` `stack.account` — Restituisce rispettivamente la AWS regione e l'account in cui verrà distribuito questo stack. Queste proprietà restituiscono una delle seguenti:
  - L'account o la regione specificati esplicitamente al momento della definizione dello stack
  - Un token con codifica a stringa che si risolve negli AWS CloudFormation pseudo parametri per account e regione per indicare che questo stack è indipendente dall'ambiente

  Per informazioni su come vengono determinati gli ambienti per gli stack, consulta. the section called “Ambienti”
- `stack.addDependency(stack)`(Python: `stack.add_dependency(stack)` — Può essere usato per definire esplicitamente l'ordine di dipendenza tra due pile. Questo ordine viene rispettato dal cdk deploy comando quando si distribuiscono più stack contemporaneamente.

- `stack.tags`— Restituisce un tag TagManagerche è possibile utilizzare per aggiungere o rimuovere tag a livello di stack. Questo gestore di tag tagga tutte le risorse all'interno dello stack e tagga anche lo stack stesso quando viene creato. AWS CloudFormation
- `stack.partition`, `stack.urlSuffix` (Python:`url_suffix`), (`stack.stackId`Python:) e (`stack.notificationArn`Python: `stack_idnotification_arn`) — Restituiscono token che si risolvono nei rispettivi pseudo parametri, ad esempio. AWS CloudFormation `{ "Ref": "AWS::Partition" }` Questi token sono associati all'oggetto stack specifico in modo che il framework possa identificare i riferimenti cross-stack. AWS CDK
- `stack.availabilityZones`(Python:`availability_zones`) — Restituisce l'insieme di zone di disponibilità disponibili nell'ambiente in cui viene distribuito questo stack. Per gli stack indipendenti dall'ambiente, questo restituisce sempre un array con due zone di disponibilità. Per gli stack specifici dell'ambiente, AWS CDK interroga l'ambiente e restituisce l'esatto set di zone di disponibilità disponibili nella regione specificata.
- `stack.parseArn(arn)`and `stack.formatArn(comps)` (Python:`parse_arn`,`format_arn`) — Può essere usato per lavorare con Amazon Resource Names (ARN).
- `stack.toJsonString(obj)`(Python:`to_json_string`) — Può essere usato per formattare un oggetto arbitrario come stringa JSON che può essere incorporata in un modello. AWS CloudFormation L'oggetto può includere token, attributi e riferimenti, che vengono risolti solo durante la distribuzione.
- `stack.templateOptions`(Python:`template_options`) — Utilizzalo per specificare le opzioni del AWS CloudFormation modello, come Transform, Description e Metadata, per il tuo stack.

Utilizzo degli stack di

Per elencare tutti gli stack in un'app CDK, usa il comando. cdk ls L'esempio precedente mostrerebbe quanto segue:

```
stack1
stack2
```

Gli stack vengono distribuiti come parte di uno AWS CloudFormation stack in un ambiente. AWS L'ambiente copre uno specifico e. Account AWS Regione AWS

Quando si esegue il cdk synth comando per un'app con più stack, l'assembly cloud include un modello separato per ogni istanza dello stack. Anche se i due stack sono istanze della stessa classe, li AWS CDK emette come due modelli singoli.

È possibile sintetizzare ogni modello specificando il nome dello stack nel comando. cdk synth L'esempio seguente sintetizza il modello per stack1.

```
$ cdk synth stack1
```

Questo approccio è concettualmente diverso dal modo in cui vengono normalmente utilizzati i AWS CloudFormation modelli, in cui un modello può essere distribuito più volte e parametrizzato tramite parametri.AWS CloudFormation Sebbene AWS CloudFormation i parametri possano essere definiti in AWS CDK, in genere sono sconsigliati perché AWS CloudFormation i parametri vengono risolti solo durante la distribuzione. Ciò significa che non è possibile determinarne il valore nel codice.

Ad esempio, per includere in modo condizionale una risorsa nell'app in base al valore di un parametro, è necessario impostare una AWS CloudFormation condizione e contrassegnare la risorsa con essa. Adotta un AWS CDK approccio in cui i modelli concreti vengono risolti in fase di sintesi. Pertanto, è possibile utilizzare un'istruzione if per verificare il valore e determinare se è necessario definire una risorsa o applicare un comportamento.

> **Note**
>
> AWS CDK Fornisce la massima risoluzione possibile durante il periodo di sintesi per consentire l'uso idiomatico e naturale del linguaggio di programmazione.

Come qualsiasi altro costrutto, le pile possono essere composte insieme in gruppi. Il codice seguente mostra un esempio di servizio composto da tre stack: un piano di controllo, un piano dati e stack di monitoraggio. Il costrutto del servizio viene definito due volte: una volta per l'ambiente beta e una volta per l'ambiente di produzione.

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();
    }
```

```
}
```

Questa AWS CDK app alla fine è composta da sei pile, tre per ogni ambiente:

```
$ cdk ls

betacpDA8372D3
betadataE23DB2BA
betamon632BD457
prodcp187264CE
proddataF7378CE5
prodmon631A1083
```

I nomi fisici degli AWS CloudFormation stack vengono determinati automaticamente in AWS CDK base al percorso di costruzione dello stack nell'albero. Per impostazione predefinita, il nome di uno stack deriva dall'ID del costrutto dell'oggetto. `Stack` Tuttavia, puoi specificare un nome esplicito usando il `stackName` prop (in Python,`stack_name`), come segue.

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"
```

```
});
```

Stack nidificati

Il NestedStackcostrutto offre un modo per aggirare il limite di AWS CloudFormation 500 risorse per gli stack. Uno stack annidato conta come una sola risorsa nello stack che lo contiene. Tuttavia, può contenere fino a 500 risorse, inclusi stack annidati aggiuntivi.

L'ambito di uno stack nidificato deve essere un costrutto or. `Stack` `NestedStack` Lo stack nidificato non deve essere dichiarato lessicalmente all'interno dello stack principale. È necessario solo passare lo stack principale come primo parametro (`scope`) quando si istanzia lo stack nidificato. A parte questa restrizione, la definizione dei costrutti in uno stack annidato funziona esattamente come in uno stack ordinario.

Al momento della sintesi, lo stack annidato viene sintetizzato in un proprio AWS CloudFormation modello, che viene caricato nello staging bucket al momento della distribuzione. AWS CDK Gli stack annidati sono legati allo stack principale e non vengono trattati come artefatti di distribuzione indipendenti. Non sono elencati da `cdk list` e non possono essere distribuiti da. `cdk deploy`

I riferimenti tra gli stack principali e gli stack annidati vengono tradotti automaticamente in parametri e output dello stack nei AWS CloudFormation modelli generati, come con qualsiasi riferimento cross-stack.

> **Warning**
>
> Le modifiche al livello di sicurezza non vengono visualizzate prima della distribuzione degli stack annidati. Queste informazioni vengono visualizzate solo per gli stack di primo livello.

Costrutti

I costrutti sono gli elementi costitutivi di base delle AWS Cloud Development Kit (AWS CDK) applicazioni. Un costrutto è un componente all'interno dell'applicazione che rappresenta una o più AWS CloudFormation risorse e la loro configurazione. Puoi creare la tua applicazione, pezzo per pezzo, importando e configurando costrutti.

I costrutti sono classi che importate nelle app CDK. I costrutti sono disponibili nella Construct Library. AWS È inoltre possibile creare e distribuire costrutti personalizzati o utilizzare costrutti creati da sviluppatori di terze parti.

I costrutti fanno parte del Construct Programming Model (CPM). Sono disponibili per l'uso con altri strumenti come CDK for Terraform (CDKtf), CDK for (CDK8s) e. Kubernetes Projen

Argomenti

AWS Costruisci una libreria

La AWS Construct Library contiene una raccolta di costrutti sviluppati e gestiti da. AWSÈ organizzata in vari moduli che contengono costrutti che rappresentano tutte le risorse disponibili su. AWS Per informazioni di riferimento, consulta l'AWS CDK API Reference.

Viene chiamato `aws-cdk-lib` il pacchetto CDK principale che contiene la maggior parte della AWS Construct Library. Contiene anche classi base come `Stack` e. `App`

Il nome effettivo del pacchetto CDK principale varia in base alla lingua.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Go

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

> **Note**
>
> Se hai creato un progetto CDK utilizzando, non è necessario installarlo manualmente. cdk init `aws-cdk-lib`

La AWS Construct Library contiene anche il `constructs` pacchetto con la classe `Construct` base. È inclusa nel suo pacchetto perché viene utilizzata da altri strumenti basati su costrutti oltre a AWS CDK, tra cui CDK for Terraform e CDK for Kubernetes.

Numerose terze parti hanno inoltre pubblicato costrutti compatibili con. AWS CDK Visita Construct Hub per esplorare l'ecosistema di partner di AWS CDK construct.

Costruisci livelli

I costrutti della AWS Construct Library sono suddivisi in tre livelli. Ogni livello offre un livello crescente di astrazione. Maggiore è l'astrazione, più facile è la configurazione e richiede meno esperienza. Più bassa è l'astrazione, maggiore è la personalizzazione disponibile, che richiede maggiore esperienza.

Costrutti di livello 1 (L1)

I costrutti L1, noti anche come risorse CFN, sono i costrutti di livello più basso e non offrono alcuna astrazione. Ogni costrutto L1 è mappato direttamente su una singola risorsa. AWS CloudFormation Con i costrutti L1, si importa un costrutto che rappresenta una risorsa specifica. AWS CloudFormation Definite quindi le proprietà della risorsa all'interno dell'istanza del costrutto.

I costrutti L1 sono ottimi da usare quando si ha familiarità AWS CloudFormation e si ha bisogno del controllo completo sulla definizione delle proprietà delle risorse. AWS

Nella AWS Construct Library, i costrutti L1 sono denominati a partire da`Cfn`, seguiti da un identificatore per la risorsa che rappresentano. AWS CloudFormation Ad esempio, il `CfnBucket` costrutto è un costrutto L1 che rappresenta una risorsa. `AWS::S3::Bucket` AWS CloudFormation

I costrutti L1 vengono generati dalla specifica della risorsa.AWS CloudFormation Se una risorsa esiste in AWS CloudFormation, sarà disponibile AWS CDK come costrutto L1. La disponibilità di nuove risorse o proprietà nella AWS Construct Library può richiedere fino a una settimana. Per ulteriori informazioni, consultate il riferimento ai tipi di AWS risorse e proprietà nella Guida per l'AWS CloudFormation utente.

Costrutti di livello 2 (L2)

I costrutti L2, noti anche come costrutti curati, sono sviluppati con cura dal team CDK e di solito sono il tipo di costrutto più utilizzato. I costrutti L2 vengono mappati direttamente su singole risorse, in modo simile ai costrutti L1. AWS CloudFormation Rispetto ai costrutti L1, i costrutti L2 forniscono un'astrazione di livello superiore tramite un'API intuitiva basata sugli intenti. I costrutti L2 includono configurazioni sensate delle proprietà predefinite, politiche di sicurezza basate sulle

migliori pratiche e generano automaticamente gran parte del codice standard e della logica di incollaggio.

I costrutti L2 forniscono anche metodi di supporto per la maggior parte delle risorse che semplificano e velocizzano la definizione di proprietà, autorizzazioni, interazioni basate su eventi tra le risorse e altro ancora.

La `s3.Bucket` classe è un esempio di costrutto L2 per una risorsa bucket Amazon Simple Storage Service (Amazon S3).

La AWS Construct Library contiene costrutti L2 designati stabili e pronti per l'uso in produzione. Per i costrutti L2 in fase di sviluppo, sono designati come sperimentali e offerti in un modulo separato.

Costrutti di livello 3 (L3)

I costrutti L3, noti anche come pattern, rappresentano il livello di astrazione più elevato. Ogni costrutto L3 può contenere una raccolta di risorse configurate per funzionare insieme per eseguire un'attività o un servizio specifico all'interno dell'applicazione. I costrutti L3 vengono utilizzati per creare intere AWS architetture per casi d'uso particolari nell'applicazione.

Per fornire progetti di sistema completi o parti sostanziali di un sistema più grande, i costrutti L3 offrono configurazioni di proprietà predefinite ponderate. Sono costruiti attorno a un approccio particolare per risolvere un problema e fornire una soluzione. Con i costrutti L3, puoi creare e configurare più risorse rapidamente, con il minor numero di input e codice.

La `ecsPatterns.ApplicationLoadBalancedFargateService` classe è un esempio di costrutto L3 che rappresenta un AWS Fargate servizio in esecuzione su un cluster Amazon Elastic Container Service (Amazon ECS) e gestito da un sistema di bilanciamento del carico delle applicazioni.

Analogamente ai costrutti L2, i costrutti L3 pronti per l'uso in produzione sono inclusi nella Construct Library. AWS Quelli in fase di sviluppo sono offerti in moduli separati.

Definizione dei costrutti

Composizione

La composizione è il modello chiave per definire astrazioni di livello superiore attraverso costrutti. Un costrutto di alto livello può essere composto da un numero qualsiasi di costrutti di livello inferiore. Da

una prospettiva dal basso verso l'alto, si utilizzano i costrutti per organizzare le singole risorse che si desidera distribuire. AWS Utilizzi tutte le astrazioni che ritieni più convenienti per il tuo scopo, con tutti i livelli di cui hai bisogno.

Con la composizione, definisci i componenti riutilizzabili e li condividi come qualsiasi altro codice. Ad esempio, un team può definire un costrutto che implementa le best practice aziendali per una tabella Amazon DynamoDB, tra cui backup, replica globale, scalabilità automatica e monitoraggio. Il team può condividere il costrutto internamente con altri team o pubblicamente.

I team possono utilizzare i costrutti come qualsiasi altro pacchetto di libreria. Quando la libreria viene aggiornata, gli sviluppatori hanno accesso ai miglioramenti e alle correzioni di bug della nuova versione, in modo simile a qualsiasi altra libreria di codici.

Inizializzazione

I costrutti sono implementati in classi che estendono la classe di base `Construct`. Definite un costrutto istanziando la classe. Tutti i costrutti accettano tre parametri al momento dell'inizializzazione:

- scope: il genitore o il proprietario del costrutto. Può trattarsi di una pila o di un altro costrutto. L'ambito determina la posizione del costrutto nell'albero del costrutto. Di solito si dovrebbe passare `this` (`self`inPython), che rappresenta l'oggetto corrente, per l'ambito.
- id — Un identificatore che deve essere univoco all'interno dell'ambito. L'identificatore funge da namespace per tutto ciò che è definito all'interno del costrutto. Viene utilizzato per generare identificatori univoci, come nomi di risorse e ID logici. AWS CloudFormation

  Gli identificatori devono essere unici solo all'interno di un ambito. Ciò consente di creare istanze e riutilizzare i costrutti senza preoccuparsi dei costrutti e degli identificatori che potrebbero contenere e consente di comporre i costrutti in astrazioni di livello superiore. Inoltre, gli ambiti consentono di fare riferimento a gruppi di costrutti contemporaneamente. Gli esempi includono l'etichettatura o la specificazione di dove verranno distribuiti i costrutti.
- props: un insieme di proprietà o argomenti di parole chiave, a seconda della lingua, che definiscono la configurazione iniziale del costrutto. I costrutti di livello superiore forniscono più impostazioni predefinite e, se tutti gli elementi prop sono opzionali, potete omettere completamente il parametro props.

Configurazione

La maggior parte dei costrutti accetta `props` come terzo argomento (o in Python, argomenti di parole chiave), una raccolta di nomi/valori che definisce la configurazione del costrutto. L'esempio seguente definisce un bucket con crittografia AWS Key Management Service (AWS KMS) e hosting di siti Web statici abilitati. Poiché non specifica esplicitamente una chiave di crittografia, il `Bucket` costrutto ne definisce una nuova `kms.Key` e la associa al 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"),
})
```

Interazione con i costrutti

I costrutti sono classi che estendono la classe Construct di base. Dopo aver creato un'istanza di un costrutto, l'oggetto construct espone un insieme di metodi e proprietà che consentono di interagire con il costrutto e di passarlo come riferimento ad altre parti del sistema.

Il AWS CDK framework non impone alcuna restrizione alle API dei costrutti. Gli autori possono definire qualsiasi API desiderino. Tuttavia, AWS i costrutti inclusi nella AWS Construct Library, ad esempios3.Bucket, seguono linee guida e modelli comuni. Ciò fornisce un'esperienza coerente su tutte le AWS risorse.

La maggior parte dei AWS costrutti dispone di una serie di metodi di concessione che è possibile utilizzare per concedere le autorizzazioni AWS Identity and Access Management (IAM) su quel costrutto a un principale. L'esempio seguente concede al gruppo IAM l'data-scienceautorizzazione alla lettura dal bucket Amazon raw-data S3.

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)
```

Un altro schema comune prevede che AWS i costrutti impostino uno degli attributi della risorsa a partire dai dati forniti altrove. Gli attributi possono includere Amazon Resource Names (ARN), nomi o URL.

Il codice seguente definisce una AWS Lambda funzione e la associa a una coda Amazon Simple Queue Service (Amazon SQS) tramite l'URL della coda in una variabile di 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()),
 },
})
```

Per informazioni sui modelli di API più comuni nella Construct Library, consulta AWS . the section called “Risorse”

L'app e lo stack costruiscono

Le `Stack` classi `App` e della AWS Construct Library sono costrutti unici. Rispetto ad altri costrutti, non configurano AWS le risorse da soli. Vengono invece utilizzati per fornire un contesto agli altri costrutti. Tutti i costrutti che rappresentano AWS risorse devono essere definiti, direttamente o indirettamente, nell'ambito di un costrutto. `Stack` `Stack`i costrutti sono definiti nell'ambito di un costrutto. `App`

Per ulteriori informazioni sulle app CDK, consulta. AWS CDK app Per ulteriori informazioni sugli stack CDK, consulta. Stack

L'esempio seguente definisce un'app con un singolo stack. All'interno dello stack, viene utilizzato un costrutto L2 per configurare una risorsa bucket Amazon S3.

TypeScript

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

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

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

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

JavaScript

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

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

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

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

Python

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

class HelloCdkStack(Stack):

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

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

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

Java

Stack definito nel file: `HelloCdkStack.java`

```
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();
    }
}
```

App definita nel `HelloCdkApp.java` file:

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

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

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

        app.synth();
    }
}
```

C#

```
using Amazon.CDK;
```

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

namespace HelloCdkApp
{
    internal static class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new HelloCdkStack(app, "HelloCdkStack");
            app.Synth();
        }
    }

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

Go

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

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

 return stack
}
```

Lavorare con i costrutti

Lavorare con i costrutti L1

I costrutti L1 si collegano direttamente alle singole risorse. AWS CloudFormation È necessario fornire la configurazione richiesta per la risorsa.

In questo esempio, creiamo un `bucket` oggetto utilizzando il `CfnBucket` costrutto 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"),
 })
```

Le proprietà di costruzione che non sono semplici booleani, stringhe, numeri o contenitori vengono gestite in modo diverso nelle lingue supportate.

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

In Python, queste proprietà sono rappresentate da tipi definiti come classi interne del costrutto L1. Ad esempio, la proprietà opzionale `cors_configuration` di a `CfnBucket` richiede un wrapper di tipo. `CfnBucket.CorsConfigurationProperty` Qui stiamo definendo un'`cors_configuration`istanza`CfnBucket`.

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

```
            allowed_methods=["GET"]
        )]
    )
)
```

Java

In Java, queste proprietà sono rappresentate da tipi definiti come classi interne del costrutto L1. Ad esempio, la proprietà opzionale `corsConfiguration` di a `CfnBucket` richiede un wrapper di tipo. `CfnBucket.CorsConfigurationProperty` Qui stiamo definendo un'`corsConfiguration`istanza`CfnBucket`.

```
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#

In C#, queste proprietà sono rappresentate da tipi definiti come classi interne del costrutto L1. Ad esempio, la proprietà opzionale `CorsConfiguration` di a `CfnBucket` richiede un wrapper di tipo. `CfnBucket.CorsConfigurationProperty` Qui stiamo definendo un'`CorsConfiguration`istanza`CfnBucket`.

```
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

In Go, questi tipi vengono denominati utilizzando il nome del costrutto L1, un carattere di sottolineatura e il nome della proprietà. Ad esempio, la proprietà opzionale `CorsConfiguration` di a `CfnBucket` richiede un wrapper di tipo. `CfnBucket_CorsConfigurationProperty` Qui stiamo definendo un'`CorsConfiguration``CfnBucket`istanza.

```
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

Non è possibile utilizzare i tipi di proprietà L2 con i costrutti L1 o viceversa. Quando lavorate con i costrutti L1, utilizzate sempre i tipi definiti per il costrutto L1 che state utilizzando. Non utilizzate tipi di altri costrutti L1 (alcuni possono avere lo stesso nome, ma non sono dello stesso tipo).
Alcuni dei nostri riferimenti API specifici della lingua attualmente presentano errori nei percorsi dei tipi di proprietà L1 o non documentano affatto queste classi. Speriamo di risolvere il problema al più presto. Nel frattempo, ricorda che questi tipi sono sempre classi interne del costrutto L1 con cui vengono utilizzati.

Lavorare con costrutti L2

Nell'esempio seguente, definiamo un bucket Amazon S3 creando un oggetto dal costrutto L2: `Bucket`

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`non è il nome del bucket che crea. AWS CloudFormation È un identificatore logico assegnato al nuovo costrutto nel contesto dell'app CDK. Il valore PhysicalName verrà utilizzato per denominare la risorsa. AWS CloudFormation

Lavorare con costrutti di terze parti

Construct Hub è una risorsa che consente di scoprire costrutti aggiuntivi forniti da AWS terze parti e dalla comunità CDK open source.

Scrivere i propri costrutti

Oltre a utilizzare costrutti esistenti, puoi anche scrivere i tuoi costrutti e consentire a chiunque di utilizzarli nelle proprie app. Tutti i costrutti sono uguali in. AWS CDK I costrutti della AWS Construct Library vengono trattati allo stesso modo di un costrutto di una libreria di terze parti pubblicata tramiteNPM, o. Maven PyPI Anche i costrutti pubblicati nell'archivio di pacchetti interno dell'azienda vengono trattati allo stesso modo.

Per dichiarare un nuovo costrutto, create una classe che estenda la classe base Construct nel `constructs` pacchetto, quindi seguite lo schema per gli argomenti dell'inizializzatore.

L'esempio seguente mostra come dichiarare un costrutto che rappresenta un bucket Amazon S3. Il bucket S3 invia una notifica Amazon Simple Notification Service (Amazon SNS) ogni volta che qualcuno carica un file al suo interno.

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**
>
> Il nostro `NotifyingBucket` costrutto non eredita da, ma piuttosto da. `Bucket` `Construct` Utilizziamo la composizione, non l'ereditarietà, per raggruppare un bucket Amazon S3 e un argomento Amazon SNS. In generale, la composizione è preferita all'ereditarietà quando si sviluppano costrutti. AWS CDK

Il `NotifyingBucket` costruttore ha una tipica firma del costrutto:, e. `scope` `id` `props` L'ultimo argomento,`props`, è facoltativo (ottiene il valore predefinito`{}`) perché tutti gli oggetti di scena sono opzionali. (La `Construct` classe base non accetta `props` argomenti.) Puoi definire un'istanza di questo costrutto nella tua app senza`props`, ad esempio:

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)
```

Oppure puoi usare `props` (in Java, un parametro aggiuntivo) per specificare il prefisso del percorso su cui filtrare, ad esempio:

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/"),
})
```

In genere, dovresti anche esporre alcune proprietà o metodi sui tuoi costrutti. Non è molto utile avere un argomento nascosto dietro il costrutto, perché gli utenti del costrutto non sono in grado di sottoscriverlo. L'aggiunta di una `topic` proprietà consente ai consumatori di accedere all'argomento interno, come mostrato nell'esempio seguente:

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

Per farlo in Go, avremo bisogno di un po' di tubature extra. La nostra `NewNotifyingBucket` funzione originale ha restituito un`awss3.Bucket`. Dovremo estendere `Bucket` per includere un `topic` membro creando una `NotifyingBucket` struttura. La nostra funzione restituirà quindi questo 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
}
```

Ora i consumatori possono iscriversi all'argomento, ad esempio:

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))
```

Ulteriori informazioni

Il video seguente fornisce una panoramica completa dei costrutti CDK e spiega come utilizzarli nelle app CDK.

Spiegazione dei costrutti CDK

Ambienti

L'ambiente è l'obiettivo Account AWS e in Regione AWS cui vengono distribuiti gli stack. Tutti gli stack nell'app CDK sono associati esplicitamente o implicitamente a un ambiente (). `env`

Argomenti

- Configurazione degli ambienti
- Ambienti di bootstrap

Configurazione degli ambienti

Per gli stack di produzione, ti consigliamo di specificare esplicitamente l'ambiente per ogni stack dell'app utilizzando la proprietà. `env` L'esempio seguente specifica ambienti diversi per i suoi due diversi stack.

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 si codifica l'account e la regione di destinazione come mostrato nell'esempio precedente, lo stack viene sempre distribuito su quell'account e sulla regione specifici. Per rendere lo stack distribuibile su una destinazione diversa, ma per determinare la destinazione al momento della sintesi, lo stack può utilizzare due variabili di ambiente fornite dalla AWS CDK CLI: e. `CDK_DEFAULT_ACCOUNT` `CDK_DEFAULT_REGION` Queste variabili vengono impostate in base al AWS profilo specificato utilizzando l'`--profile`opzione o al AWS profilo predefinito se non ne specifichi uno.

Il seguente frammento di codice mostra come accedere all'account e alla regione passati dalla AWS CDK CLI nello stack.

TypeScript

Accedi alle variabili di ambiente tramite l'oggetto di Node. `process`

> **Note**
>
> È necessario il `DefinitelyTyped` modulo da utilizzare `process` TypeScript. `cdk init`installa questo modulo per te. Tuttavia, dovresti installare questo modulo manualmente se stai lavorando con un progetto creato prima che fosse aggiunto o se non hai configurato il progetto utilizzando`cdk init`.
>
> ```
> npm install @types/node
> ```

```
new MyDevStack(app, 'dev', {
```

```
  env: {
    account: process.env.CDK_DEFAULT_ACCOUNT,
    region: process.env.CDK_DEFAULT_REGION
}});
```

JavaScript

Accedi alle variabili di ambiente tramite l'`process`oggetto di Node.

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

Python

Usa il `environ` dizionario del `os` modulo per accedere alle variabili di ambiente.

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

Java

Usa `System.getenv()` per ottenere il valore di una variabile d'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#

Si usa `System.Environment.GetEnvironmentVariable()` per ottenere il valore di una variabile di 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() });
```

Specificare l' Regione AWS utilizzo di un codice regionale. Per un elenco, consulta Endpoint regionali.

La AWS CDK distingue tra non specificare affatto la `env` proprietà e specificarla utilizzando and. `CDK_DEFAULT_ACCOUNT` `CDK_DEFAULT_REGION` Il primo implica che lo stack debba sintetizzare un modello indipendente dall'ambiente. I costrutti definiti in tale pila non possono utilizzare alcuna informazione sul loro ambiente. Ad esempio, non puoi scrivere codice `if (stack.region === 'us-east-1')` o utilizzare strutture di framework come vPC.fromLookup (Python`from_lookup`:),

che devono interrogare il tuo account. AWS Queste funzionalità non funzionano affatto finché non si specifica un ambiente esplicito; per utilizzarle, è necessario specificare. `env`

Quando si passa all'ambiente utilizzando `CDK_DEFAULT_ACCOUNT` and`CDK_DEFAULT_REGION`, lo stack verrà distribuito nell'account e nella regione determinati dalla AWS CDK CLI al momento della sintesi. Ciò consente il funzionamento del codice dipendente dall'ambiente, ma significa anche che il modello sintetizzato potrebbe essere diverso in base alla macchina, all'utente o alla sessione in cui viene sintetizzato. Questo comportamento è spesso accettabile o addirittura auspicabile durante lo sviluppo, ma probabilmente costituirebbe un anti-modello per l'uso in produzione.

Puoi impostarlo come `env` preferisci, usando qualsiasi espressione valida. Ad esempio, potreste scrivere lo stack in modo da supportare due variabili di ambiente aggiuntive per consentirvi di sovrascrivere l'account e la regione al momento della sintesi. Le chiameremo `CDK_DEPLOY_ACCOUNT` e le chiameremo `CDK_DEPLOY_REGION` qui, ma puoi chiamarle come preferisci, poiché non sono impostate da. AWS CDK Nell'ambiente dello stack seguente, vengono utilizzate variabili di ambiente alternative se impostate. Se non sono impostate, tornano all'ambiente predefinito fornito da. 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() });
```

Con l'ambiente dello stack dichiarato in questo modo, puoi scrivere un breve script o un file batch come il seguente per impostare le variabili dagli argomenti della riga di comando, quindi chiamare`cdk deploy`. Tutti gli argomenti oltre ai primi due vengono passati `cdk deploy` e possono essere utilizzati per specificare le opzioni o gli stack della riga di comando.

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
```

Salva lo script con nome`cdk-deploy-to.sh`, quindi eseguilo `chmod +x cdk-deploy-to.sh` per renderlo eseguibile.

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
}
```

La versione Windows dello script fornisce PowerShell le stesse funzionalità della versione macOS/Linux. Contiene inoltre istruzioni per consentirne l'esecuzione come file batch in modo che possa essere facilmente richiamato da una riga di comando. Dovrebbe essere salvato come`cdk-deploy-to.bat`. Il file `cdk-deploy-to.ps1` verrà creato quando viene richiamato il file batch.

Quindi puoi scrivere script aggiuntivi che richiamano lo script «deploy-to» per la distribuzione in ambienti specifici (anche più ambienti per 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 %*
```

Quando esegui la distribuzione in più ambienti, valuta se desideri continuare la distribuzione in altri ambienti dopo un errore di implementazione. L'esempio seguente evita la distribuzione nel secondo ambiente di produzione se il primo non ha esito positivo.

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 %*
```

Gli sviluppatori possono comunque utilizzare il `cdk deploy` comando normale per la distribuzione nei propri AWS ambienti per lo sviluppo.

Se non si specifica un ambiente quando si crea un'istanza di uno stack, si dice che lo stack è indipendente dall'ambiente. AWS CloudFormation i modelli sintetizzati da tale stack cercheranno di utilizzare la risoluzione del tempo di distribuzione su attributi relativi all'ambiente come`stack.account`, `stack.region` e (Python:). `stack.availabilityZones` `availability_zones`

Quando si utilizza cdk deploy per distribuire stack indipendenti dall'ambiente, utilizzeranno il profilo specificato per determinare dove eseguire la distribuzione. AWS CDK CLI AWS CLI Se non viene specificato alcun profilo, viene utilizzato il profilo predefinito. AWS CDK CLISegue un protocollo simile a quello per AWS CLI determinare quali AWS credenziali utilizzare quando si eseguono operazioni AWS sul proprio account. Per informazioni dettagliate, vedi the section called “AWS CDK Kit di strumenti”.

In uno stack indipendente dall'ambiente, tutti i costrutti che utilizzano zone di disponibilità vedranno due zone di disponibilità, che consentono di distribuire lo stack in qualsiasi regione.

Ambienti di bootstrap

È necessario eseguire il bootstrap di ogni ambiente in cui verranno distribuiti gli stack CDK. Il bootstrap prepara l'ambiente per la distribuzione. Per ulteriori informazioni, consulta Bootstrapping (Processo di bootstrap).

Bootstrapping (Processo di bootstrap)

Il bootstrap è il processo di preparazione di un ambiente per la distribuzione. Il bootstrap è un'azione da eseguire una sola volta per ogni ambiente in cui vengono distribuite le risorse.

Argomenti

- Differenze tra i modelli di bootstrap
- Sintetizzatori Stack
- Personalizzazione della sintesi
- Il modello di contratto bootstrap
- Risultati del Security Hub

Ambienti di bootstrap

> **Important**
>
> Potrebbero essere AWS addebitati costi per i dati archiviati nelle risorse avviate.

Il bootstrap fornisce risorse nel tuo ambiente, come un bucket Amazon Simple Storage Service (Amazon S3) per l'archiviazione di file AWS Identity and Access Management e ruoli (IAM) che concedono le autorizzazioni necessarie per eseguire le distribuzioni. Queste risorse vengono fornite in uno stack, chiamato stack bootstrap. AWS CloudFormation Di solito è denominato. `CDKToolkit` Come ogni AWS CloudFormation stack, apparirà nella AWS CloudFormation console del tuo ambiente una volta distribuito.

> **Note**
>
> CDK v2 utilizza un modello di bootstrap moderno. Il modello legacy di CDK v1 non è supportato nella v2.

Gli ambienti sono indipendenti. Se si desidera eseguire l'implementazione in più ambienti, è necessario avviare ogni ambiente separatamente.

Se tenti di implementare un'app CDK in un ambiente che non è stato avviato, riceverai un messaggio di errore che ti ricorda di avviare l'ambiente.

Avvio con CDK Pipelines

Se utilizzi CDK Pipelines per la distribuzione nell'ambiente di un altro account e ricevi un messaggio simile al seguente:

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

Questo messaggio di errore indica che i ruoli IAM appropriati non esistono nell'altro ambiente. La causa più probabile è che l'ambiente non è stato avviato. Avvia l'ambiente e riprova.

> **Note**
>
> Se l'ambiente è avviato, non eliminare e ricreare lo stack di bootstrap dell'ambiente. L'eliminazione dello stack di bootstrap eliminerà AWS le risorse originariamente fornite nell'ambiente per supportare le distribuzioni CDK. Ciò causerà l'interruzione del funzionamento della pipeline. Prova invece ad aggiornare lo stack di bootstrap a una nuova versione eseguendo nuovamente il comando CLI `cdk bootstrap` CDK.

Come eseguire il bootstrap

Quando si avvia un ambiente, viene distribuito un AWS CloudFormation modello nell'ambiente specifico. Questo modello fornisce risorse nell'account per preparare l'ambiente alla distribuzione.

Il modello di bootstrap accetta parametri che personalizzano alcuni aspetti delle risorse avviate. Per ulteriori informazioni, consulta the section called “Personalizzazione del bootstrap”.

È possibile eseguire il bootstrap in uno dei seguenti modi:

- Utilizza il comando AWS CDK CLI.cdk bootstrap Questo è il metodo più semplice e funziona bene se avete solo pochi ambienti da avviare.
- Implementa il modello fornito AWS CDK CLI utilizzando un altro strumento di AWS CloudFormation distribuzione. Ciò consente di utilizzare AWS CloudFormation StackSets o AWS Control Tower e anche la AWS CloudFormation console o il AWS CLI. È possibile apportare piccole modifiche al modello prima della distribuzione. Questo approccio è più flessibile ed è adatto per implementazioni su larga scala.

Non è un errore avviare un ambiente più di una volta. Se un ambiente avviato è già stato avviato, il relativo stack di bootstrap verrà aggiornato se necessario. Altrimenti non succederà nulla.

Bootstrap con AWS CDKCLI

Usa il `cdk bootstrap` comando per avviare uno o più ambienti. AWS

L'esempio seguente avvia due ambienti:

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

Gli esempi seguenti mostrano diversi modi di avviare gli ambienti. Come illustrato nel secondo esempio, il `aws://` prefisso è facoltativo quando si specifica un ambiente.

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

Quando si eseguecdk bootstrap, il CDK sintetizza CLI sempre l'app CDK nella directory corrente. Se non specificate almeno un ambiente, il CDK CLI avvierà tutti gli ambienti a cui fa riferimento l'app.

Per gli stack indipendenti dall'ambiente, il CDK CLI tenterà di determinare un ambiente dalle sorgenti predefinite. Potrebbe trattarsi di un ambiente specificato utilizzando l'--profileopzione, da variabili di ambiente o da fonti predefinite. AWS CLI Se trovato, l'ambiente viene quindi avviato.

Ad esempio, il comando seguente sintetizza l' AWS CDK app corrente utilizzando il `prod` AWS profilo, quindi avvia i relativi ambienti.

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

Esecuzione dell'avvio dal modello AWS CloudFormation

È possibile avviare un ambiente ottenendo e distribuendo il modello di bootstrap. AWS CloudFormation

Per ottenere una copia di questo modello nel file`bootstrap-template.yaml`, esegui il seguente comando:

macOS/Linux

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

Windows

In Windows, PowerShell deve essere utilizzato per preservare la codifica del modello.

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

Il modello è disponibile anche nell'AWS CDK GitHub archivio.

Distribuisci questo modello utilizzando la CLI CDK o il tuo meccanismo di distribuzione preferito per i modelli. AWS CloudFormation Per eseguire la distribuzione utilizzando la CLI CDK, esegui. cdk bootstrap --template ***TEMPLATE_FILENAME*** Puoi anche distribuirlo AWS CLI utilizzando il comando seguente oppure distribuirlo su uno o più account contemporaneamente utilizzando Stack Sets. AWS CloudFormation

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
```

Personalizzazione del bootstrap

Esistono due modi per personalizzare l'avvio delle risorse nell'ambiente:

- Usa i parametri della riga di comando con il `cdk bootstrap` comando. Ciò consente di modificare alcuni aspetti del modello.
- Modifica il modello di bootstrap predefinito e distribuiscilo tu stesso. Questo ti dà un controllo più completo sulle risorse di bootstrap.

Le seguenti opzioni della riga di comando, se utilizzate con CDK CLIcdk bootstrap, forniscono le modifiche più comuni al modello di bootstrap:

- - -bootstrap-bucket-name sostituisce il nome del bucket Amazon S3. Potrebbe richiedere modifiche all'app CDK (vedi). the section called “Sintetizzatori Stack”

- --bootstrap-kms-key-idsostituisce la AWS KMS chiave utilizzata per crittografare il bucket S3.
- --cloudformation-execution-policiesspecifica gli ARN delle policy gestite da allegare al ruolo di distribuzione assunto da durante la distribuzione degli stack. AWS CloudFormation Per impostazione predefinita, gli stack vengono distribuiti con le autorizzazioni complete di amministratore utilizzando la policy. `AdministratorAccess`

  Gli ARN della policy devono essere passati come argomento a stringa singola, con i singoli ARN separati da virgole. Per esempio:

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

  > **Important**
  >
  > Per evitare errori di distribuzione, assicurati che le politiche specificate siano sufficienti per tutte le distribuzioni che eseguirai nell'ambiente in cui viene avviato il bootstrap.

- --qualifierè una stringa che viene aggiunta ai nomi di tutte le risorse nello stack di bootstrap. Un qualificatore consente di evitare conflitti tra i nomi delle risorse quando si effettua il provisioning di più stack di bootstrap nello stesso ambiente. L'impostazione predefinita è `hnb659fds` (questo valore non ha alcun significato).

  La modifica del qualificatore richiede inoltre che l'app CDK passi il valore modificato al sintetizzatore stack. Per ulteriori informazioni, consulta the section called “Sintetizzatori Stack”.
- --tagsaggiunge uno o più AWS CloudFormation tag allo stack di bootstrap.
- --trustelenca gli AWS account che possono essere distribuiti nell'ambiente di avvio.

  Usa questo flag per avviare un ambiente in cui verrà implementata una CDK Pipeline in un altro ambiente. L'account che esegue il bootstrap è sempre attendibile.
- --trust-for-lookupelenca gli AWS account che possono cercare informazioni di contesto dall'ambiente che viene avviato.

  Utilizzate questo flag per autorizzare gli account a sintetizzare gli stack che verranno distribuiti nell'ambiente, senza concedere loro effettivamente il permesso di distribuire direttamente quegli stack.
- --termination-protectionimpedisce l'eliminazione dello stack di bootstrap. Per ulteriori informazioni, consulta Proteggere uno stack dall'eliminazione nella Guida per l'AWS CloudFormation utente.

> **Important**
>
> Il moderno modello di bootstrap concede in modo efficace le autorizzazioni implicite da `--cloudformation-execution-policies` a qualsiasi AWS account nell'elenco. `--trust` Per impostazione predefinita, questo estende le autorizzazioni di lettura e scrittura a qualsiasi risorsa nell'account avviato. Assicurati di configurare lo stack di bootstrap con politiche e account affidabili con cui ti senti a tuo agio.

Personalizzazione del modello

Se hai bisogno di una personalizzazione maggiore di quella fornita dal CDK, CLI puoi modificare il modello di bootstrap in base alle tue esigenze. Innanzitutto, ottenete il modello utilizzando l'opzione. --show-template Di seguito è riportato un esempio:

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

Qualsiasi modifica apportata deve rispettare il contratto del modello di bootstrap. Per garantire che le personalizzazioni non vengano sovrascritte accidentalmente in un secondo momento da qualcuno che cdk bootstrap utilizza il modello predefinito, modifica il valore predefinito del parametro template. `BootstrapVariant` La CLI CDK consentirà di sovrascrivere lo stack di bootstrap solo con modelli che hanno la `BootstrapVariant` stessa versione e una versione uguale o superiore a quella del modello attualmente distribuito.

È quindi possibile distribuire il modello modificato come descritto in, o utilizzando. the section called “Esecuzione dell'avvio dal modello AWS CloudFormation” cdk bootstrap --template

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

Differenze tra i modelli di bootstrap

Come accennato in precedenza, la AWS CDK v1 supportava due modelli di bootstrap, legacy e moderni. CDK v2 supporta solo il modello moderno. A titolo di riferimento, ecco le differenze di alto livello tra questi due modelli.

Funzionalità	Legacy (solo v1)	Moderno (v1 e v2)
Implementazioni tra account	Non consentito	Consentito

Funzionalità	Legacy (solo v1)	Moderno (v1 e v2)
AWS CloudFormation Autorizzazioni	Esegue la distribuzione utilizzando le autorizzazioni dell'utente corrente (determinate dal AWS profilo, dalle variabili di ambiente, ecc.)	Esegue la distribuzione utilizzando le autorizzazioni specificate al momento del provisioning dello stack di bootstrap (ad esempio, utilizzando) `--trust`
Funzione Versioni multiple	È disponibile solo una versione dello stack di bootstrap	Lo stack Bootstrap ha una versione; nuove risorse possono essere aggiunte nelle versioni future e le AWS CDK app possono richiedere una versione minima
Risorse*	Bucket Amazon S3	Bucket Amazon S3
		AWS KMS key
		Ruoli IAM
		Repository Amazon ECR
		Parametro SSM per il controllo delle versioni
Denominazione delle risorse	Generato automaticamente	Deterministico
Crittografia Bucket	Chiave predefinita	Chiave gestita dal cliente

* Aggiungeremo risorse aggiuntive al modello di bootstrap secondo necessità.

Un ambiente che è stato avviato utilizzando il modello precedente deve essere aggiornato per utilizzare il modello moderno per CDK v2 mediante riavvio. Ridistribuite tutte le AWS CDK applicazioni nell'ambiente almeno una volta prima di eliminare il bucket legacy.

Sintetizzatori Stack

La tua AWS CDK app deve conoscere le risorse di bootstrap a sua disposizione per sintetizzare correttamente uno stack che può essere distribuito. Lo stack synthesizer è una AWS CDK classe che controlla il modo in cui viene sintetizzato il modello dello stack. Ciò include il modo in cui utilizza le risorse di bootstrap (ad esempio, il modo in cui si riferisce alle risorse archiviate nel bucket bootstrap).

Si chiamano AWS CDK sintetizzatori stack integrati. `DefaultStackSynthesizer` Include funzionalità per le implementazioni tra account e le implementazioni CDK Pipelines.

È possibile passare un sintetizzatore di stack a uno stack quando lo si istanzia utilizzando la proprietà. `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 non si fornisce la proprietà, viene utilizzata. `synthesizer` `DefaultStackSynthesizer`

Personalizzazione della sintesi

A seconda delle modifiche apportate al modello di bootstrap, potrebbe essere necessario personalizzare anche la sintesi. `DefaultStackSynthesizer`Possono essere personalizzati utilizzando le proprietà descritte di seguito.

Se nessuna di queste proprietà fornisce le personalizzazioni richieste, potete scrivere il sintetizzatore come una classe che implementa `IStackSynthesizer` (magari derivando da). `DefaultStackSynthesizer`

Modifica del qualificatore

Il qualificatore viene aggiunto al nome delle risorse di bootstrap per distinguere le risorse in stack di bootstrap separati. Per distribuire due versioni diverse dello stack di bootstrap nello stesso ambiente (AWS account e regione), gli stack devono avere qualificatori diversi.

Questa funzionalità è destinata all'isolamento dei nomi tra i test automatici del CDK stesso. A meno che non sia possibile definire in modo molto preciso le autorizzazioni IAM assegnate al ruolo di AWS CloudFormation esecuzione, non vi sono vantaggi in termini di isolamento delle autorizzazioni

nell'avere due diversi stack di bootstrap in un unico account. Pertanto, in genere non è necessario modificare questo valore.

Per cambiare il qualificatore, configuralo `DefaultStackSynthesizer` entrambi creando un'istanza del sintetizzatore con la proprietà:

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"
    })
});
```

Oppure configurando il qualificatore come chiave di contesto. `cdk.json`

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

Modifica dei nomi delle risorse

Tutte le altre `DefaultStackSynthesizer` proprietà si riferiscono ai nomi delle risorse nel modello di bootstrap. È necessario fornire una di queste proprietà solo se si è modificato il modello di bootstrap e si sono modificati i nomi delle risorse o lo schema di denominazione.

Tutte le proprietà accettano i segnaposto speciali`${Qualifier}`,, e`${AWS::Partition}`. `${AWS::AccountId}` `${AWS::Region}` Questi segnaposto vengono sostituiti rispettivamente dai valori del `qualifier` parametro e dai valori di AWS partizione, ID account e regione per l'ambiente dello stack.

L'esempio seguente mostra le proprietà più comunemente utilizzate `DefaultStackSynthesizer` insieme ai relativi valori predefiniti, come se si stesse istanziando il sintetizzatore. Per un elenco completo, consulta 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,
})
```

Il modello di contratto bootstrap

I requisiti dello stack di bootstrap dipendono dal sintetizzatore di stack in uso. Se scrivi il tuo sintetizzatore stack, hai il controllo completo delle risorse di bootstrap richieste dal sintetizzatore e del modo in cui il sintetizzatore le trova.

Questa sezione descrive le aspettative che ha nei confronti del modello di bootstrap. `DefaultStackSynthesizer`

Controllo delle versioni

Il modello deve contenere una risorsa per creare un parametro SSM con un nome noto e un output che rifletta la versione del modello.

```
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]
```

Roles

`DefaultStackSynthesizer`Richiede cinque ruoli IAM per cinque scopi diversi. Se non si utilizzano i ruoli predefiniti, è necessario comunicare al sintetizzatore gli ARN per i ruoli che si desidera utilizzare.

I ruoli sono i seguenti:

- Il ruolo di distribuzione viene assunto dal AWS CDK Toolkit e dalla AWS CodePipeline distribuzione in un ambiente. `AssumeRolePolicy`Controlla chi può effettuare l'implementazione nell'ambiente. Nel modello, puoi vedere le autorizzazioni necessarie per questo ruolo.
- Il ruolo di ricerca viene assunto dal AWS CDK Toolkit per eseguire ricerche contestuali in un ambiente. `AssumeRolePolicy`Controlla chi può eseguire l'implementazione nell'ambiente. Le autorizzazioni necessarie per questo ruolo possono essere visualizzate nel modello.

- Il ruolo di pubblicazione dei file e il ruolo di pubblicazione delle immagini sono assunti dal AWS CDK Toolkit e dai AWS CodeBuild progetti di pubblicazione delle risorse in un ambiente. Vengono utilizzati rispettivamente per scrivere nel bucket S3 e nel repository ECR. Questi ruoli richiedono l'accesso in scrittura a queste risorse.
- Il ruolo di AWS CloudFormation esecuzione viene passato AWS CloudFormation a eseguire la distribuzione effettiva. Le relative autorizzazioni sono le autorizzazioni in base alle quali verrà eseguita la distribuzione. Le autorizzazioni vengono passate allo stack come parametro che elenca gli ARN delle policy gestite.

Output

Il AWS CDK Toolkit richiede che nello stack di bootstrap siano presenti i seguenti CloudFormation output.

- `BucketName`: il nome del file asset bucket
- `BucketDomainName`: il bucket di risorse di file nel formato del nome di dominio
- `BootstrapVersion`: la versione corrente dello stack bootstrap

Cronologia dei modelli

Il modello bootstrap ha una versione e si evolve nel tempo con se stesso. AWS CDK Se fornisci il tuo modello di bootstrap, tienilo aggiornato con il modello canonico predefinito. Vuoi assicurarti che il tuo modello continui a funzionare con tutte le funzionalità di CDK.

> **Note**
>
> Per impostazione predefinita, le versioni precedenti del modello di bootstrap ne AWS KMS key creavano uno in ogni ambiente bootstrap. Per evitare addebiti per la chiave KMS, riavvia questi ambienti utilizzando. `--no-bootstrap-customer-key` L'impostazione predefinita attuale non prevede alcuna chiave KMS, il che aiuta a evitare questi addebiti.

Questa sezione contiene un elenco delle modifiche apportate in ciascuna versione.

Versione del modello	AWS CDK versione	Modifiche
1	1.40.0	Versione iniziale del modello con Bucket, Key, Repository e Roles.
2	1.45.0	Dividi il ruolo di pubblicazione delle risorse in ruoli separati per la pubblicazione di file e immagini.
3	1.46.0	Aggiungi `FileAsset KeyArn` l'esportazione per poter aggiungere autorizza zioni di decrittografia ai consumatori di risorse.
4	1.61.0	AWS KMS le autorizzazioni sono ora implicite tramite Amazon S3 e non sono più necessarie. `FileAsetK eyArn` Aggiungi il parametro `CdkBootstrapVersion` SSM in modo che la versione dello stack di bootstrap possa essere verificata senza conoscere il nome dello stack.
5	1.87.0	Il ruolo di distribuzione può leggere il parametro SSM.
6	1.108.0	Aggiungi il ruolo di ricerca separato dal ruolo di distribuz ione.
6	1.109.0	Allega `aws-cdk:b ootstrap-role` tag ai ruoli di distribuzione, pubblicaz

Versione del modello	AWS CDK versione	Modifiche
		ione di file e pubblicazione di immagini.
7	1.110.0	Il ruolo di distribuzione non può più leggere direttame nte i Bucket nell'account di destinazione. (Tuttavia, questo ruolo è effettivamente un amministratore e può sempre utilizzare le sue AWS CloudFormation autorizza zioni per rendere comunque leggibile il bucket).
8	1.114.0	Il ruolo di ricerca dispone di autorizzazioni complete di sola lettura per l'ambient e di destinazione e dispone anche di un tag. `aws-cdk:b ootstrap-role`
9	2.1.0	Risolve il rifiuto dei caricamen ti di risorse di Amazon S3 da parte della crittografia di riferimento comune SCP.
10	2.4.0	Amazon ECR ScanOnPush è ora abilitato per impostazione predefinita.
11	2.18.0	Aggiunge una politica che consente a Lambda di estrarre dai repository Amazon ECR in modo che sopravviva al riavvio.

Versione del modello	AWS CDK versione	Modifiche
12	2.20.0	Aggiunge il supporto per la sperimentazionecdk import.
13	2.25.0	Rende immutabili le immagini dei contenitori nei repositor y Amazon ECR creati con bootstrap.
14	2.34.0	Per impostazione predefini ta, disattiva la scansione delle immagini di Amazon ECR a livello di repository per consentire l'avvio delle regioni che non supportano la scansione delle immagini.
15	2,60,0	Le chiavi KMS non possono essere etichettate.
16	2,69,0	Indirizza Security Hub trovando KMS.2.
17	2,72,0	Indirizza Security Hub trovando ECR.3.
18	2,80,0	Sono state annullate le modifiche apportate alla versione 16 in quanto non funzionano in tutte le partizioni e non sono consigliate.

Versione del modello	AWS CDK versione	Modifiche
19.	2106,1	Sono state annullate le modifiche apportate alla versione 18 in cui la AccessControl proprietà è stata rimossa dal modello. (#27964)
20	2,119,0	Aggiungi `ssm:GetParameters` un'azione al ruolo AWS CloudFormation Deploy IAM. Per ulteriori informazioni, consulta #28336.

Risultati del Security Hub

Se lo utilizzi AWS Security Hub, potresti vedere i risultati riportati su alcune delle risorse create dal processo di AWS CDK Bootstrapping. I risultati del Security Hub ti aiutano a trovare le configurazioni delle risorse che dovresti ricontrollare per verificarne l'accuratezza e la sicurezza. Abbiamo esaminato queste configurazioni specifiche delle risorse con AWS Security e siamo certi che non costituiscano un problema di sicurezza.

[KMS.2] I responsabili IAM non dovrebbero disporre di policy IAM in linea che consentano azioni di decrittografia su tutte le chiavi KMS

Il Deploy Role (nome predefinito`cdk-hnb659fds-deploy-role-ACCOUNT-REGION`) dispone delle autorizzazioni per leggere i dati crittografati archiviati in Amazon S3. La policy di per sé non autorizza alcun dato: solo i dati letti da Amazon S3 possono essere decrittografati e solo i bucket che consentono esplicitamente al Deploy Role di leggerli tramite la relativa Bucket Policy e le chiavi che consentono esplicitamente al Deploy Role di decrittografarli utilizzando la propria Key Policy. Questa dichiarazione viene utilizzata per consentire a Pipelines di eseguire distribuzioni tra account. AWS CDK

Perché Security Hub lo segnala? La policy contiene una `Condition` clausola `Resource: *` combinata con una; Security Hub contrassegna il. `* *`È necessario perché al momento dell'avvio dell'account, la AWS KMS chiave creata da AWS CDK Pipelines per CodePipeline Artifact Bucket

non esiste ancora, quindi non possiamo fare riferimento al suo ARN. Inoltre, Security Hub non include la `Condition` clausola della dichiarazione politica nel suo ragionamento.

Cosa succede se voglio correggere questo risultato? Finché le politiche relative alle risorse sulle AWS KMS chiavi non sono inutilmente permissive, l'attuale politica relativa ai ruoli non consente a Deploy Role di accedere a più dati di quanto dovrebbe. Se vuoi comunque eliminare il risultato, puoi farlo personalizzando lo stack di bootstrap (usando la procedura descritta sopra) in uno di questi 2 modi:

- Se non utilizzi AWS CDK Pipelines per distribuzioni tra account: rimuovi l'istruzione with dal ruolo deploy; oppure `Sid: PipelineCrossAccountArtifactsBucket`
- Se utilizzi AWS CDK Pipelines per distribuzioni tra account: dopo aver distribuito la tua AWS CDK Pipeline, cerca il Key AWS KMS ARN dell'Artifact Bucket e sostituisci l'istruzione con l'ARN chiave effettivo. `Resource: *` `Sid: PipelineCrossAccountArtifactsBucket`

Risorse

Le risorse sono ciò che configuri per l'uso Servizi AWS nelle tue applicazioni. Le risorse sono una caratteristica di AWS CloudFormation. Configurando le risorse e le relative proprietà in un AWS CloudFormation modello, è possibile implementarlo per AWS CloudFormation eseguire il provisioning delle risorse. Con AWS Cloud Development Kit (AWS CDK), è possibile configurare le risorse tramite costrutti. Quindi distribuisci l'app CDK, che prevede la sintesi di un AWS CloudFormation modello e la sua implementazione per fornire le tue risorse. AWS CloudFormation

Argomenti

Configurazione delle risorse mediante costrutti

Come descritto inthe section called “Costrutti”, AWS CDK fornisce una ricca libreria di classi di costrutti, chiamati AWS costrutti, che rappresentano tutte le risorse. AWS

Per creare un'istanza di una risorsa utilizzando il costrutto corrispondente, inserite l'ambito come primo argomento, l'ID logico del costrutto e un insieme di proprietà di configurazione (props). Ad esempio, ecco come creare una coda Amazon SQS con AWS KMS crittografia utilizzando il costrutto SQS.Queue della 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
});
```

Alcune proprietà di configurazione sono opzionali e in molti casi hanno valori predefiniti. In alcuni casi, tutti gli oggetti di scena sono opzionali e l'ultimo argomento può essere omesso completamente.

Attributi delle risorse

La maggior parte delle risorse della AWS Construct Library espongono gli attributi, che vengono risolti al momento della distribuzione da. AWS CloudFormation Gli attributi sono esposti sotto forma di proprietà sulle classi di risorse con il nome del tipo come prefisso. L'esempio seguente mostra come ottenere l'URL di una coda Amazon SQS utilizzando la proprietà (`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
```

Vedi the section called “Gettoni” per informazioni su come AWS CDK codifica gli attributi di deploy-time come stringhe.

Risorse di riferimento

Quando si configurano le risorse, è spesso necessario fare riferimento alle proprietà di un'altra risorsa. Di seguito vengono mostrati gli esempi:

- Una risorsa Amazon Elastic Container Service (Amazon ECS) richiede un riferimento al cluster su cui viene eseguita.
- Una CloudFront distribuzione Amazon richiede un riferimento al bucket Amazon Simple Storage Service (Amazon S3) contenente il codice sorgente.

Puoi fare riferimento alle risorse in uno dei seguenti modi:

- Passando una risorsa definita nell'app CDK, nello stesso stack o in uno diverso
- Passando un oggetto proxy che fa riferimento a una risorsa definita nell' AWS account, creata da un identificatore univoco della risorsa (ad esempio un ARN)

Se la proprietà di un costrutto rappresenta un costrutto per un'altra risorsa, il suo tipo è quello del tipo di interfaccia del costrutto. Ad esempio, il costrutto Amazon ECS accetta una proprietà `cluster` di tipo. `ecs.ICluster` Un altro esempio è il costrutto di CloudFront distribuzione che accetta una proprietà `sourceBucket` (Python`source_bucket`:) di tipo. `s3.IBucket`

È possibile passare direttamente qualsiasi oggetto risorsa del tipo corretto definito nella stessa AWS CDK app. L'esempio seguente definisce un cluster Amazon ECS e quindi lo utilizza per definire un servizio 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 });
```

Riferimento a risorse in uno stack diverso

Puoi fare riferimento a risorse in uno stack diverso purché siano definite nella stessa app e si trovino nello stesso ambiente. AWS In genere viene utilizzato lo schema seguente:

- Memorizza un riferimento al costrutto come attributo dello stack che produce la risorsa. (Per ottenere un riferimento allo stack del costrutto corrente, usa.) `Stack.of(this)`

- Passa questo riferimento al costruttore dello stack che consuma la risorsa come parametro o proprietà. Lo stack di consumo lo passa quindi come proprietà a qualsiasi costrutto che ne abbia bisogno.

L'esempio seguente definisce uno stack. `stack1` Questo stack definisce un bucket Amazon S3 e memorizza un riferimento al costrutto del bucket come attributo dello stack. Quindi l'app definisce un secondo stack, `stack2` che accetta un bucket al momento dell'istanziazione. `stack2`potrebbe, ad esempio, definire una AWS Glue tabella che utilizza il bucket per l'archiviazione dei dati.

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 determina che la risorsa si trova nello stesso ambiente, ma in uno stack diverso, sintetizza automaticamente AWS CloudFormation le esportazioni nello stack di produzione e un Fn::ImportValue nello stack di consumo per trasferire tali informazioni da uno stack all'altro.

Risoluzione delle situazioni di stallo legate alle dipendenze

Il riferimento a una risorsa da uno stack in uno stack diverso crea una dipendenza tra i due stack. In questo modo si assicura che vengano distribuiti nell'ordine corretto. Dopo l'implementazione degli stack, questa dipendenza è concreta. Dopodiché, la rimozione dell'uso della risorsa condivisa dallo stack di consumo può causare un errore di distribuzione imprevisto. Ciò accade se esiste un'altra dipendenza tra i due stack che li obbliga a essere distribuiti nello stesso ordine. Può avvenire anche senza dipendenza se lo stack di produzione viene semplicemente scelto dal CDK Toolkit per essere distribuito per primo. L' AWS CloudFormation esportazione viene rimossa dallo stack di produzione perché non è più necessaria, ma la risorsa esportata viene ancora utilizzata nello stack di consumo perché il relativo aggiornamento non è ancora stato distribuito. Pertanto, l'implementazione dello stack di produzione non riesce.

Per superare questa situazione di stallo, rimuovete l'uso della risorsa condivisa dallo stack di consumo. (Ciò rimuove l'esportazione automatica dallo stack di produzione.) Successivamente, aggiungete manualmente la stessa esportazione allo stack di produzione utilizzando esattamente lo stesso ID logico dell'esportazione generata automaticamente. Rimuovi l'uso della risorsa condivisa nello stack di consumo e distribuisci entrambi gli stack. Quindi, rimuovi l'esportazione manuale (e la risorsa condivisa se non è più necessaria) e distribuisci nuovamente entrambi gli stack. Il `exportValue()` metodo dello stack è un modo conveniente per creare l'esportazione manuale per questo scopo. (Vedi l'esempio nel riferimento al metodo collegato.)

Riferimento alle risorse presenti nel tuo account AWS

Supponiamo che tu voglia utilizzare una risorsa già disponibile nel tuo AWS account nella tua AWS CDK app. Potrebbe trattarsi di una risorsa definita tramite la console, un AWS SDK, direttamente con AWS CloudFormation o in un'altra AWS CDK applicazione. È possibile trasformare l'ARN della risorsa (o un altro attributo identificativo o gruppo di attributi) in un oggetto proxy. L'oggetto proxy funge da riferimento alla risorsa chiamando un metodo factory statico sulla classe della risorsa.

Quando crei un proxy di questo tipo, la risorsa esterna non diventa parte dell' AWS CDK app. Pertanto, le modifiche apportate al proxy nell' AWS CDK app non influiscono sulla risorsa distribuita. Il proxy può, tuttavia, essere passato a qualsiasi AWS CDK metodo che richieda una risorsa di quel tipo.

L'esempio seguente mostra come fare riferimento a un bucket basato su un bucket esistente con l'ARN arn:aws:s3::: e a un Amazon my-bucket-name Virtual Private Cloud basato su un VPC esistente con un ID specifico.

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"
});
```

Diamo un'occhiata più da vicino al metodo. `Vpc.fromLookup()` Poiché il `ec2.Vpc` costrutto è complesso, ci sono molti modi in cui potresti voler selezionare il VPC da utilizzare con la tua app CDK. Per risolvere questo problema, il costrutto VPC ha un metodo `fromLookup` statico (Python:`from_lookup`) che consente di cercare il VPC Amazon desiderato interrogando il proprio account al momento della sintesi. AWS

Per poter essere utilizzato`Vpc.fromLookup()`, il sistema che sintetizza lo stack deve avere accesso all'account proprietario di Amazon VPC. Questo perché CDK Toolkit interroga l'account per trovare l'Amazon VPC giusto al momento della sintesi.

Inoltre, **`Vpc.fromLookup()`** funziona solo in pile definite con un account e una regione espliciti (vedi). the section called “Ambienti” Se AWS CDK tenta di cercare un Amazon VPC da uno stack indipendente dall'ambiente, CDK Toolkit non sa in quale ambiente interrogare per trovare il VPC.

Devi fornire `Vpc.fromLookup()` attributi sufficienti per identificare in modo univoco un VPC nel AWS tuo account. Ad esempio, può esserci sempre un solo VPC predefinito, quindi è sufficiente specificare il VPC come predefinito.

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 });
```

È inoltre possibile utilizzare la `tags` proprietà per ricercare i VPC tramite tag. Puoi aggiungere tag ad Amazon VPC al momento della sua creazione utilizzando AWS CloudFormation o il. AWS CDK Puoi modificare i tag in qualsiasi momento dopo la creazione utilizzando il AWS Management Console AWS CLI, il o un AWS SDK. Oltre ai tag che aggiungi tu stesso, aggiunge AWS CDK automaticamente i seguenti tag a tutti i VPC che crea.

- Nome: il nome del VPC.
- aws-cdk:subnet-name — Il nome della sottorete.
- aws-cdk:subnet-type — Il tipo di sottorete: pubblica, privata o isolata.

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" });
```

I risultati di vengono memorizzati nella cache del file del progetto. `Vpc.fromLookup()` `cdk.context.json` Consultare the section called “Context”. Affida questo file al controllo della versione in modo che la tua app continui a fare riferimento allo stesso Amazon VPC. Funziona anche se successivamente modifichi gli attributi dei tuoi VPC in modo da selezionare un VPC diverso. Ciò è particolarmente importante se si implementa lo stack in un ambiente che non ha accesso all' AWS account che definisce il VPC, come CDK Pipelines.

Sebbene sia possibile utilizzare una risorsa esterna ovunque si utilizzi una risorsa simile definita nella propria AWS CDK app, non è possibile modificarla. Ad esempio, chiamare `addToResourcePolicy` (Python:`add_to_resource_policy`) su un dispositivo esterno `s3.Bucket` non fa nulla.

Nomi fisici delle risorse

I nomi logici delle risorse in AWS CloudFormation sono diversi dai nomi delle risorse visualizzati AWS Management Console dopo la loro distribuzione da AWS CloudFormation. Le AWS CDK chiamate questi nomi finali sono nomi fisici.

Ad esempio, AWS CloudFormation potrebbe creare il bucket Amazon S3 con l'ID `Stack2MyBucket4DD88B4F` logico dell'esempio precedente con il nome fisico. `stack2mybucket4dd88b4f-iuv1rbv9z3to`

È possibile specificare un nome fisico durante la creazione di costrutti che rappresentano risorse utilizzando la proprietà <resourceType>Name. L'esempio seguente crea un bucket Amazon S3 con il nome fisico. `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" });
```

L'assegnazione di nomi fisici alle risorse presenta alcuni svantaggi in. AWS CloudFormation Soprattutto, qualsiasi modifica alle risorse distribuite che richieda la sostituzione di una risorsa, ad esempio le modifiche alle proprietà di una risorsa che sono immutabili dopo la creazione, avrà esito negativo se a una risorsa è assegnato un nome fisico. Se ti ritrovi in quello stato, l'unica soluzione è eliminare lo AWS CloudFormation stack, quindi distribuire nuovamente l'app. AWS CDK Consulta la AWS CloudFormation documentazione per i dettagli.

In alcuni casi, ad esempio quando si crea un' AWS CDK app con riferimenti interambientali, sono necessari nomi fisici AWS CDK per il corretto funzionamento. In questi casi, se non vuoi preoccuparti di inventarti un nome fisico, puoi lasciare che sia lui a AWS CDK chiamarlo per te. A tale scopo, utilizzate il valore specialePhysicalName.GENERATE_IF_NEEDED, come segue.

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 });
```

Passaggio di identificatori di risorse univoci

Quando possibile, è necessario passare le risorse per riferimento, come descritto nella sezione precedente. Tuttavia, in alcuni casi non avete altra scelta se non quella di fare riferimento a una risorsa tramite uno dei suoi attributi. I casi d'uso di esempio includono quanto segue:

- Quando si utilizzano AWS CloudFormation risorse di basso livello.
- Quando è necessario esporre risorse ai componenti di runtime di un' AWS CDK applicazione, ad esempio quando si fa riferimento alle funzioni Lambda tramite variabili di ambiente.

Questi identificatori sono disponibili come attributi sulle risorse, come i seguenti.

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

L' AWS CDK associazione Java utilizza metodi getter per gli attributi.

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

C#

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

L'esempio seguente mostra come passare il nome di un bucket generato a una AWS Lambda funzione.

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
    }
});
```

Concessione di autorizzazioni tra risorse

I costrutti di livello superiore rendono possibili le autorizzazioni con privilegi minimi offrendo API semplici e basate sugli intenti per esprimere i requisiti di autorizzazione. Ad esempio, molti costrutti L2 offrono metodi di concessione che è possibile utilizzare per concedere a un'entità (come un ruolo o un utente IAM) l'autorizzazione a lavorare con la risorsa, senza dover creare manualmente istruzioni di autorizzazione IAM.

L'esempio seguente crea le autorizzazioni per consentire al ruolo di esecuzione di una funzione Lambda di leggere e scrivere oggetti in un particolare bucket Amazon S3. Se il bucket Amazon S3 è crittografato con una AWS KMS chiave, questo metodo concede anche le autorizzazioni al ruolo di esecuzione della funzione Lambda per la decrittografia con la chiave.

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)
{
    // ...
}
```

I metodi di concessione restituiscono un oggetto. `iam.Grant` Utilizzate l'`success`attributo dell'`Grant`oggetto per determinare se la concessione è stata effettivamente applicata (ad esempio, potrebbe non essere stata applicata a risorse esterne). Puoi anche usare il metodo `assertSuccess` (Python:`assert_success`) dell'`Grant`oggetto per far sì che la concessione sia stata applicata con successo.

Se un metodo di concessione specifico non è disponibile per il particolare caso d'uso, è possibile utilizzare un metodo di concessione generico per definire una nuova concessione con un elenco di azioni specificato.

L'esempio seguente mostra come concedere a una funzione Lambda l'accesso all'azione 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");
```

Molte risorse, come le funzioni Lambda, richiedono l'assunzione di un ruolo durante l'esecuzione del codice. Una proprietà di configurazione consente di specificare un`iam.IRole`. Se non viene specificato alcun ruolo, la funzione crea automaticamente un ruolo specifico per questo uso. È quindi possibile utilizzare i metodi di concessione sulle risorse per aggiungere istruzioni al ruolo.

I metodi di concessione sono creati utilizzando API di livello inferiore per la gestione con le policy IAM. Le politiche sono modellate come oggetti. PolicyDocument Aggiungi istruzioni direttamente ai ruoli (o al ruolo associato a un costrutto) usando il `addToRolePolicy` metodo (Python`add_to_role_policy`:) o alla politica di una risorsa (come `Bucket` una politica) usando il metodo `addToResourcePolicy` (`add_to_resource_policy`Python:).

Metriche e allarmi relativi alle risorse

Molte risorse emettono CloudWatch metriche che possono essere utilizzate per configurare dashboard di monitoraggio e allarmi. I costrutti di livello superiore dispongono di metodi metrici che consentono di accedere alle metriche senza cercare il nome corretto da utilizzare.

L'esempio seguente mostra come definire un allarme quando il numero `ApproximateNumberOfMessagesNotVisible` di una coda Amazon SQS supera 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 non esiste un metodo per una particolare metrica, puoi utilizzare il metodo metrico generale per specificare il nome della metrica manualmente.

Le metriche possono anche essere aggiunte ai dashboard. CloudWatch Per informazioni, consulta CloudWatch.

Traffico di rete

In molti casi, è necessario abilitare le autorizzazioni su una rete affinché un'applicazione funzioni, ad esempio quando l'infrastruttura di elaborazione deve accedere al livello di persistenza. Le risorse che stabiliscono o ascoltano le connessioni espongono metodi che abilitano i flussi di traffico, inclusa l'impostazione di regole per i gruppi di sicurezza o ACL di rete.

Le risorse iConnectable hanno una `connections` proprietà che funge da gateway per la configurazione delle regole del traffico di rete.

È possibile abilitare il flusso dei dati su un determinato percorso di rete utilizzando metodi. `allow` L'esempio seguente abilita le connessioni HTTPS al Web e le connessioni in entrata dal gruppo 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());
```

Ad alcune risorse sono associate porte predefinite. Gli esempi includono il listener di un load balancer sulla porta pubblica e le porte su cui il motore di database accetta connessioni per le istanze di un database Amazon RDS. In questi casi, è possibile applicare un controllo rigoroso della rete senza dover specificare manualmente la porta. Per farlo, usa i `allowToDefaultPort` metodi `allowDefaultPortFrom` and (Python:`allow_default_port_from`,`allow_to_default_port`).

L'esempio seguente mostra come abilitare le connessioni da qualsiasi indirizzo IPV4 e una connessione da un gruppo Auto Scaling per accedere a un database.

TypeScript

```
listener.connections.allowDefaultPortFromAnyIpv4('Allow public access');

fleet.connections.allowToDefaultPort(rdsDatabase, 'Fleet can access database');
```

JavaScript

```
listener.connections.allowDefaultPortFromAnyIpv4('Allow public access');
```

```
fleet.connections.allowToDefaultPort(rdsDatabase, 'Fleet can access database');
```

Python

```
listener.connections.allow_default_port_from_any_ipv4("Allow public access")

fleet.connections.allow_to_default_port(rds_database, "Fleet can access database")
```

Java

```
listener.getConnections().allowDefaultPortFromAnyIpv4("Allow public access");

fleet.getConnections().AllowToDefaultPort(rdsDatabase, "Fleet can access database");
```

C#

```
listener.Connections.AllowDefaultPortFromAnyIpv4("Allow public access");

fleet.Connections.AllowToDefaultPort(rdsDatabase, "Fleet can access database");
```

Gestione degli eventi

Alcune risorse possono fungere da fonti di eventi. Usa il `addEventNotification` metodo (Python:`add_event_notification`) per registrare un obiettivo di evento su un particolare tipo di evento emesso dalla risorsa. Inoltre, i `addXxxNotification` metodi offrono un modo semplice per registrare un gestore per i tipi di eventi più comuni.

L'esempio seguente mostra come attivare una funzione Lambda quando un oggetto viene aggiunto a un bucket Amazon S3.

TypeScript

```
import * as s3nots from '@aws-cdk/aws-s3-notifications';

const handler = new lambda.Function(this, 'Handler', { /*…*/ });
const bucket = new s3.Bucket(this, 'Bucket');
bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler));
```

JavaScript

```
const s3nots = require('@aws-cdk/aws-s3-notifications');

const handler = new lambda.Function(this, 'Handler', { /*…*/ });
const bucket = new s3.Bucket(this, 'Bucket');
bucket.addObjectCreatedNotification(new s3nots.LambdaDestination(handler));
```

Python

```
import aws_cdk.aws_s3_notifications as s3_nots

handler = lambda_.Function(self, "Handler", ...)
bucket = s3.Bucket(self, "Bucket")
bucket.add_object_created_notification(s3_nots.LambdaDestination(handler))
```

Java

```
import software.amazon.awscdk.services.s3.Bucket;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.s3.notifications.LambdaDestination;

Function handler = Function.Builder.create(this, "Handler")/* ... */.build();
Bucket bucket = new Bucket(this, "Bucket");
bucket.addObjectCreatedNotification(new LambdaDestination(handler));
```

C#

```
using lambda = Amazon.CDK.AWS.Lambda;
using s3 = Amazon.CDK.AWS.S3;
using s3Nots = Amazon.CDK.AWS.S3.Notifications;

var handler = new lambda.Function(this, "Handler", new lambda.FunctionProps { .. });
var bucket = new s3.Bucket(this, "Bucket");
bucket.AddObjectCreatedNotification(new s3Nots.LambdaDestination(handler));
```

Politiche di rimozione

Le risorse che mantengono dati persistenti, come database, bucket Amazon S3 e registri Amazon ECR, hanno una politica di rimozione. La politica di rimozione indica se eliminare gli oggetti

persistenti quando lo AWS CDK stack che li contiene viene distrutto. I valori che specificano la politica di rimozione sono disponibili tramite l'`RemovalPolicy`enumerazione nel modulo. AWS CDK `core`

> **Note**
>
> Le risorse oltre a quelle che archiviano i dati in modo persistente potrebbero anche avere una `removalPolicy` che viene utilizzata per uno scopo diverso. Ad esempio, una versione della funzione Lambda utilizza un `removalPolicy` attributo per determinare se una determinata versione viene mantenuta quando viene distribuita una nuova versione. Questi hanno significati e impostazioni predefinite diversi rispetto alla politica di rimozione su un bucket Amazon S3 o una tabella DynamoDB.

Valore	significato
`RemovalPolicy.CONSERVARE`	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.DISTRUGGERE`	The resource will be destroyed along with the stack.

AWS CloudFormation non rimuove i bucket Amazon S3 che contengono file anche se la politica di rimozione è impostata su. `DESTROY` Tentare di farlo è un errore. AWS CloudFormation Per fare in modo che AWS CDK elimini tutti i file dal bucket prima di distruggerlo, imposta la proprietà del bucket su. `autoDeleteObjects` `true`

Di seguito è riportato un esempio di creazione di un bucket Amazon S3 con `RemovalPolicy` of `DESTROY` e `autoDeleteOjbects` impostato su. `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
    });
}
```

Puoi anche applicare una politica di rimozione direttamente alla AWS CloudFormation risorsa sottostante tramite il `applyRemovalPolicy()` metodo. Questo metodo è disponibile su alcune risorse stateful che non hanno una `removalPolicy` proprietà negli oggetti di scena della risorsa L2. Considerare i seguenti esempi:

- AWS CloudFormation pile
- Pool di utenti Amazon Cognito

- Istanze di database Amazon DocumentDB
- Volumi Amazon EC2
- Domini Amazon OpenSearch Service
- File system Amazon FSx
- Code 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**
>
> La «s AWS CDK» si `RemovalPolicy` traduce in «s». AWS CloudFormation`DeletionPolicy` Tuttavia, l'impostazione predefinita AWS CDK è quella di conservare i dati, che è l'opposto dell' AWS CloudFormation impostazione predefinita.

Identificatori

Durante la creazione di AWS Cloud Development Kit (AWS CDK) app, utilizzerai molti tipi di identificatori e nomi. Per utilizzarli AWS CDK in modo efficace ed evitare errori, è importante comprendere i tipi di identificatori.

Gli identificatori devono essere univoci nell'ambito in cui vengono creati; non è necessario che siano univoci a livello globale nell'applicazione AWS CDK .

Se si tenta di creare un identificatore con lo stesso valore all'interno dello stesso ambito, viene generata un' AWS CDK eccezione.

Argomenti

Costruisci ID

L'identificatore più comune,`id`, è l'identificatore passato come secondo argomento durante l'istanziazione di un oggetto di costruzione. Questo identificatore, come tutti gli identificatori, deve essere univoco solo nell'ambito in cui viene creato, che è il primo argomento quando si crea un'istanza di un oggetto di costruzione.

> **Note**
>
> L'`id`of a stack è anche l'identificatore utilizzato per fare riferimento ad esso in. the section called “AWS CDK Kit di strumenti”

Diamo un'occhiata a un esempio in cui abbiamo due costrutti con l'identificatore `MyBucket` nella nostra app. Il primo è definito nell'ambito dello stack con l'identificatore. `Stack1` Il secondo è definito nell'ambito di uno stack con l'identificatore. `Stack2` Poiché sono definiti in ambiti diversi, ciò non causa alcun conflitto e possono coesistere nella stessa app senza problemi.

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");
    }
}
```

Percorsi

I costrutti di un' AWS CDK applicazione formano una gerarchia radicata nella classe. App Ci riferiamo alla raccolta di ID di un determinato costrutto, al suo costrutto genitore, al suo predecessore e così via alla radice dell'albero dei costrutti, come percorso.

AWS CDK In genere visualizza i percorsi nei modelli come una stringa. Gli ID dei livelli sono separati da barre, a partire dal nodo immediatamente sotto l'Appistanza root, che di solito è una pila. Ad esempio, i percorsi delle due risorse del bucket Amazon S3 nell'esempio di codice precedente sono e. `Stack1/MyBucket` `Stack2/MyBucket`

È possibile accedere al percorso di qualsiasi costrutto a livello di codice, come illustrato nell'esempio seguente. Questo ottiene il percorso di `myConstruct` (`omy_construct`, come lo scriverebbero gli sviluppatori Python). Poiché gli ID devono essere unici nell'ambito in cui vengono creati, i loro percorsi sono sempre unici all'interno di un' AWS CDK applicazione.

TypeScript

```
const path: string = myConstruct.node.path;
```

JavaScript

```
const path = myConstruct.node.path;
```

Python

```
path = my_construct.node.path
```

Java

```
String path = myConstruct.getNode().getPath();
```

C#

```
string path = myConstruct.Node.Path;
```

ID univoci

AWS CloudFormation richiede che tutti gli ID logici in un modello siano univoci. Per questo motivo, AWS CDK devono essere in grado di generare un identificatore univoco per ogni costrutto in un'applicazione. Le risorse hanno percorsi univoci a livello globale (i nomi di tutti gli ambiti dallo stack a una risorsa specifica). Pertanto, AWS CDK genera gli identificatori univoci necessari concatenando gli elementi del percorso e aggiungendo un hash a 8 cifre. (L'hash è necessario per distinguere percorsi distinti, come `A/B/C` e`A/BC`, che risulterebbero nello stesso identificatore. AWS CloudFormation AWS CloudFormation gli identificatori sono alfanumerici e non possono contenere barre o altri caratteri separatori.) AWS CDK Chiama questa stringa l'ID univoco del costrutto.

In generale, la tua AWS CDK app non dovrebbe aver bisogno di conoscere gli ID univoci. Tuttavia, potete accedere all'ID univoco di qualsiasi costrutto a livello di codice, come illustrato nell'esempio seguente.

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);
```

L'indirizzo è un altro tipo di identificatore univoco che distingue in modo univoco le risorse CDK. Derivato dall'hash SHA-1 del percorso, non è leggibile dall'uomo. Tuttavia, la sua lunghezza costante e relativamente breve (sempre 42 caratteri esadecimali) lo rende utile in situazioni in cui l'ID univoco «tradizionale» potrebbe essere troppo lungo. Alcuni costrutti possono utilizzare l'indirizzo nel AWS CloudFormation modello sintetizzato anziché l'ID univoco. Anche in questo caso, la tua app in genere non dovrebbe aver bisogno di conoscere gli indirizzi dei suoi costrutti, ma puoi recuperare l'indirizzo di un costrutto come segue.

TypeScript

```
const addr: string = myConstruct.node.addr;
```

JavaScript

```
const addr = myConstruct.node.addr;
```

Python

```
addr = my_construct.node.addr
```

Java

```
String addr = myConstruct.getNode().getAddr();
```

C#

```
string addr = myConstruct.Node.Addr;
```

ID logici

Gli ID univoci fungono da identificatori logici (o nomi logici) delle risorse nei AWS CloudFormation modelli generati per i costrutti che rappresentano AWS le risorse.

Ad esempio, il bucket Amazon S3 dell'esempio precedente, creato all'interno, restituisce una `Stack2` risorsa. `AWS::S3::Bucket` L'ID logico della risorsa si trova `Stack2MyBucket4DD88B4F` nel modello risultante AWS CloudFormation . (Per i dettagli su come viene generato questo identificatore, vederethe section called “ID univoci”.)

Stabilità dell'ID logico

Evita di modificare l'ID logico di una risorsa dopo che è stata creata. AWS CloudFormation identifica le risorse in base al relativo ID logico. Pertanto, se si modifica l'ID logico di una risorsa, AWS CloudFormation crea una nuova risorsa con il nuovo ID logico, quindi elimina quello esistente. A seconda del tipo di risorsa, ciò potrebbe causare l'interruzione del servizio, la perdita di dati o entrambe le cose.

Gettoni

I token rappresentano valori che possono essere risolti solo in un secondo momento nel ciclo di vita dell'app. Ad esempio, il nome di un bucket Amazon Simple Storage Service (Amazon S3) definito nell'app CDK viene assegnato solo quando il modello viene sintetizzato. AWS CloudFormation Se stampi l'`bucket.bucketName`attributo, che è una stringa, vedrai che contiene qualcosa di simile al seguente:

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

Questo è il modo in cui AWS CDK codifica un token il cui valore non è ancora noto al momento della costruzione, ma sarà disponibile in seguito. AWS CDK Chiama questi segnaposto token. In questo caso, è un token codificato come stringa.

Puoi passare questa stringa come se fosse il nome del bucket. Nell'esempio seguente, il nome del bucket viene specificato come variabile di ambiente di una AWS Lambda funzione.

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 il AWS CloudFormation modello viene finalmente sintetizzato, il token viene reso come intrinseco. AWS CloudFormation `{ "Ref": "MyBucket" }` Al momento della distribuzione, AWS CloudFormation sostituisce questo elemento intrinseco con il nome effettivo del bucket che è stato creato.

Argomenti

- Token e codifiche dei token
- Token codificati in stringhe
- Token codificati in un elenco
- Token con codifica numerica
- Valori pigri
- Conversione in JSON

Token e codifiche dei token

I token sono oggetti che implementano l'interfaccia IResolvable, che contiene un unico metodo. `resolve` AWS CDK Richiama questo metodo durante la sintesi per produrre il valore finale per il modello. AWS CloudFormation I token partecipano al processo di sintesi per produrre valori arbitrari di qualsiasi tipo.

> **Note**
>
> Raramente lavorerai direttamente con l'`IResolvable`interfaccia. Molto probabilmente vedrai solo versioni di token con codifica a stringa.

Le altre funzioni in genere accettano solo argomenti di tipo base, come o. `string` `number` Per utilizzare i token in questi casi, potete codificarli in uno dei tre tipi utilizzando metodi statici sulla classe CDK.Token.

- `Token.asString`per generare una codifica di stringhe (o chiamare `.toString()` l'oggetto token)
- `Token.asList`per generare una codifica di elenco
- `Token.asNumber`per generare una codifica numerica

Questi prendono un valore arbitrario, che può essere un`IResolvable`, e li codificano in un valore primitivo del tipo indicato.

> **Important**
>
> Poiché uno qualsiasi dei tipi precedenti può essere potenzialmente un token codificato, fai attenzione quando analizzi o provi a leggerne il contenuto. Ad esempio, se tenti di analizzare una stringa per estrarne un valore e la stringa è un token codificato, l'analisi fallisce. Analogamente, se si tenta di interrogare la lunghezza di un array o di eseguire operazioni matematiche con un numero, è necessario innanzitutto verificare che non si tratti di token codificati.

Per verificare se un valore contiene un token non risolto, chiamate il metodo `Token.isUnresolved` (Python`is_unresolved`:).

L'esempio seguente verifica che un valore di stringa, che potrebbe essere un token, non sia lungo più di 10 caratteri.

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 name è un token, la convalida non viene eseguita e potrebbe comunque verificarsi un errore in una fase successiva del ciclo di vita, ad esempio durante la distribuzione.

> **Note**
>
> È possibile utilizzare le codifiche dei token per sfuggire al sistema di tipi. Ad esempio, potete codificare tramite stringa un token che produce un valore numerico al momento della sintesi. Se utilizzate queste funzioni, è vostra responsabilità assicurarvi che il modello si risolva in uno stato utilizzabile dopo la sintesi.

Token codificati in stringhe

I token con codifica in stringa hanno il seguente aspetto.

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

Possono essere passati in giro come normali stringhe e possono essere concatenati, come illustrato nell'esempio seguente.

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";
```

È inoltre possibile utilizzare l'interpolazione di stringhe, se la lingua in uso la supporta, come illustrato nell'esempio seguente.

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";
```

Evitate di manipolare la stringa in altri modi. Ad esempio, è probabile che l'acquisizione di una sottostringa di una stringa interrompa il token della stringa.

Token codificati in un elenco

I token con codifica a elenco hanno il seguente aspetto:

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

L'unica cosa sicura da fare con questi elenchi è passarli direttamente ad altri costrutti. I token in forma di elenco di stringhe non possono essere concatenati, né è possibile prelevare un elemento dal token. L'unico modo sicuro per manipolarli è utilizzare AWS CloudFormation funzioni intrinseche come fn.select.

Token con codifica numerica

I token con codifica numerica sono un insieme di minuscoli numeri negativi a virgola mobile che assomigliano ai seguenti.

```
-1.8881545897087626e+289
```

Come con i token di lista, non è possibile modificare il valore numerico, poiché così facendo si rischia di rompere il token numerico. L'unica operazione consentita è passare il valore a un altro costrutto.

Valori pigri

Oltre a rappresentare i valori del tempo di implementazione, come i AWS CloudFormation parametri, i token vengono comunemente utilizzati anche per rappresentare valori lazy in fase di sintesi. Si tratta di valori per i quali il valore finale verrà determinato prima del completamento della sintesi, ma non

nel punto in cui viene costruito il valore. Utilizzate i token per passare una stringa letterale o un valore numerico a un altro costrutto, mentre il valore effettivo al momento della sintesi potrebbe dipendere da alcuni calcoli che devono ancora essere eseguiti.

È possibile creare token che rappresentano valori lazy in fase di sintetizzazione utilizzando metodi statici sulla classe, ad esempio lazy.String e Lazy.Number. `Lazy` Questi metodi accettano un oggetto la cui `produce` proprietà è una funzione che accetta un argomento di contesto e restituisce il valore finale quando viene chiamata.

L'esempio seguente crea un gruppo Auto Scaling la cui capacità viene determinata dopo la sua creazione.

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;
```

Conversione in JSON

A volte vuoi produrre una stringa JSON di dati arbitrari e potresti non sapere se i dati contengono token. Per codificare correttamente in JSON qualsiasi struttura di dati, indipendentemente dal fatto che contenga token, usa lo stack di metodi. toJsonString, come illustrato nell'esempio seguente.

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
});
```

Parametri

I parametri sono valori personalizzati forniti al momento della distribuzione. I parametri sono una funzionalità di AWS CloudFormation. Poiché AWS Cloud Development Kit (AWS CDK) sintetizza i AWS CloudFormation modelli, offre anche supporto per i parametri relativi al tempo di implementazione.

Argomenti

- Informazioni sui parametri
- Definizione dei parametri
- Utilizzo dei parametri
- Distribuzione con parametri

Informazioni sui parametri

Utilizzando AWS CDK, è possibile definire i parametri, che possono quindi essere utilizzati nelle proprietà dei costrutti creati. È inoltre possibile distribuire pile che contengono parametri.

Quando si distribuisce il AWS CloudFormation modello utilizzando il AWS CDK Toolkit, si forniscono i valori dei parametri sulla riga di comando. Se si distribuisce il modello tramite la AWS CloudFormation console, vengono richiesti i valori dei parametri.

In generale, si sconsiglia di utilizzare AWS CloudFormation parametri con. AWS CDK I metodi usuali per passare valori nelle AWS CDK app sono i valori di contesto e le variabili di ambiente. Poiché non sono disponibili al momento della sintesi, i valori dei parametri non possono essere utilizzati facilmente per il controllo del flusso e per altri scopi nell'app CDK.

> **Note**
>
> Per controllare il flusso con i parametri, è possibile utilizzare `CfnCondition`i costrutti, sebbene ciò sia scomodo rispetto alle istruzioni native. `if`

L'utilizzo dei parametri richiede di prestare attenzione al comportamento del codice che state scrivendo al momento della distribuzione e anche in fase di sintesi. Ciò rende più difficile comprendere e ragionare sull' AWS CDK applicazione, in molti casi con scarsi vantaggi.

In genere, è meglio fare in modo che l'app CDK accetti le informazioni necessarie in un modo ben definito e le utilizzi direttamente per dichiarare i costrutti nell'app CDK. Un AWS CloudFormation modello ideale AWS CDK generato è concreto, senza altri valori da specificare al momento dell'implementazione.

Esistono, tuttavia, casi d'uso per i quali AWS CloudFormation i parametri si adattano in modo univoco. Se disponi di team separati che definiscono e implementano l'infrastruttura, ad esempio, puoi utilizzare i parametri per rendere i modelli generati più utili. Inoltre, poiché AWS CDK supporta i AWS CloudFormation parametri, è possibile utilizzarli AWS CDK con AWS servizi che utilizzano AWS CloudFormation modelli (come Service Catalog). Questi AWS servizi utilizzano parametri per configurare il modello che viene distribuito.

Definizione dei parametri

Utilizzate la `CfnParameter`classe per definire un parametro. Ti consigliamo di specificare almeno un tipo e una descrizione per la maggior parte dei parametri, sebbene entrambi siano tecnicamente opzionali. La descrizione viene visualizzata quando all'utente viene richiesto di immettere il valore del parametro nella AWS CloudFormation console. Per ulteriori informazioni sui tipi disponibili, vedere Tipi.

> **Note**
>
> È possibile definire i parametri in qualsiasi ambito. Tuttavia, consigliamo di definire i parametri a livello di stack in modo che il loro ID logico non cambi quando rifattorizzate il codice.

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"
});
```

Utilizzo dei parametri

Un'`CfnParameter`istanza espone il suo valore all' AWS CDK app tramite un token. Come tutti i token, il token del parametro viene risolto al momento della sintesi. Ma si risolve in un riferimento al parametro definito nel AWS CloudFormation modello (che verrà risolto al momento della distribuzione), piuttosto che a un valore concreto.

È possibile recuperare il token come istanza della `Token` classe o in una stringa, in un elenco di stringhe o in una codifica numerica. La scelta dipende dal tipo di valore richiesto dalla classe o dal metodo con cui si desidera utilizzare il parametro.

TypeScript

Property	kind of value
`value`	`Token` class instance
`valueAsList`	The token represented as a string list
`valueAsNumber`	The token represented as a number
`valueAsString`	The token represented as a string

JavaScript

Property	kind of value
`value`	`Token` class instance
`valueAsList`	The token represented as a string list
`valueAsNumber`	The token represented as a number

Property	kind of value
`valueAsString`	The token represented as a string

Python

Property	kind of value
`value`	`Token` class instance
`valore_come_lista`	The token represented as a string list
`valore_come_numero`	The token represented as a number
`valore_come_stringa`	The token represented as a string

Java

Property	kind of value
`recuperaValore ()`	`Token` class instance
`getValueAsElenco ()`	The token represented as a string list
`getValueAsNumero ()`	The token represented as a number
`getValueAsStringa ()`	The token represented as a string

C#

Property	kind of value
`Valore`	`Token` class instance
`ValueAsList`	The token represented as a string list
`ValueAsNumber`	The token represented as a number

Property	kind of value
`ValueAsString`	The token represented as a string

Ad esempio, per utilizzare un parametro in una `Bucket` definizione:

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
};
```

Distribuzione con parametri

Un modello generato contenente parametri può essere distribuito nel modo consueto tramite la AWS CloudFormation console. Ti vengono richiesti i valori di ogni parametro.

Il AWS CDK Toolkit (strumento da riga di `cdk` comando) supporta anche la specificazione dei parametri al momento della distribuzione. Questi vengono forniti sulla riga di comando dopo il `--parameters` flag. È possibile distribuire uno stack che utilizza il `uploadBucketName` parametro, come nell'esempio seguente.

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

Per definire più parametri, utilizzate più `--parameters` flag.

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

Se state distribuendo più stack, potete specificare un valore diverso di ogni parametro per ogni stack. A tale scopo, aggiungete al nome del parametro il nome dello stack e i due punti.

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

Per impostazione predefinita, AWS CDK conserva i valori dei parametri delle distribuzioni precedenti e li utilizza nelle distribuzioni successive se non vengono specificati in modo esplicito. Utilizzate il `--no-previous-parameters` flag per richiedere che tutti i parametri siano specificati.

Assegnazione di tag

I tag sono elementi chiave-valore informativi che puoi aggiungere ai costrutti della tua app. AWS CDK Un tag applicato a un determinato costrutto si applica anche a tutti i suoi figli taggabili. I tag sono inclusi nel AWS CloudFormation modello sintetizzato dall'app e vengono applicati alle risorse che distribuisce. AWS Puoi utilizzare i tag per identificare e classificare le risorse per i seguenti scopi:

- Semplificazione della gestione
- Allocazione dei costi
- Controllo accessi
- Qualsiasi altro scopo da te ideato

> **Tip**
>
> Per ulteriori informazioni su come utilizzare i tag con AWS le risorse, consulta le migliori pratiche per l'etichettatura AWS delle risorse nel white paper.AWS

Argomenti

Utilizzo dei tag

La `Tags`classe include il metodo statico`of()`, tramite il quale è possibile aggiungere o rimuovere tag dal costrutto specificato.

- `Tags.of(SCOPE).add()`applica un nuovo tag al costrutto dato e a tutti i suoi figli.
- `Tags.of(SCOPE).remove()`rimuove un tag dal costrutto dato e da tutti i suoi figli, compresi i tag che un costrutto figlio potrebbe aver applicato a se stesso.

> **Note**
>
> Il tagging è implementato utilizzando. the section called “Aspetti” Gli aspetti sono un modo per applicare un'operazione (come l'etichettatura) a tutti i costrutti in un determinato ambito.

L'esempio seguente applica la chiave del tag con il valore value a un costrutto.

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");
```

L'esempio seguente elimina la chiave del tag da un costrutto.

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 utilizzate `Stage` costrutti, applicate il tag al `Stage` livello o al di sotto. I tag non vengono applicati oltre `Stage` i limiti.

Priorità dei tag

AWS CDK Applica e rimuove i tag in modo ricorsivo. In caso di conflitti, vince l'operazione di tagging con la priorità più alta. (Le priorità vengono impostate utilizzando la `priority` proprietà opzionale). Se le priorità di due operazioni sono le stesse, vince l'operazione di etichettatura più vicina alla parte inferiore dell'albero di costruzione. Per impostazione predefinita, l'applicazione di un tag ha una priorità di 100 (ad eccezione dei tag aggiunti direttamente a una AWS CloudFormation risorsa, che ha una priorità di 50). La priorità predefinita per la rimozione di un tag è 200.

Quanto segue applica un tag con una priorità di 300 a un costrutto.

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 });
```

Proprietà facoltative

I tag `properties`consentono di ottimizzare il modo in cui i tag vengono applicati o rimossi dalle risorse. Tutte le proprietà sono facoltative.

`applyToLaunchedInstances`(Python:) `apply_to_launched_instances`

Disponibile solo per add (). Per impostazione predefinita, i tag vengono applicati alle istanze avviate in un gruppo Auto Scaling. Imposta questa proprietà su false per ignorare le istanze avviate in un gruppo Auto Scaling.

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

Usali per manipolare i tag solo su un sottoinsieme di risorse, in base AWS CloudFormation ai tipi di risorse. Per impostazione predefinita, l'operazione viene applicata a tutte le risorse del sottoalbero di costruzione, ma può essere modificata includendo o escludendo determinati tipi di risorse. L'esclusione ha la precedenza sull'inclusione, se vengono specificati entrambi.

`priority`

Utilizzatelo per impostare la priorità di questa operazione rispetto alle altre `Tags.remove()` operazioni `Tags.add()` and. I valori più alti hanno la precedenza sui valori più bassi. L'impostazione predefinita è 100 per le operazioni di aggiunta (50 per i tag applicati direttamente alle AWS CloudFormation risorse) e 200 per le operazioni di rimozione.

L'esempio seguente applica il tag tagname con il valore value e la priorità 100 alle risorse di tipo presente AWS::Xxx::Yyynel costrutto. Non applica il tag alle istanze avviate in un gruppo Amazon EC2 Auto Scaling o a risorse di tipo diverso. AWS::Xxx::Zzz (Si tratta di segnaposto per due tipi di risorse arbitrari ma diversi). AWS CloudFormation

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
});
```

L'esempio seguente rimuove il tag tagname con priorità 200 dalle risorse di tipo presente AWS::Xxx::Yyynel costrutto, ma non dalle risorse di 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
});
```

Esempio

L'esempio seguente aggiunge il tag key StackTypecon value TheBesta qualsiasi risorsa creata all'interno del `Stack` file denominato. `MarketingSystem` Quindi lo rimuove nuovamente da tutte

le risorse tranne le sottoreti VPC di Amazon EC2. Il risultato è che solo le sottoreti hanno il tag applicato.

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"]
});
```

Il codice seguente consente di ottenere lo stesso risultato. Considerate quale approccio (inclusione o esclusione) rende più chiaro il vostro intento.

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"]
});
```

Etichettatura di singoli costrutti

`Tags.of(scope).add(key, value)`è il modo standard per aggiungere tag ai costrutti in. AWS CDK Il suo comportamento di tree-walking, che contrassegna in modo ricorsivo tutte le risorse taggabili nell'ambito di un determinato ambito, è quasi sempre quello che si desidera. A volte, tuttavia, è necessario etichettare uno o più costrutti specifici e arbitrari.

Uno di questi casi prevede l'applicazione di tag il cui valore deriva da alcune proprietà del costrutto da etichettare. L'approccio di etichettatura standard applica in modo ricorsivo la stessa chiave e lo stesso valore a tutte le risorse corrispondenti nell'ambito. Tuttavia, qui il valore potrebbe essere diverso per ogni costrutto taggato.

I tag vengono implementati utilizzando gli aspetti e il CDK chiama il `visit()` metodo del tag per ogni costrutto nell'ambito specificato. `Tags.of(scope)` Possiamo chiamare `Tag.visit()` direttamente per applicare un tag a un singolo costrutto.

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);
```

È possibile etichettare tutti i costrutti in un ambito, ma lasciare che i valori dei tag derivino dalle proprietà di ciascun costrutto. A tale scopo, scrivete un aspetto e applicate il tag nel `visit()` metodo dell'aspetto, come mostrato nell'esempio precedente. Quindi, aggiungete l'aspetto all'ambito desiderato utilizzando`Aspects.of(scope).add(aspect)`.

L'esempio seguente applica un tag a ciascuna risorsa in uno stack contenente il percorso della risorsa.

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**
>
> La logica dell'etichettatura condizionale, che include priorità, tipi di risorse e così via, è integrata nella classe. `Tag` È possibile utilizzare queste funzionalità quando si applicano tag a risorse arbitrarie; il tag non viene applicato se le condizioni non sono soddisfatte. Inoltre, la `Tag` classe tagga solo le risorse etichettabili, quindi non è necessario verificare se un costrutto è taggabile prima di applicare un tag.

Asset

Le risorse sono file, directory o immagini Docker locali che possono essere raggruppati in librerie e app. AWS CDK Ad esempio, una risorsa potrebbe essere una directory che contiene il codice del gestore per una funzione. AWS Lambda Le risorse possono rappresentare qualsiasi elemento di cui l'app ha bisogno per funzionare.

Il seguente video tutorial fornisce una panoramica completa delle risorse CDK e spiega come utilizzarle nella propria infrastruttura come codice (IaC).

Spiegazione delle risorse CDK

Si aggiungono risorse tramite API che sono esposte da costrutti specifici AWS . Ad esempio, quando definite un costrutto lambda.Function, la proprietà code consente di passare una risorsa (directory). `Function`utilizza le risorse per raggruppare il contenuto della directory e utilizzarlo per il codice della funzione. Allo stesso modo, ecs. ContainerImage.fromAsset utilizza un'immagine Docker creata da una directory locale per definire una definizione di attività Amazon ECS.

Risorse in dettaglio

Quando fai riferimento a una risorsa nella tua app, l'assembly cloud sintetizzato dall'applicazione include informazioni sui metadati con istruzioni per la CLI. AWS CDK Le istruzioni includono dove trovare la risorsa sul disco locale e che tipo di raggruppamento eseguire in base al tipo di risorsa, ad esempio una directory da comprimere (zip) o un'immagine Docker da creare.

AWS CDK Genera un hash sorgente per le risorse. Questo può essere utilizzato in fase di costruzione per determinare se il contenuto di un asset è cambiato.

Per impostazione predefinita, AWS CDK crea una copia della risorsa nella directory di cloud assembly, che per impostazione predefinita è`cdk.out`, sotto l'hash di origine. In questo modo, l'assembly cloud è autonomo, quindi se viene spostato su un altro host per l'implementazione, può comunque essere distribuito. Per informazioni dettagliate, vedi the section called “Assemblaggi cloud”.

Quando AWS CDK distribuisce un'app che fa riferimento agli asset (direttamente tramite il codice dell'app o tramite una libreria), la AWS CDK CLI prima prepara e pubblica gli asset in un bucket Amazon S3 o in un repository Amazon ECR. (Il bucket o il repository S3 viene creato durante il bootstrap.) Solo allora vengono distribuite le risorse definite nello stack.

Questa sezione descrive le API di basso livello disponibili nel framework.

Tipi di asset

AWS CDK Supporta i seguenti tipi di risorse:

Risorse Amazon S3

Si tratta di file e directory locali che vengono AWS CDK caricati su Amazon S3.

Immagine Docker

Si tratta di immagini Docker che vengono AWS CDK caricate su Amazon ECR.

Questi tipi di risorse sono spiegati nelle seguenti sezioni.

Risorse Amazon S3

Puoi definire file e directory locali come risorse, nonché AWS CDK pacchettizzarli e caricarli su Amazon S3 tramite il modulo aws-s3-assets.

L'esempio seguente definisce una risorsa di directory locale e una risorsa di file.

TypeScript

```
import { Asset } from 'aws-cdk-lib/aws-s3-assets';

// Archived and uploaded to Amazon S3 as a .zip file
```

```
const directoryAsset = new Asset(this, "SampleZippedDirAsset", {
  path: path.join(__dirname, "sample-asset-directory")
});

// Uploaded to Amazon S3 as-is
const fileAsset = new Asset(this, 'SampleSingleFileAsset', {
  path: path.join(__dirname, 'file-asset.txt')
});
```

JavaScript

```
const { Asset } = require('aws-cdk-lib/aws-s3-assets');

// Archived and uploaded to Amazon S3 as a .zip file
const directoryAsset = new Asset(this, "SampleZippedDirAsset", {
  path: path.join(__dirname, "sample-asset-directory")
});

// Uploaded to Amazon S3 as-is
const fileAsset = new Asset(this, 'SampleSingleFileAsset', {
  path: path.join(__dirname, 'file-asset.txt')
});
```

Python

```
import os.path
dirname = os.path.dirname(__file__)

from aws_cdk.aws_s3_assets import Asset

# Archived and uploaded to Amazon S3 as a .zip file
directory_asset = Asset(self, "SampleZippedDirAsset",
  path=os.path.join(dirname, "sample-asset-directory")
)

# Uploaded to Amazon S3 as-is
file_asset = Asset(self, 'SampleSingleFileAsset',
  path=os.path.join(dirname, 'file-asset.txt')
)
```

Java

```
import java.io.File;
```

```
import software.amazon.awscdk.services.s3.assets.Asset;

// Directory where app was started
File startDir = new File(System.getProperty("user.dir"));

// Archived and uploaded to Amazon S3 as a .zip file
Asset directoryAsset = Asset.Builder.create(this, "SampleZippedDirAsset")
                .path(new File(startDir, "sample-asset-
directory").toString()).build();

// Uploaded to Amazon S3 as-is
Asset fileAsset = Asset.Builder.create(this, "SampleSingleFileAsset")
                .path(new File(startDir, "file-asset.txt").toString()).build();
```

C#

```
using System.IO;
using Amazon.CDK.AWS.S3.Assets;

// Archived and uploaded to Amazon S3 as a .zip file
var directoryAsset = new Asset(this, "SampleZippedDirAsset", new AssetProps
{
    Path = Path.Combine(Directory.GetCurrentDirectory(), "sample-asset-directory")
});

// Uploaded to Amazon S3 as-is
var fileAsset = new Asset(this, "SampleSingleFileAsset", new AssetProps
{
    Path = Path.Combine(Directory.GetCurrentDirectory(), "file-asset.txt")
});
```

Go

```
dirName, err := os.Getwd()
if err != nil {
  panic(err)
}

awss3assets.NewAsset(stack, jsii.String("SampleZippedDirAsset"),
 &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "sample-asset-directory")),
})
```

```
awss3assets.NewAsset(stack, jsii.String("SampleSingleFileAsset"),
 &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "file-asset.txt")),
})
```

Nella maggior parte dei casi, non è necessario utilizzare direttamente le API del `aws-s3-assets` modulo. I moduli che supportano le risorse, ad esempio`aws-lambda`, dispongono di metodi pratici che consentono di utilizzare le risorse. Per le funzioni Lambda, il metodo statico fromAsset () consente di specificare una directory o un file.zip nel file system locale.

Esempio di funzione Lambda

Un caso d'uso comune è la creazione di funzioni Lambda con il codice del gestore come risorsa Amazon S3.

L'esempio seguente utilizza una risorsa Amazon S3 per definire un gestore Python nella directory locale. `handler` Crea anche una funzione Lambda con la risorsa della directory locale come proprietà. `code` Di seguito è riportato il codice Python per il gestore.

```
def lambda_handler(event, context):
  message = 'Hello World!'
  return {
    'message': message
  }
```

Il codice per l' AWS CDK app principale dovrebbe essere simile al seguente.

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
}
```

Il `Function` metodo utilizza le risorse per raggruppare il contenuto della directory e utilizzarlo per il codice della funzione.

> **Tip**
>
> `.jar`I file Java sono file ZIP con un'estensione diversa. Questi vengono caricati così come sono su Amazon S3, ma quando vengono distribuiti come funzione Lambda, i file che contengono vengono estratti, cosa che potresti non volere. Per evitare ciò, posiziona il `.jar` file in una directory e specifica quella directory come risorsa.

Esempio di attributi Deploy-time

I tipi di asset Amazon S3 espongono anche attributi di implementazione a cui è possibile fare riferimento nelle librerie e nelle app. AWS CDK Il comando AWS CDK CLI cdk synth visualizza le proprietà delle risorse come AWS CloudFormation parametri.

L'esempio seguente utilizza gli attributi deploy-time per passare la posizione di una risorsa di immagine a una funzione Lambda come variabili di ambiente. (Il tipo di file non ha importanza; l'immagine PNG utilizzata qui è solo un esempio.)

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(),
  },
})
```

Autorizzazioni

Se utilizzi gli asset Amazon S3 direttamente tramite il modulo aws-s3-assets, i ruoli, gli utenti o i gruppi IAM e hai bisogno di leggere gli asset in fase di esecuzione, concedi a tali asset le autorizzazioni IAM tramite il metodo asset.grantRead.

L'esempio seguente concede a un gruppo IAM le autorizzazioni di lettura su un file asset.

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)
```

Risorse di immagini Docker

AWS CDK Supporta il raggruppamento di immagini Docker locali come risorse tramite il modulo. aws-ecr-assets

L'esempio seguente definisce un'immagine Docker creata localmente e inviata ad Amazon ECR. Le immagini vengono create da una directory di contesto Docker locale (con un Dockerfile) e caricate su Amazon ECR dalla AWS CDK CLI o dalla pipeline CI/CD dell'app. Le immagini possono essere referenziate in modo naturale nella tua app. AWS CDK

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")),
})
```

La `my-image` directory deve includere un Dockerfile. La AWS CDK CLI crea un'immagine Docker da`my-image`, la invia a un repository Amazon ECR e specifica il nome del repository come parametro dello stack. AWS CloudFormation I tipi di risorse di immagini Docker espongono attributi di implementazione a cui è possibile fare riferimento nelle librerie e nelle app. AWS CDK Il comando AWS CDK CLI cdk synth visualizza le proprietà delle risorse come AWS CloudFormation parametri.

Esempio di definizione di attività in Amazon ECS

Un caso d'uso comune è creare un Amazon ECS TaskDefinitionper eseguire contenitori Docker. L'esempio seguente specifica la posizione di una risorsa di immagine Docker che viene creata localmente e AWS CDK trasferita ad 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),
})
```

Esempio di attributi Deploy-time

L'esempio seguente mostra come utilizzare gli attributi di deploy-time `repository` e `imageUri` creare una definizione di attività Amazon ECS con il AWS Fargate tipo di avvio. Tieni presente che la ricerca del repository Amazon ECR richiede il tag dell'immagine, non il relativo URI, quindi la separiamo dalla fine dell'URI dell'asset.

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()),
})
```

Esempio di compilazione degli argomenti

Puoi fornire argomenti di compilazione personalizzati per la fase di compilazione di Docker tramite l'opzione di proprietà `buildArgs` (Python`build_args`:) quando la AWS CDK CLI crea l'immagine durante la distribuzione.

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"),
  },
})
```

Autorizzazioni

Se utilizzi un modulo che supporta le risorse di immagini Docker, come aws-ecs, AWS CDK gestisce le autorizzazioni per te quando utilizzi le risorse direttamente o tramite. ContainerImage fromEcrRepository(Python:`from_ecr_repository`). Se utilizzi direttamente le risorse di immagini Docker, assicurati che l'utente principale disponga delle autorizzazioni per estrarre l'immagine.

Nella maggior parte dei casi, dovresti usare il metodo asset.repository.grantPull (Python:. `grant_pull` Ciò modifica la politica IAM del principale per consentirgli di estrarre immagini da questo repository. Se il responsabile che sta recuperando l'immagine non si trova nello stesso account, o se si tratta di un AWS servizio che non assume alcun ruolo nel tuo account (ad esempio AWS CodeBuild), devi concedere i permessi di pull sulla politica delle risorse e non sulla politica del principale. Usa asset.repository. addToResourceMetodo di policy (Python:`add_to_resource_policy`) per concedere le autorizzazioni principali appropriate.

AWS CloudFormation metadati delle risorse

> **Note**
>
> Questa sezione è rilevante solo per gli autori dei costrutti. In determinate situazioni, gli strumenti devono sapere che una determinata risorsa CFN utilizza una risorsa locale. Ad esempio, puoi utilizzare la AWS SAM CLI per richiamare le funzioni Lambda localmente per scopi di debug. Per informazioni dettagliate, vedi the section called “AWS SAM integrazione”.

Per abilitare tali casi d'uso, gli strumenti esterni consultano una serie di voci di metadati sulle risorse: AWS CloudFormation

- `aws:asset:path`— Indica il percorso locale della risorsa.
- `aws:asset:property`— Il nome della proprietà della risorsa in cui viene utilizzata la risorsa.

Utilizzando queste due voci di metadati, gli strumenti possono identificare che le risorse vengono utilizzate da una determinata risorsa e consentire esperienze locali avanzate.

Per aggiungere queste voci di metadati a una risorsa, utilizzate il metodo `asset.addResourceMetadata` (Pythonadd_resource_metadata:).

Autorizzazioni

La AWS Construct Library utilizza alcuni idiomi comuni e ampiamente implementati per gestire l'accesso e le autorizzazioni. Il modulo IAM fornisce gli strumenti necessari per utilizzare questi idiomi.

AWS CDK utilizza AWS CloudFormation per implementare le modifiche. Ogni implementazione coinvolge un attore (uno sviluppatore o un sistema automatizzato) che avvia una AWS CloudFormation distribuzione. Nel corso di questa operazione, l'attore assumerà una o più identità IAM (utente o ruoli) e, facoltativamente, assegnerà un ruolo a. AWS CloudFormation

Se utilizzi l' AWS IAM Identity Center autenticazione come utente, il provider Single Sign-On fornisce credenziali di sessione di breve durata che ti autorizzano a svolgere il ruolo IAM predefinito. Per scoprire come AWS CDK ottiene le AWS credenziali dall'autenticazione IAM Identity Center, consulta Understand IAM Identity Center Authentication nella Guida di riferimento agli SDK e agli strumenti.AWS

Principali

Un principal IAM è un' AWS entità autenticata che rappresenta un utente, un servizio o un'applicazione che può chiamare le API. AWS La AWS Construct Library supporta la specificazione dei principali in diversi modi flessibili per consentire loro di accedere alle tue risorse. AWS

Nei contesti di sicurezza, il termine «principale» si riferisce specificamente a entità autenticate come gli utenti. Oggetti come gruppi e ruoli non rappresentano gli utenti (e altre entità autenticate) ma li identificano indirettamente allo scopo di concedere le autorizzazioni.

Ad esempio, se crei un gruppo IAM, puoi concedere al gruppo (e quindi ai suoi membri) l'accesso in scrittura a una tabella Amazon RDS. Tuttavia, il gruppo stesso non è un principale perché non rappresenta una singola entità (inoltre, non è possibile accedere a un gruppo).

Nella libreria IAM del CDK, le classi che identificano direttamente o indirettamente i principali implementano l'`IPrincipal`interfaccia, che consente di utilizzare questi oggetti in modo intercambiabile nelle politiche di accesso. Tuttavia, non tutti sono principi in senso di sicurezza. Questi oggetti includono:

1. Risorse IAM come `RoleUser`, e `Group`
2. Responsabili del servizio () `new iam.ServicePrincipal('service.amazonaws.com')`

3. Responsabili federati () `new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`
4. Responsabili del conto (`new iam.AccountPrincipal('0123456789012')`)
5. Principi utente canonici () `new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`
6. AWS Organizations presidi () `new iam.OrganizationPrincipal('org-id')`
7. Principi ARN arbitrari () `new iam.ArnPrincipal(res.arn)`
8. E `iam.CompositePrincipal(principal1, principal2, ...)` fidarsi di più principi

Concessioni

Ogni costrutto che rappresenta una risorsa a cui è possibile accedere, ad esempio un bucket Amazon S3 o una tabella Amazon DynamoDB, ha metodi che garantiscono l'accesso a un'altra entità. Tutti questi metodi hanno nomi che iniziano con grant.

Ad esempio, i bucket Amazon S3 dispongono dei metodi e `grantRead` (`grantReadWrite`Python:`grant_read`,`grant_read_write`) per abilitare l'accesso in lettura e lettura/scrittura, rispettivamente, da un'entità al bucket. L'entità non deve sapere esattamente quali autorizzazioni IAM di Amazon S3 sono necessarie per eseguire queste operazioni.

Il primo argomento di un metodo di concessione è sempre di tipo IGranTable. Questa interfaccia rappresenta le entità a cui possono essere concesse le autorizzazioni. Cioè, rappresenta risorse con ruoli, come gli oggetti IAM `RoleUser`, e`Group`.

Le autorizzazioni possono essere concesse anche ad altre entità. Ad esempio, più avanti in questo argomento, mostriamo come concedere a un CodeBuild progetto l'accesso a un bucket Amazon S3. In genere, il ruolo associato viene ottenuto tramite una `role` proprietà sull'entità a cui viene concesso l'accesso.

Anche le risorse che utilizzano ruoli di esecuzione, ad esempio`lambda.Function`, implementano`IGrantable`, quindi è possibile concedere loro l'accesso diretto anziché concedere l'accesso al loro ruolo. Ad esempio, se `bucket` è un bucket Amazon S3 ed è `function` una funzione Lambda, il codice seguente concede alla funzione l'accesso in lettura al bucket.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

A volte le autorizzazioni devono essere applicate mentre lo stack viene distribuito. Uno di questi casi si verifica quando si concede a una risorsa AWS CloudFormation personalizzata l'accesso a un'altra risorsa. La risorsa personalizzata verrà richiamata durante la distribuzione, quindi deve disporre delle autorizzazioni specificate al momento della distribuzione.

Un altro caso si verifica quando un servizio verifica che al ruolo assegnato siano applicate le politiche corrette. (Alcuni AWS servizi eseguono questa operazione per assicurarsi che tu non abbia dimenticato di impostare le politiche.) In questi casi, la distribuzione potrebbe non riuscire se le autorizzazioni vengono applicate troppo tardi.

Per forzare l'applicazione delle autorizzazioni della concessione prima che venga creata un'altra risorsa, puoi aggiungere una dipendenza dalla concessione stessa, come mostrato qui. Sebbene il valore restituito dai metodi di concessione venga comunemente scartato, ogni metodo di concessione restituisce in effetti un oggetto. `iam.Grant`

TypeScript

```
const grant = bucket.grantRead(lambda);
const custom = new CustomResource(...);
custom.node.addDependency(grant);
```

JavaScript

```
const grant = bucket.grantRead(lambda);
const custom = new CustomResource(...);
custom.node.addDependency(grant);
```

Python

```
grant = bucket.grant_read(function)
custom = CustomResource(...)
custom.node.add_dependency(grant)
```

Java

```
Grant grant = bucket.grantRead(function);
CustomResource custom = new CustomResource(...);
custom.node.addDependency(grant);
```

C#

```
var grant = bucket.GrantRead(function);
var custom = new CustomResource(...);
custom.node.AddDependency(grant);
```

Roles

Il pacchetto IAM contiene un `Role` costrutto che rappresenta i ruoli IAM. Il codice seguente crea un nuovo ruolo, affidandosi al servizio 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
});
```

Puoi aggiungere autorizzazioni a un ruolo chiamando il `addToPolicy` metodo del ruolo (Python`add_to_policy`:), passando `PolicyStatement` un che definisce la regola da aggiungere. L'istruzione viene aggiunta alla politica predefinita del ruolo; se non ne ha, ne viene creata una.

L'esempio seguente aggiunge una dichiarazione di `Deny` policy al ruolo per le azioni `ec2:SomeAction` e `s3:AnotherAction` sulle risorse `bucket` e `otherRole` (Python:`other_role`), a condizione che il servizio autorizzato sia. 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"
        }
    }
}));
```

Nell'esempio precedente, abbiamo creato un nuovo `PolicyStatement` inline con la chiamata (`addToPolicy`Python:). `add_to_policy` Puoi anche inserire una dichiarazione politica esistente o una che hai modificato. L'PolicyStatementoggetto dispone di numerosi metodi per aggiungere principi, risorse, condizioni e azioni.

Se utilizzate un costrutto che richiede un ruolo per funzionare correttamente, potete effettuare una delle seguenti operazioni:

- Assegnate un ruolo esistente quando istanziate l'oggetto di costruzione.
- Lascia che il costrutto crei un nuovo ruolo per te, affidandoti al responsabile del servizio appropriato. L'esempio seguente utilizza un costrutto di questo tipo: un progetto. CodeBuild

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
});
```

Una volta creato l'oggetto, il ruolo (indipendentemente dal ruolo passato o da quello predefinito creato dal costrutto) è disponibile come proprietà. `role` Tuttavia, questa proprietà non è disponibile su risorse esterne. Pertanto, questi costrutti hanno un metodo `addToRolePolicy` (Python`add_to_role_policy`:).

Il metodo non fa nulla se il costrutto è una risorsa esterna e in caso contrario chiama il metodo `addToPolicy` (Python`add_to_policy`:) `role` della proprietà. Questo ti evita la fatica di gestire il caso indefinito in modo esplicito.

L'esempio seguente dimostra:

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
}));
```

Policy delle risorse

Alcune risorse AWS, come i bucket Amazon S3 e i ruoli IAM, hanno anche una politica delle risorse. Questi costrutti hanno un `addToResourcePolicy` metodo (Python`add_to_resource_policy`:), che accetta `PolicyStatement` a come argomento. Ogni dichiarazione politica aggiunta a una politica delle risorse deve specificare almeno un principio.

Nell'esempio seguente, il bucket Amazon S3 `bucket` concede un ruolo con l'`s3:SomeAction`autorizzazione a se stesso.

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 }
}));
```

Utilizzo di oggetti IAM esterni

Se hai definito un utente, un principale, un gruppo o un ruolo IAM al di fuori AWS CDK dell'app, puoi utilizzare quell'oggetto IAM nell' AWS CDK app. A tale scopo, create un riferimento ad esso utilizzando il suo ARN o il suo nome. (Usa il nome per utenti, gruppi e ruoli). Il riferimento restituito può quindi essere utilizzato per concedere autorizzazioni o per creare dichiarazioni politiche come spiegato in precedenza.

- Per gli utenti, chiama `User.fromUserArn()` o. `User.fromUserName()` `User.fromUserAttributes()`è anche disponibile, ma attualmente offre le stesse funzionalità di`User.fromUserArn()`.
- Per i principali, crea un'istanza di un oggetto. `ArnPrincipal`
- Per i gruppi, chiama o. `Group.fromGroupArn()` `Group.fromGroupName()`
- Per ruoli, chiama `Role.fromRoleArn()` o`Role.fromRoleName()`.

Le politiche (incluse le politiche gestite) possono essere utilizzate in modo simile utilizzando i seguenti metodi. È possibile utilizzare riferimenti a questi oggetti ovunque sia richiesta una policy IAM.

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

> **Note**
>
> Come per tutti i riferimenti a AWS risorse esterne, non è possibile modificare oggetti IAM esterni nell'app CDK.

Contesto di runtime

I valori di contesto sono coppie chiave-valore che possono essere associate a un'app, uno stack o un costrutto. Possono essere forniti all'app da un file (di solito o `cdk.context.json` nella directory del progetto) `cdk.json` o dalla riga di comando.

Il CDK Toolkit utilizza il contesto per memorizzare nella cache i valori recuperati dall' AWS account durante la sintesi. I valori includono le zone di disponibilità nel tuo account o gli ID Amazon Machine Image (AMI) attualmente disponibili per le istanze Amazon EC2. Poiché questi valori sono forniti dal tuo AWS account, possono cambiare tra le esecuzioni dell'applicazione CDK. Ciò li rende una potenziale fonte di cambiamenti involontari. Il comportamento di memorizzazione nella cache di CDK Toolkit «blocca» questi valori per l'app CDK finché non decidi di accettare i nuovi valori.

Immaginate lo scenario seguente senza la memorizzazione nella cache del contesto. Supponiamo che tu abbia specificato «la versione più recente di Amazon Linux» come AMI per le tue istanze Amazon EC2 e che sia stata rilasciata una nuova versione di questa AMI. Quindi, la prossima volta che distribuisci lo stack CDK, le istanze già distribuite utilizzeranno l'AMI obsoleta («sbagliata») e dovranno essere aggiornate. L'aggiornamento comporterebbe la sostituzione di tutte le istanze esistenti con altre nuove, il che sarebbe probabilmente inaspettato e indesiderato.

Invece, il CDK registra le AMI disponibili del tuo account nel `cdk.context.json` file del tuo progetto e utilizza il valore memorizzato per future operazioni di sintesi. In questo modo, l'elenco delle AMI non è più una potenziale fonte di cambiamento. Puoi anche essere certo che i tuoi stack vengano sempre sintetizzati negli stessi modelli. AWS CloudFormation

Non tutti i valori di contesto sono valori memorizzati nella cache del tuo ambiente. AWS the section called “Bandiere di funzionalità”sono anche valori di contesto. Puoi anche creare i tuoi valori di contesto da utilizzare con le tue app o costrutti.

Le chiavi di contesto sono stringhe. I valori possono essere di qualsiasi tipo supportato da JSON: numeri, stringhe, matrici o oggetti.

> **Tip**
>
> Se i tuoi costrutti creano i propri valori di contesto, incorpora il nome del pacchetto della libreria nelle sue chiavi in modo che non entrino in conflitto con i valori di contesto di altri pacchetti.

Molti valori di contesto sono associati a un particolare AWS ambiente e una determinata app CDK può essere distribuita in più di un ambiente. La chiave per tali valori include l' AWS account e la regione, in modo che i valori di ambienti diversi non siano in conflitto.

La seguente chiave di contesto illustra il formato utilizzato da AWS CDK, inclusi l'account e la regione.

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

> **Important**
>
> I valori di contesto memorizzati nella cache sono gestiti da AWS CDK e dai relativi costrutti, inclusi i costrutti che è possibile scrivere. Non aggiungete o modificate i valori di contesto memorizzati nella cache modificando manualmente i file. Può essere utile, tuttavia, rivedere di `cdk.context.json` tanto in tanto per vedere quali valori vengono memorizzati nella cache. I valori di contesto che non rappresentano i valori memorizzati nella cache devono essere archiviati sotto la `context` chiave di. `cdk.json` In questo modo, non verranno cancellati quando i valori memorizzati nella cache vengono cancellati.

Fonti di valori contestuali

I valori di contesto possono essere forniti all' AWS CDK app in sei modi diversi:

- Automaticamente dall' AWS account corrente.
- Tramite l'`--context`opzione al cdk comando. (Questi valori sono sempre stringhe.)
- Nel `cdk.context.json` file del progetto.
- Nella `context` chiave del `cdk.json` file del progetto.
- Nella `context` chiave del tuo `~/.cdk.json` file.
- Nella tua AWS CDK app usando il `construct.node.setContext()` metodo.

Il file di progetto `cdk.context.json` è dove vengono memorizzati nella AWS CDK cache i valori di contesto recuperati dal tuo AWS account. Questa pratica evita modifiche impreviste alle distribuzioni quando, ad esempio, viene introdotta una nuova zona di disponibilità. AWS CDK Non scrive dati contestuali in nessuno degli altri file elencati.

> **Important**
>
> Perché fanno parte dello stato dell'applicazione `cdk.json` e `cdk.context.json` devono essere sottoposti al controllo del codice sorgente insieme al resto del codice sorgente dell'app. In caso contrario, le implementazioni in altri ambienti (ad esempio, una pipeline CI) potrebbero produrre risultati incoerenti.

I valori di contesto rientrano nell'ambito del costrutto che li ha creati; sono visibili ai costrutti figli, ma non ai genitori o ai fratelli. I valori di contesto impostati dal AWS CDK Toolkit (il cdk comando) possono essere impostati automaticamente, da un file o dall'opzione. --context I valori di contesto di queste fonti sono impostati implicitamente nel App costrutto. Pertanto, sono visibili a tutti i costrutti in ogni stack dell'app.

La tua app può leggere un valore di contesto utilizzando il `construct.node.tryGetContext` metodo. Se la voce richiesta non viene trovata nel costrutto corrente o in uno dei suoi genitori, il risultato è`undefined`. (In alternativa, il risultato potrebbe essere l'equivalente della tua lingua, ad esempio `None` in Python.)

Metodi del contesto

AWS CDK Supporta diversi metodi contestuali che consentono alle AWS CDK app di ottenere informazioni contestuali dall' AWS ambiente. Ad esempio, puoi ottenere un elenco di zone di disponibilità disponibili in un determinato AWS account e regione, utilizzando il metodo Stack.availabilityZones.

Di seguito sono riportati i metodi contestuali:

HostedZone.fromLookup

Ottiene le zone ospitate nel tuo account.

Stack.Availability Zones

Ottiene le zone di disponibilità supportate.

StringParameter.valueFromLookup

Ottiene un valore dall'Amazon EC2 Systems Manager Parameter Store della regione corrente.

vpc.fromLookup

Acquisisce gli Amazon Virtual Private Cloud esistenti nei tuoi account.

LookupMachineImage

Cerca l'immagine di una macchina da utilizzare con un'istanza NAT in un Amazon Virtual Private Cloud.

Se un valore di contesto richiesto non è disponibile, l' AWS CDK app notifica al CDK Toolkit che le informazioni di contesto sono mancanti. Successivamente, la CLI interroga l' AWS account corrente per le informazioni e memorizza le informazioni di contesto risultanti nel file. `cdk.context.json` Quindi, esegue nuovamente l' AWS CDK app con i valori di contesto.

Visualizzazione e gestione del contesto

Usa il cdk context comando per visualizzare e gestire le informazioni nel tuo `cdk.context.json` file. Per visualizzare queste informazioni, usa il cdk context comando senza alcuna opzione. L'output dovrebbe essere simile al seguente.

```
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.
```

Per rimuovere un valore di contesto, eseguicdk context --reset, specificando la chiave o il numero corrispondente al valore. L'esempio seguente rimuove il valore che corrisponde alla seconda chiave dell'esempio precedente. Questo valore rappresenta l'elenco delle zone di disponibilità nella regione 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.
```

Pertanto, se desideri eseguire l'aggiornamento all'ultima versione dell'AMI Amazon Linux, utilizza l'esempio precedente per eseguire un aggiornamento controllato del valore di contesto e reimpostarlo. Quindi, sintetizza e distribuisci nuovamente l'app.

```
cdk synth
```

Per cancellare tutti i valori di contesto memorizzati per la tua appcdk context --clear, esegui come segue.

```
cdk context --clear
```

Solo i valori di contesto memorizzati in `cdk.context.json` possono essere ripristinati o cancellati. AWS CDK Non tocca altri valori di contesto. Pertanto, per proteggere un valore di contesto dalla reimpostazione mediante questi comandi, è possibile copiare il valore in`cdk.json`.

AWS CDK Bandiera Toolkit `--context`

Utilizzate l'opzione `--context` (in breve) `-c` per passare i valori del contesto di runtime all'app CDK durante la sintesi o la distribuzione.

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

Per specificare più valori di contesto, ripetete l'--contextopzione un numero qualsiasi di volte, fornendo ogni volta una coppia chiave-valore.

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

Quando si sintetizzano più stack, i valori di contesto specificati vengono passati a tutti gli stack. Per fornire valori di contesto diversi ai singoli stack, utilizzate chiavi diverse per i valori oppure utilizzate più comandi o. cdk synth cdk deploy

I valori di contesto passati dalla riga di comando sono sempre stringhe. Se un valore è in genere di un altro tipo, il codice deve essere preparato per convertire o analizzare il valore. È possibile che i valori di contesto non stringhe vengano forniti in altri modi (ad esempio, in`cdk.context.json`). Per assicurarti che questo tipo di valore funzioni come previsto, verifica che il valore sia una stringa prima di convertirlo.

Esempio

Di seguito è riportato un esempio di utilizzo di un Amazon VPC esistente utilizzando AWS CDK il contesto.

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()
        });
    }
}
```

Puoi usarlo cdk diff per vedere gli effetti del passaggio di un valore di contesto sulla riga di comando:

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

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

I valori di contesto risultanti possono essere visualizzati come illustrato qui.

```
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"
    ]
  }
}
```

Bandiere caratteristiche

AWS CDK Utilizza i flag di funzionalità per abilitare comportamenti potenzialmente dannosi in una versione. I flag vengono memorizzati come the section called “Context” valori in `cdk.json` (o). `~/.cdk.json` Non vengono rimossi dai cdk context --clear comandi cdk context --reset o.

I flag delle funzionalità sono disabilitati per impostazione predefinita. I progetti esistenti che non specificano il flag continueranno a funzionare come prima con le AWS CDK versioni successive. I nuovi progetti creati utilizzando i flag cdk init includono che abilitano tutte le funzionalità disponibili nella versione che ha creato il progetto. Modifica `cdk.json` per disabilitare tutti i flag per i quali preferisci il comportamento precedente. È inoltre possibile aggiungere flag per abilitare nuovi comportamenti dopo l'aggiornamento di. AWS CDK

Un elenco di tutti i flag di funzionalità correnti è disponibile nel repository in. AWS CDK GitHub `FEATURE_FLAGS.md` Per una descrizione di tutti i nuovi flag di funzionalità aggiunti `CHANGELOG` in quella versione, consulta la sezione relativa a una determinata versione.

Ripristino del comportamento v1

In CDK v2, le impostazioni predefinite per alcuni flag di funzionalità sono state modificate rispetto alla v1. È possibile reimpostarli su per ripristinare il comportamento specifico della `false` v1. AWS CDK Utilizzate il `cdk diff` comando per esaminare le modifiche al modello sintetizzato per vedere se qualcuno di questi flag è necessario.

`@aws-cdk/core:newStyleStackSynthesis`

Usa il nuovo metodo di sintesi dello stack, che presuppone risorse bootstrap con nomi noti. Richiede un bootstrap moderno, ma a sua volta consente la CI/CD tramite CDK Pipelines e le distribuzioni tra account preconfigurate.

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

Se la tua applicazione utilizza più chiavi API Amazon API Gateway e le associa a piani di utilizzo.

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

Se la tua applicazione utilizza istanze di database Amazon RDS o cluster di database e specifica esplicitamente l'identificatore per questi.

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

Se la tua applicazione utilizza la politica di sicurezza TLS_V1_2_2019 con distribuzioni. Amazon CloudFront CDK v2 utilizza la politica di sicurezza TLSv1.2_2021 per impostazione predefinita.

`@aws-cdk/core:stackRelativeExports`

Se l'applicazione utilizza più stack e si fa riferimento alle risorse di uno stack all'altro, ciò determina se viene utilizzato un percorso assoluto o relativo per costruire le esportazioni. AWS CloudFormation

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

Se impostato su`false`, il CDK include i metadati quando rileva se una funzione Lambda è cambiata. Ciò può causare errori di distribuzione quando sono stati modificati solo i metadati, poiché non sono consentite versioni duplicate. Non è necessario ripristinare questo flag se hai apportato almeno una modifica a tutte le funzioni Lambda dell'applicazione.

La sintassi per ripristinare questi flag è mostrata qui. `cdk.json`

```
{
  "context": {
    "@aws-cdk/core:newStyleStackSynthesis": false,
    "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": false,
    "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": false,
    "@aws-cdk/aws-rds:lowercaseDbIdentifier": false,
    "@aws-cdk/core:stackRelativeExports": false,
    "@aws-cdk/aws-lambda:recognizeVersionProps": false
  }
```

```
}
```

Aspetti

Gli aspetti sono un modo per applicare un'operazione a tutti i costrutti in un determinato ambito. L'aspetto potrebbe modificare i costrutti, ad esempio aggiungendo tag. Oppure potrebbe verificare qualcosa sullo stato dei costrutti, ad esempio assicurarsi che tutti i bucket siano crittografati.

Per applicare un aspetto a un costrutto e a tutti i costrutti nello stesso ambito, chiamate `Aspects.of(SCOPE).add()`con un nuovo aspetto, come mostrato nell'esempio seguente.

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 Utilizza gli aspetti per etichettare le risorse, ma il framework può essere utilizzato anche per altri scopi. Ad esempio, potete usarlo per convalidare o modificare le AWS CloudFormation risorse definite per voi da costrutti di livello superiore.

Aspetti in dettaglio

Gli aspetti utilizzano lo schema dei visitatori. Un aspetto è una classe che implementa la seguente interfaccia.

TypeScript

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

JavaScript

JavaScript non ha interfacce come funzionalità linguistica. Pertanto, un aspetto è semplicemente un'istanza di una classe con un `visit` metodo che accetta il nodo su cui operare.

Python

Python non ha interfacce come funzionalità del linguaggio. Pertanto, un aspetto è semplicemente un'istanza di una classe con un `visit` metodo che accetta il nodo su cui operare.

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 si chiama`Aspects.of(`*`SCOPE`*`).add(...)`, il costrutto aggiunge l'aspetto a un elenco interno di aspetti. È possibile ottenere l'elenco con`Aspects.of(`*`SCOPE`*`)`.

Durante la fase di preparazione, AWS CDK chiama il `visit` metodo dell'oggetto per il costrutto e ciascuno dei suoi figli in ordine dall'alto verso il basso.

Il `visit` metodo è libero di modificare qualsiasi cosa nel costrutto. In linguaggi fortemente tipizzati, trasmetti il costrutto ricevuto a un tipo più specifico prima di accedere a proprietà o metodi specifici del costrutto.

Gli aspetti non si propagano oltre i confini del `Stage` costrutto, perché dopo la definizione `Stages` sono autonomi e immutabili. Applica gli aspetti sul `Stage` costrutto stesso (o inferiore) se vuoi che visitino i costrutti all'interno di. `Stage`

Esempio

L'esempio seguente verifica che tutti i bucket creati nello stack abbiano il controllo delle versioni abilitato. L'aspetto aggiunge un'annotazione di errore ai costrutti che non superano la convalida. Ciò comporta il fallimento dell'synthoperazione e impedisce la distribuzione dell'assemblaggio cloud risultante.

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());
```

Iniziare con AWS CDK

Inizia installando AWS CDK CLI e creando la tua prima app CDK. AWS Cloud Development Kit (AWS CDK)

Argomenti

Prerequisiti

Risorse consigliate

Prima di iniziare con AWS CDK, ti consigliamo di acquisire una conoscenza di base di quanto segue:

- Un'introduzione a AWS CDK. Per ulteriori informazioni, consulta Che cos'è il AWS CDK?
- Concetti fondamentali alla base di AWS CDK. Per ulteriori informazioni, vedi AWS CDK concetti.
- Quello Servizi AWS che vuoi gestire con AWS CDK.
- AWS Identity and Access Management. Per ulteriori informazioni, consulta Cos'è IAM? e cos'è IAM Identity Center?
- AWS CloudFormation poiché AWS CDK utilizza il AWS CloudFormation servizio per fornire risorse create nel CDK. Per ulteriori informazioni, consulta Che cos'è AWS CloudFormation?
- Il linguaggio di programmazione supportato che si prevede di utilizzare con. AWS CDK

Prepara il tuo ambiente locale

Tutti AWS CDK gli sviluppatori, indipendentemente dalla lingua preferita, necessitano della versione Node.js14.15.0 o successiva. Tutti i linguaggi di programmazione supportati utilizzano lo stesso backend, che funziona su. Node.js Consigliamo una versione con supporto attivo a lungo termine. La tua organizzazione potrebbe avere una raccomandazione diversa.

> **Important**
>
> Le versioni da 13.0.0 a 13.6.0 di Node.js non sono compatibili con a AWS CDK causa di problemi di compatibilità con le relative dipendenze.

Gli altri prerequisiti dipendono dal linguaggio in cui si sviluppano AWS CDK le applicazioni e sono i seguenti.

TypeScript

- TypeScript 3.8 o versione successiva () `npm -g install typescript`

JavaScript

Nessun requisito aggiuntivo

Python

- Python 3.7 o successivo incluso e `pip` `virtualenv`

Java

- Java Development Kit (JDK) 8 (alias 1.8) o versione successiva
- Apache Maven 3.5 o versione successiva

Java IDE consigliato (utilizziamo Eclipse in alcuni esempi di questa guida). L'IDE deve essere in grado di importare progetti Maven. Assicurati che il tuo progetto sia impostato per utilizzare Java 1.8. Imposta la variabile di ambiente JAVA_HOME sul percorso in cui hai installato il JDK.

C#

.NET Core 3.1 o versione successiva oppure .NET 6.0 o versione successiva.

Si consiglia Visual Studio 2019 (qualsiasi edizione) o Visual Studio Code.

Go

Passa alla versione 1.1.8 o successiva.

Per informazioni più dettagliate, consulta la sezione Prerequisiti per la tua lingua:

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

> Deprecazione della lingua di terze parti
>
> Ogni versione linguistica è supportata solo fino alla fine del ciclo di vita ed è soggetta a modifiche con preavviso. EOL

Fase 1: Creare un Account AWS

Se sei nuovo AWS, devi registrarti Account AWS e creare un utente amministrativo. Per ulteriori informazioni, consulta Getting up up with IAM nella IAM User Guide.

Quando interagisci con AWS, specifichi le tue credenziali di AWS sicurezza per verificare chi sei e se disponi dell'autorizzazione per accedere alle risorse che stai richiedendo. AWS utilizza le credenziali di sicurezza per autenticare e autorizzare le richieste. Per ulteriori informazioni, consulta le credenziali AWS di sicurezza nella Guida per l'utente IAM.

Fase 2: Configurazione dell'accesso programmatico

Quando svilupperai con the AWS CDK nel tuo ambiente locale, farai affidamento su di loro AWS CDK CLI per interagire Servizi AWS e gestire AWS le tue risorse. Per utilizzare AWS CDK CLI, è necessario configurare l'accesso programmatico. Per ulteriori informazioni sui diversi modi in cui è possibile configurare l'accesso programmatico, consulta Autenticazione e accesso nella Guida di riferimento agli AWS SDK e agli strumenti.

Per i nuovi utenti a cui non viene fornito un metodo di autenticazione dal datore di lavoro, consigliamo di utilizzare. AWS IAM Identity Center Questo metodo include l'installazione di AWS Command Line Interface (AWS CLI) e il suo utilizzo per la configurazione e l' AWS accesso al portale di accesso.

Per configurare l'accesso programmatico utilizzando IAM Identity Center, consulta l'autenticazione IAM Identity Center nella Guida di riferimento agli AWS SDK e agli strumenti. Dopo il completamento, l'ambiente dovrebbe contenere i seguenti elementi:

- Il AWS CLI, che viene utilizzato per avviare una sessione del portale di AWS accesso prima di eseguire l'applicazione.
- Un AWS`config`file condiviso con un `[default]` profilo con un set di valori di configurazione a cui è possibile fare riferimento da. AWS CDK Per trovare la posizione di questo file, consulta l'argomento relativo alla posizione dei file condivisi nella Guida di riferimento per SDK e strumenti AWS .
- Il `config` file condiviso imposta l'`region`impostazione. Questo imposta gli AWS CDK usi predefiniti Regione AWS per AWS le richieste.
- AWS CDK Utilizza la configurazione del provider di token SSO del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il `sso_role_name` valore, che è un ruolo IAM connesso a un set di autorizzazioni IAM Identity Center, dovrebbe consentire l'accesso ai dati Servizi AWS utilizzati nell'applicazione.

  Il seguente `config` file di esempio mostra un profilo predefinito impostato con la configurazione del provider di token SSO. L'`sso_session`impostazione del profilo si riferisce alla `sso-session`sezione denominata. La `sso-session` sezione contiene le impostazioni per avviare una sessione del portale di AWS accesso.

```
[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
```

Avviare una sessione del portale di AWS accesso

Prima di accedere Servizi AWS, è necessaria una sessione attiva del portale di AWS accesso AWS CDK per utilizzare l'autenticazione IAM Identity Center per risolvere le credenziali. A seconda della

durata della sessione configurata, l'accesso alla fine scadrà e si AWS CDK verificherà un errore di autenticazione. Esegui il seguente comando in AWS CLI per accedere al portale di AWS accesso.

```
aws sso login
```

Se la configurazione del provider di token SSO utilizza un profilo denominato anziché il profilo predefinito, il comando è`aws sso login --profile` *NAME*. Specificate inoltre questo profilo quando emettete cdk comandi utilizzando l'`--profile`opzione o la variabile di `AWS_PROFILE` ambiente.

Per verificare se hai già una sessione attiva, esegui il AWS CLI comando seguente.

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

La risposta a questo comando dovrebbe restituire l'account IAM Identity Center e il set di autorizzazioni configurati nel file `config` condiviso.

> **Note**
>
> Se hai già una sessione attiva del portale di AWS accesso ed eseguia`aws sso login`, non ti verrà richiesto di fornire credenziali.
> La procedura di accesso potrebbe richiedere all'utente di consentire l' AWS CLI accesso ai dati. Poiché AWS CLI è basato sull'SDK per Python, i messaggi di autorizzazione possono contenere variazioni del `botocore` nome.

Fase 3: Installare AWS CDKCLI

Installa AWS CDK CLI globalmente utilizzando il seguente comando Node Package Manager.

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

> **Note**
>
> Se ricevi un errore di autorizzazione e disponi dell'accesso come amministratore sul tuo sistema, provasudo `npm install -g aws-cdk`.

Esegui il comando seguente per verificare l'avvenuta installazione. AWS CDK CLIDovrebbe mostrare il numero di versione:

```
cdk --version
```

Se ricevi un messaggio di errore, prova a disinstallarlo AWS CDK CLI eseguendo quanto segue:

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

Quindi, ripeti i passaggi per reinstallare. AWS CDK CLI

Se il messaggio di errore persiste, elimina la `node-modules` cartella dal progetto corrente e anche dalla `node-modules` cartella globale. Per individuare questa cartella, eseguinpm `config get prefix`.

AWS CDK CLIOtterrà le credenziali di sicurezza dalle fonti configurate nei passaggi precedenti.

> **Note**
>
> CDK Toolkit v2 funziona con i progetti CDK v1 esistenti. Tuttavia, non può inizializzare nuovi progetti CDK v1. Vedi the section called “Nuovi prerequisiti” se devi essere in grado di farlo.

Fase 4: Avvia il tuo ambiente

Ogni AWSambiente in cui intendi distribuire le risorse deve essere avviato.

Per eseguire il bootstrap, esegui quanto segue:

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

> **Tip**
>
> Se non hai il tuo numero di AWS conto a portata di mano, puoi richiederlo AWS Management Console da. Oppure, se lo hai AWS CLI installato, il comando seguente mostra le informazioni predefinite del tuo account, incluso il numero di account.
>
> ```
> aws sts get-caller-identity
> ```

Se hai creato profili denominati nella AWS configurazione locale, puoi utilizzare l'`--profile`opzione per visualizzare le informazioni sull'account per un profilo specifico. L'esempio seguente mostra come visualizzare le informazioni sull'account per il profilo prod.

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

Per visualizzare la regione predefinita, utilizzare`aws configure get`.

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

AWS CDK Strumenti opzionali

AWS Toolkit for Visual Studio CodeÈ un plug-in open source per Visual Studio Code che consente di creare, eseguire il debug e distribuire applicazioni su. AWS Il toolkit offre un'esperienza integrata per lo sviluppo di applicazioni. AWS CDK Include la funzionalità AWS CDK Explorer per elencare i AWS CDK progetti e sfogliare i vari componenti dell'applicazione CDK. Installa il plug-in e scopri di più sull'utilizzo di Explorer. AWS CDK

Passaggi successivi

Ora che lo hai installato AWS CDK CLI, usalo per creare la tua prima AWS CDK app.

Per ulteriori informazioni sull'utilizzo di AWS CDK nel linguaggio di programmazione preferito, consultaLavorare con i AWS CDK linguaggi di programmazione supportati.

AWS CDK Si tratta di un progetto open source. Per contribuire, vedi Contributing to. AWS Cloud Development Kit (AWS CDK)

Ulteriori informazioni

Per ulteriori informazioni su AWS CDK, consulta quanto segue:

- Workshop CDK — Workshop pratico approfondito.
- Riferimento alle API: esplora i costrutti disponibili per i Servizi AWS quali utilizzerai.

- Construct Hub: trova i costrutti della community CDK.
- AWS CDK esempi: esplora esempi di codice di progetti. AWS CDK

La tua prima AWS CDK app

Inizia a utilizzarla AWS Cloud Development Kit (AWS CDK) creando la tua prima app CDK.

Prima di iniziare questo tutorial, ti consigliamo di completare quanto segue:

- Vedi Che cos'è il AWS CDK? per un'introduzione a AWS CDK.
- Vedi AWS CDK concetti per apprendere i concetti fondamentali di AWS CDK.
- Esegui i prerequisiti e le fasi AWS CDK di configurazione suIniziare con AWS CDK.

Argomenti

Informazioni sul tutorial

In questo tutorial, creerai e distribuirai una semplice AWS CDK app. Questa app contiene uno stack con una singola risorsa bucket Amazon Simple Storage Service (Amazon S3). Attraverso questo tutorial, imparerai quanto segue:

- La struttura di un AWS CDK progetto.
- Come creare un' AWS CDK app.

- Come utilizzare la AWS Construct Library per definire app, stack e AWS risorse.
- Come utilizzare il CDK CLI per sintetizzare, differenziare, distribuire ed eliminare l'app CDK.
- Come modificare e ridistribuire l'app CDK per aggiornare le risorse distribuite.

Il flusso di lavoro di AWS CDK sviluppo standard prevede i seguenti passaggi:

1. Crea la tua AWS CDK app: qui, utilizzerai un modello fornito da AWS CDK CLI.
2. Definisci gli stack e le risorse: utilizza i costrutti per definire gli stack e AWS le risorse all'interno della tua app.
3. Crea la tua app: questo passaggio è facoltativo. Esegue AWS CDK CLI automaticamente questo passaggio se necessario. Si consiglia di eseguire questo passaggio per identificare gli errori di sintassi e di tipo.
4. Sintetizza i tuoi stack: questo passaggio crea un AWS CloudFormation modello per ogni stack dell'app. Questo passaggio è utile per identificare gli errori logici nelle risorse definite. AWS
5. Implementa la tua app: esegui la distribuzione nel tuo AWS ambiente utilizzando AWS CloudFormation per il provisioning delle tue risorse. Durante la distribuzione, identificherai eventuali problemi di autorizzazione con la tua app.

Attraverso un tipico flusso di lavoro, tornerai indietro e ripeterai i passaggi precedenti per modificare o eseguire il debug dell'app.

Ti consigliamo di utilizzare il controllo della versione per i tuoi AWS CDK progetti.

Passaggio 1: crea l'app

Un'app CDK deve trovarsi nella propria directory, con le proprie dipendenze dei moduli locali. Sul tuo computer di sviluppo, crea una nuova directory. Di seguito è riportato un esempio che crea una nuova `hello-cdk` directory:

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

> **Important**
>
> Assicuratevi di assegnare un nome alla cartella del progetto`hello-cdk`, esattamente come mostrato qui. Il modello di AWS CDK progetto utilizza il nome della directory per

denominare le cose nel codice generato. Se usi un nome diverso, il codice in questo tutorial non funzionerà.

Successivamente, dalla nuova directory, inizializza l'app utilizzando il cdk init comando. Specificate il `app` modello e il linguaggio di programmazione preferito con l'`--language`opzione. Di seguito è riportato un esempio:

TypeScript

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

JavaScript

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

Python

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

Dopo aver creato l'app, inserisci anche i due comandi seguenti. Questi attivano l'ambiente virtuale Python dell'app e installano le dipendenze AWS CDK principali.

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

Java

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

Se utilizzi un IDE, ora puoi aprire o importare il progetto. In Eclipse, ad esempio, scegli File > Importa > Maven > Progetti Maven esistenti. Assicuratevi che le impostazioni del progetto siano impostate per utilizzare Java 8 (1.8).

C#

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

Se utilizzi Visual Studio, apri il file della soluzione nella `src` directory.

Go

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

Dopo aver creato l'app, inserisci anche il seguente comando per installare i moduli AWS Construct Library richiesti dall'app.

```
go get
```

Il cdk init comando crea una serie di file e cartelle all'interno della `hello-cdk` directory per aiutarvi a organizzare il codice sorgente AWS CDK dell'app. Collettivamente, questo è chiamato il vostro AWS CDK progetto. Prendetevi un momento per esplorare il progetto CDK.

Se lo hai Git installato, ogni progetto che crei utilizzando cdk init viene inizializzato anche come repository. Git

Passaggio 2: crea l'app

Nella maggior parte degli ambienti di programmazione, il codice viene creato o compilato dopo aver apportato modifiche. Questa operazione non è necessaria in AWS CDK quanto il CDK CLI eseguirà automaticamente questo passaggio. Tuttavia, puoi comunque creare manualmente se desideri catturare errori di sintassi e di tipo. Di seguito è riportato un esempio:

TypeScript

```
npm run build
```

JavaScript

Non è necessaria alcuna fase di compilazione.

Python

Non è necessaria alcuna fase di costruzione.

Java

```
mvn compile -q
```

Oppure premi Control-B in Eclipse (gli altri IDE Java possono variare)

C#

```
dotnet build src
```

Oppure premi F6 in Visual Studio

Go

```
go build
```

Passaggio 3: Elenca gli stack nell'app

Verifica che l'app sia stata creata correttamente elencando gli stack presenti nell'app. Esegui il seguente codice:

```
cdk ls
```

L'output dovrebbe essere visualizzato`HelloCdkStack`. Se non vedi questo output, verifica di trovarti nella directory di lavoro corretta del progetto e riprova. Se ancora non vedi il tuo stack, ripeti the section called “Passaggio 1: crea l'app” e riprova.

Fase 4: aggiungere un bucket Amazon S3

A questo punto, l'app CDK contiene un unico stack. Successivamente, definirai una risorsa bucket Amazon Simple Storage Service (Amazon S3) all'interno del tuo stack. Per fare ciò, importerai e utilizzerai il costrutto `Bucket` L2 dalla Construct Library. AWS

Modifica la tua app CDK importando il `Bucket` costrutto e definendo la risorsa del bucket Amazon S3. Di seguito è riportato un esempio:

TypeScript

In `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

In `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

In `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

In `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#

In `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

In `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
}
```

Diamo un'occhiata più da vicino al costrutto. `Bucket` Come tutti i costrutti, la `Bucket` classe accetta tre parametri:

- scope — Definisce la `Stack` classe come genitore del `Bucket` costrutto. Tutti i costrutti che definiscono AWS le risorse vengono creati nell'ambito di uno stack. È possibile definire costrutti all'interno dei costrutti, creando una gerarchia (albero). Qui, e nella maggior parte dei casi, l'ambito è `this` (`self`in). Python
- Id: l'ID logico di all'`Bucket`interno AWS CDK dell'app. Questo ID, più un hash basato sulla posizione del bucket all'interno dello stack, identifica in modo univoco il bucket durante la distribuzione. AWS CDK Inoltre, fa riferimento a questo ID quando aggiorni il costrutto nell'app e lo ridistribuisci per aggiornare la risorsa distribuita. Qui, il tuo ID logico è. `MyFirstBucket` I bucket possono anche avere un nome, specificato con la `bucketName` proprietà. È diverso dall'ID logico.
- props: un insieme di valori che definiscono le proprietà del bucket. Qui hai definito la `versioned` proprietà come`true`, che abilita il controllo delle versioni per i file nel bucket.

  Gli oggetti di scena sono rappresentati in modo diverso nelle lingue supportate da. AWS CDK

  - In TypeScript and JavaScript, `props` è un singolo argomento e si passa un oggetto contenente le proprietà desiderate.
  - In Python, gli oggetti di scena vengono passati come argomenti di parole chiave.
  - In Java, viene fornito un Builder per passare gli oggetti di scena. Ce ne sono due: uno per `BucketProps` e un secondo per `Bucket` consentire di creare il costrutto e il relativo oggetto props in un unico passaggio. Questo codice utilizza quest'ultimo.
  - In C#, si crea un'istanza di un `BucketProps` oggetto utilizzando un inizializzatore di oggetti e lo si passa come terzo parametro.

  Se gli oggetti di scena di un costrutto sono opzionali, potete omettere completamente il parametro. `props`

Tutti i costrutti utilizzano gli stessi tre argomenti, quindi è facile rimanere orientati man mano che ne impari di nuovi. E come ci si potrebbe aspettare, è possibile sottoclassare qualsiasi costrutto per estenderlo in base alle proprie esigenze o se si desidera modificarne i valori predefiniti.

Fase 5: Sintetizzare un modello AWS CloudFormation

Sintetizza un AWS CloudFormation modello per l'app, come segue:

```
cdk synth
```

Se l'app contiene più di uno stack, devi specificare quali stack sintetizzare. Poiché l'app contiene un singolo stack, il CDK CLI rileva automaticamente lo stack da sintetizzare.

Se non lo eseguicdk synth, il CDK CLI eseguirà automaticamente questo passaggio al momento della distribuzione. Tuttavia, si consiglia di eseguire questo passaggio prima di ogni distribuzione.

> **Tip**
>
> Se ricevete un errore del tipo: controllate la directory da cui state eseguendo CLI i comandi CDK. `--app is required ...` Dovresti trovarti nella directory principale dell'app.

Il `cdk synth` comando esegue l'app. Questo crea un AWS CloudFormation modello per ogni stack dell'app. Il CDK CLI mostrerà una versione in formato YAML del modello nella riga di comando e salverà una versione in formato JSON del modello nella directory. `cdk.out` Quanto segue è un frammento dell'output della riga di comando che mostra il bucket definito nel modello: AWS CloudFormation

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

> **Note**
>
> Per impostazione predefinita, ogni modello generato contiene una `AWS::CDK::Metadata` risorsa. Il AWS CDK team utilizza questi metadati per ottenere informazioni sull' AWS CDK utilizzo e trovare modi per migliorarlo. Per i dettagli, incluso come disattivare la segnalazione delle versioni, consultaReportistica delle versioni.

Il modello generato può essere distribuito tramite la AWS CloudFormation console o qualsiasi strumento di AWS CloudFormation distribuzione. È inoltre possibile utilizzare il CDK CLI per la distribuzione. Nella fase successiva, si utilizza il CDK per la distribuzione. CLI

Fase 6: Implementa il tuo stack

Per distribuire lo stack CDK AWS CloudFormation utilizzando il CDK, esegui quanto segue: CLI

```
cdk deploy
```

> **Important**
>
> È necessario eseguire un bootstrap una tantum dell'ambiente prima della distribuzione. AWS Per istruzioni, consulta Bootstrap your environment.

Analogamente`cdk synth`, non è necessario specificare lo AWS CDK stack poiché l'app contiene un unico stack.

Se il codice ha implicazioni sulla sicurezza, il CDK CLI genererà un riepilogo. Dovrai confermarli per continuare con la distribuzione. L'app in questo tutorial non ha queste implicazioni.

Dopo l'esecuzione`cdk deploy`, il CDK CLI visualizza le informazioni sullo stato di avanzamento man mano che lo stack viene distribuito. Al termine, puoi accedere alla AWS CloudFormation console per visualizzare lo stack. `HelloCdkStack` Puoi anche accedere alla console Amazon S3 per visualizzare la tua `MyFirstBucket` risorsa.

Complimenti! Hai distribuito il tuo primo stack utilizzando. AWS CDK Successivamente, modificherai la tua app e la ridistribuirai per aggiornare la tua risorsa.

Passaggio 7: modifica l'app

In questo passaggio, modificherai il tuo bucket Amazon S3 configurandolo in modo che venga eliminato automaticamente quando lo stack viene eliminato. Questa modifica comporta la modifica della proprietà del bucket. `RemovalPolicy` Configurerai anche la `autoDeleteObjects` proprietà per configurare il CDK CLI per eliminare oggetti dal bucket prima di distruggerlo. Per impostazione predefinita, AWS CloudFormation non elimina i bucket Amazon S3 che contengono oggetti.

Usa l'esempio seguente per modificare la tua risorsa:

TypeScript

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

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

JavaScript

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

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

Python

Aggiornare `hello_cdk/hello_cdk_stack.py`.

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

Java

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

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

C#

Aggiornare `src/HelloCdk/HelloCdkStack.cs`.

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

Go

Aggiornare `hello-cdk.go`.

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

Al momento, le modifiche al codice non hanno apportato alcun aggiornamento diretto alla risorsa bucket Amazon S3 distribuita. Il codice definisce lo stato desiderato della risorsa. Per modificare la risorsa distribuita, utilizzerai il CDK CLI per sintetizzare lo stato desiderato in un nuovo modello. AWS CloudFormation Quindi, distribuirai il nuovo AWS CloudFormation modello come set di modifiche. I set di modifiche apportano solo le modifiche necessarie per raggiungere il nuovo stato desiderato.

Per vedere queste modifiche, usa il `cdk diff` comando. Esegui il seguente codice:

```
cdk diff
```

Il CDK chiede al CLI tuo Account AWS account il AWS CloudFormation modello più recente per lo stack. `HelloCdkStack` Quindi, confronta il modello più recente con il modello appena sintetizzato dalla tua app. L'output dovrebbe essere simile al seguente.

```
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
```

Questa differenza ha quattro sezioni:

- Modifiche alle dichiarazioni IAM e modifiche alle policy IAM: queste modifiche alle autorizzazioni sono state apportate perché hai impostato la `AutoDeleteObjects` proprietà sul tuo bucket Amazon S3. La funzione di eliminazione automatica utilizza una risorsa personalizzata per eliminare gli oggetti nel bucket prima che il bucket stesso venga eliminato. Gli oggetti IAM garantiscono al codice della risorsa personalizzata l'accesso al bucket.
- Parametri: AWS CDK utilizza queste voci per individuare la risorsa AWS Lambda funzionale per la risorsa personalizzata.
- Risorse: le risorse nuove e modificate di questo stack. Possiamo vedere l'aggiunta degli oggetti IAM menzionati in precedenza, della risorsa personalizzata e della funzione Lambda associata. Possiamo anche vedere che i bucket `DeletionPolicy` e gli `UpdateReplacePolicy` attributi

vengono aggiornati. Questi consentono di eliminare il bucket insieme allo stack e di sostituirlo con uno nuovo.

Potresti notare che abbiamo specificato `RemovalPolicy` nella nostra AWS CDK app ma abbiamo ottenuto una `DeletionPolicy` proprietà nel modello risultante AWS CloudFormation . Questo perché AWS CDK utilizza un nome diverso per la proprietà. L' AWS CDK impostazione predefinita è conservare il bucket quando lo stack viene eliminato, mentre l' AWS CloudFormation impostazione predefinita è eliminarlo. Per ulteriori informazioni, consulta the section called “Politiche di rimozione”.

Per vedere il tuo nuovo AWS CloudFormation modello, puoi eseguire. cdk synth Apportando alcune modifiche all'app CDK, il nuovo AWS CloudFormation modello ora include molte righe di codice aggiuntive rispetto al AWS CloudFormation modello originale.

Quindi, distribuisci la tua app eseguendo quanto segue:

```
cdk deploy
```

Ti AWS CDK informerà sulle modifiche alla politica di sicurezza che abbiamo già visto nel diff. Entra y per approvare le modifiche e distribuire lo stack aggiornato. Il CDK CLI distribuirà lo stack per apportare le modifiche desiderate. Di seguito è riportato un esempio di output:

```
HelloCdkStack: deploying...
[0%] start: Publishing
 4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392:current
[100%] success: Published
 4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392:current
HelloCdkStack: creating CloudFormation changeset...
 0/5 | 4:32:31 PM | UPDATE_IN_PROGRESS   | AWS::CloudFormation::Stack  | HelloCdkStack
 User Initiated
 0/5 | 4:32:36 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092)
 1/5 | 4:32:36 PM | UPDATE_COMPLETE      | AWS::S3::Bucket             | MyFirstBucket
 (MyFirstBucketB8884501)
 1/5 | 4:32:36 PM | CREATE_IN_PROGRESS   | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092) Resource creation
 Initiated
 3/5 | 4:32:54 PM | CREATE_COMPLETE      | AWS::IAM::Role
       | Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 (CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092)
```

```
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::Lambda::Function
      | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD)
 3/5 | 4:32:56 PM | CREATE_IN_PROGRESS   | AWS::Lambda::Function
      | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F) Resource creation
 Initiated
 3/5 | 4:32:57 PM | CREATE_COMPLETE      | AWS::Lambda::Function
      | Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 (CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F)
 3/5 | 4:32:57 PM | CREATE_IN_PROGRESS   | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD) Resource creation Initiated
 4/5 | 4:32:57 PM | CREATE_COMPLETE      | AWS::S3::BucketPolicy       | MyFirstBucket/
Policy (MyFirstBucketPolicy3243DEFD)
 4/5 | 4:32:59 PM | CREATE_IN_PROGRESS   | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E)
 5/5 | 4:33:06 PM | CREATE_IN_PROGRESS   | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E) Resource creation Initiated
 5/5 | 4:33:06 PM | CREATE_COMPLETE      | Custom::S3AutoDeleteObjects
 | MyFirstBucket/AutoDeleteObjectsCustomResource/Default
 (MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E)
 5/5 | 4:33:08 PM | UPDATE_COMPLETE_CLEA | AWS::CloudFormation::Stack  | HelloCdkStack
 6/5 | 4:33:09 PM | UPDATE_COMPLETE      | AWS::CloudFormation::Stack  | HelloCdkStack

 #  HelloCdkStack

Stack ARN:
arn:aws:cloudformation:REGION:ACCOUNT:stack/HelloCdkStack/UNIQUE-ID
```

Fase 8: Distruggere le risorse dell'app

Ora che hai completato questo tutorial, puoi eliminare lo AWS CloudFormation stack distribuito e tutte le risorse ad esso associate. Questa è una buona pratica per ridurre al minimo i costi non necessari e mantenere pulito l'ambiente. Esegui il seguente codice:

```
cdk destroy
```

Entra y per approvare le modifiche ed eliminare lo stack.

> **Note**
>
> Se non modificassi il bucket`RemovalPolicy`, l'eliminazione dello stack verrebbe completata correttamente, ma il bucket diventerebbe orfano (non più associato allo stack).

Passaggi successivi

Complimenti! Hai completato questo tutorial e lo hai utilizzato per creare, modificare ed eliminare correttamente le risorse in. AWS CDK Cloud AWS Ora sei pronto per iniziare a utilizzare il AWS CDK.

Per ulteriori informazioni sull'utilizzo di AWS CDK nel linguaggio di programmazione preferito, consultaLavorare con i AWS CDK linguaggi di programmazione supportati.

Per ulteriori risorse, consulta quanto segue:

- Prova il CDK Workshop per un tour più approfondito che coinvolge un progetto più complesso.
- Approfondisci concetti comethe section called “Ambienti”,the section called “Asset”,, the section called “Autorizzazioni” the section called “Context”the section called “Parametri”, e. the section called “Personalizzazione dei costrutti”
- Consulta il riferimento all'API per iniziare a esplorare i costrutti CDK disponibili per i tuoi servizi preferiti AWS .
- Visita Construct Hub per scoprire i costrutti creati da e altri. AWS
- Esplora esempi di utilizzo di. AWS CDK

AWS CDK È un progetto open source. Per contribuire, vedere Contributing to the AWS Cloud Development Kit (AWS CDK).

Migrazione dalla AWS CDK v1 alla v2 AWS CDK

La versione 2 di AWS Cloud Development Kit (AWS CDK) è progettata per semplificare la scrittura dell'infrastruttura come codice nel linguaggio di programmazione preferito. Questo argomento descrive le modifiche tra la v1 e la v2 di. AWS CDK

> **Tip**
>
> Per identificare gli stack distribuiti con la AWS CDK v1, usa l'utilità awscdk-v1-stack-finder.

Le principali modifiche dalla AWS CDK v1 alla CDK v2 sono le seguenti.

- AWS CDK v2 consolida le parti stabili della AWS Construct Library, inclusa la libreria principale, in un unico pacchetto,. `aws-cdk-lib` Gli sviluppatori non devono più installare pacchetti aggiuntivi per i singoli AWS servizi che utilizzano. Questo approccio a pacchetto singolo significa anche che non è necessario sincronizzare le versioni dei vari pacchetti della libreria CDK.

  I costrutti L1 (CFNxxxx), che rappresentano le esatte risorse disponibili in AWS CloudFormation, sono sempre considerati stabili e quindi sono inclusi. `aws-cdk-lib`
- I moduli sperimentali, in cui stiamo ancora collaborando con la comunità per sviluppare nuovi costrutti L2 o L3, non sono inclusi. `aws-cdk-lib` Vengono invece distribuiti come pacchetti singoli. I pacchetti sperimentali sono denominati con un `alpha` suffisso e un numero di versione semantico. Il numero di versione semantico corrisponde alla prima versione della AWS Construct Library con cui sono compatibili, anch'essa con un suffisso. `alpha` I costrutti entrano in gioco `aws-cdk-lib` dopo essere stati designati come stabili, permettendo alla Construct Library principale di aderire a un rigoroso sistema di versioni semantiche.

  La stabilità è specificata a livello di servizio. Ad esempio, se iniziamo a creare uno o più costrutti L2 per Amazon AppFlow, che al momento della stesura di questo documento ha solo costrutti L1, questi appaiono per la prima volta in un modulo denominato. `@aws-cdk/aws-appflow-alpha` Poi, passano a `aws-cdk-lib` quando riteniamo che i nuovi costrutti soddisfino le esigenze fondamentali dei clienti.

  Una volta che un modulo è stato designato stabile e incorporato`aws-cdk-lib`, vengono aggiunte nuove API utilizzando la convenzione «BetAN» descritta nel prossimo bullet.

Una nuova versione di ogni modulo sperimentale viene rilasciata con ogni versione di. AWS CDK Per la maggior parte, tuttavia, non è necessario mantenerli sincronizzati. Puoi aggiornare `aws-cdk-lib` il modulo sperimentale ogni volta che vuoi. L'eccezione è che quando due o più moduli sperimentali correlati dipendono l'uno dall'altro, devono avere la stessa versione.

- Per i moduli stabili a cui vengono aggiunte nuove funzionalità, le nuove API (che si tratti di costrutti completamente nuovi o di nuovi metodi o proprietà su un costrutto esistente) ricevono un `Beta1` suffisso mentre il lavoro è in corso. (Seguito da `Beta2``Beta3`, e così via quando sono necessarie modifiche sostanziali). Una versione dell'API senza il suffisso viene aggiunta quando l'API è designata stabile. Tutti i metodi tranne il più recente (versione beta o finale) vengono quindi obsoleti.

  Ad esempio, se aggiungiamo un nuovo metodo `grantPower()` a un costrutto, inizialmente appare come. `grantPowerBeta1()` Se sono necessarie modifiche sostanziali (ad esempio, un nuovo parametro o proprietà obbligatorio), verrà denominata `grantPowerBeta2()` la versione successiva del metodo e così via. Quando il lavoro è completo e l'API è finalizzata, viene aggiunto il metodo `grantPower()` (senza suffisso) e i metodi BetAN diventano obsoleti.

  Tutte le API beta rimangono nella Construct Library fino alla prossima versione principale (3.0) e le loro firme non cambieranno. Se le utilizzate, vedrete degli avvisi di obsolescenza, quindi dovreste passare alla versione finale dell'API il prima possibile. Tuttavia, nessuna futura release AWS CDK 2.x interromperà l'applicazione.
- La `Construct` classe è stata estratta da una libreria separata, insieme ai tipi correlati. AWS CDK Questo viene fatto per supportare gli sforzi volti ad applicare il Construct Programming Model ad altri domini. Se state scrivendo i vostri costrutti o utilizzate le API correlate, dovete dichiarare il `constructs` modulo come dipendenza e apportare piccole modifiche alle importazioni. Se utilizzi funzionalità avanzate, come il collegamento al ciclo di vita dell'app CDK, potrebbero essere necessarie ulteriori modifiche. Per tutti i dettagli, consulta la RFC.
- Le proprietà, i metodi e i tipi obsoleti nella AWS CDK v1.x e nella relativa Construct Library sono stati rimossi completamente dall'API CDK v2. Nella maggior parte delle lingue supportate, queste API generano avvisi nella versione 1.x, quindi potresti aver già effettuato la migrazione alle API sostitutive. Un elenco completo delle API obsolete in CDK v1.x è disponibile su. GitHub
- Il comportamento limitato dai flag di funzionalità nella v1.x è abilitato di default in AWS CDK CDK v2. I flag di funzionalità precedenti non sono più necessari e nella maggior parte dei casi non sono supportati. Alcuni sono ancora disponibili per consentire di ripristinare il comportamento di CDK v1 in circostanze molto specifiche. Per ulteriori informazioni, consulta the section called “Aggiornamento dei contrassegni delle funzionalità”.

- Con CDK v2, gli ambienti in cui vengono implementati devono essere avviati utilizzando il moderno stack di bootstrap. Lo stack di bootstrap legacy (predefinito nella versione v1) non è più supportato. CDK v2 richiede inoltre una nuova versione dello stack moderno. Per aggiornare gli ambienti esistenti, riavviali. Non è più necessario impostare alcun flag di funzionalità o variabile di ambiente per utilizzare il moderno stack di bootstrap.

> **Important**
>
> Il moderno modello di bootstrap concede efficacemente le autorizzazioni implicite da a qualsiasi account nell'`--cloudformation-execution-policies`elenco. AWS `--trust` Per impostazione predefinita, questo estende le autorizzazioni di lettura e scrittura a qualsiasi risorsa nell'account avviato. Assicurati di configurare lo stack di bootstrap con politiche e account affidabili con cui ti senti a tuo agio.

Nuovi prerequisiti

La maggior parte dei requisiti per la AWS CDK v2 è la stessa della v1.x. AWS CDK I requisiti aggiuntivi sono elencati qui.

- Per TypeScript gli sviluppatori, è richiesta la versione TypeScript 3.8 o successiva.
- È necessaria una nuova versione del CDK Toolkit per l'uso con CDK v2. Ora che CDK v2 è generalmente disponibile, la v2 è la versione predefinita per l'installazione di CDK Toolkit. È retrocompatibile con i progetti CDK v1, quindi non è necessario mantenere installata la versione precedente a meno che non si desideri creare progetti CDK v1. Per eseguire l'aggiornamento, emetti. `npm install -g aws-cdk`

Aggiornamento dalla versione 2 Developer AWS CDK Preview

Se utilizzi la CDK v2 Developer Preview, il tuo progetto dipende da una versione Release Candidate di, ad esempio. AWS CDK`2.0.0-rc1` Aggiorna questi moduli a`2.0.0`, quindi aggiorna i moduli installati nel progetto.

TypeScript

`npm install` o `yarn install`

JavaScript

`npm install` o `yarn install`

Python

```
python -m pip install -r requirements.txt
```

Java

```
mvn package
```

C#

```
dotnet restore
```

Go

```
go get
```

Dopo aver aggiornato le dipendenze, esegui l'aggiornamento `npm update -g aws-cdk` di CDK Toolkit alla versione di rilascio.

Migrazione dalla AWS CDK v1 alla CDK v2

Per migrare la tua app alla AWS CDK v2, aggiorna prima i flag delle funzionalità in. `cdk.json` Aggiorna quindi le dipendenze e le importazioni dell'app in base alle esigenze del linguaggio di programmazione in cui è scritta.

Aggiornamento a una versione v1 recente

Stiamo assistendo all'aggiornamento da una vecchia versione della AWS CDK v1 alla versione più recente della v2 in un unico passaggio. Anche se è certamente possibile farlo, si comporterebbe sia l'aggiornamento dopo diversi anni di modifiche (che sfortunatamente non sono stati tutti sottoposti alla stessa quantità di test evolutivi che abbiamo oggi), sia l'aggiornamento tra versioni con nuove impostazioni predefinite e un'organizzazione del codice diversa.

Per un'esperienza di aggiornamento più sicura e per diagnosticare più facilmente l'origine di eventuali modifiche impreviste, ti consigliamo di separare questi due passaggi: prima esegui l'aggiornamento alla versione v1 più recente, quindi passa alla v2.

Aggiornamento dei contrassegni delle funzionalità

Rimuovi i seguenti flag di funzionalità della v1, `cdk.json` se esistono, poiché sono tutti attivi per impostazione predefinita nella v2. AWS CDK Se il loro vecchio effetto è importante per l'infrastruttura, sarà necessario apportare modifiche al codice sorgente. Per ulteriori informazioni, consulta l'elenco delle bandiere su GitHub.

- `@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`

È possibile impostare una serie di flag di funzionalità della v1 per `false` ripristinare comportamenti specifici della AWS CDK v1; per un riferimento completo, consulta l'elenco qui the section called “Ripristino del comportamento v1” sotto. GitHub

Per entrambi i tipi di flag, utilizzate il `cdk diff` comando per controllare le modifiche al modello sintetizzato per vedere se le modifiche a uno di questi flag influiscono sulla vostra infrastruttura.

Compatibilità con CDK Toolkit

CDK v2 richiede la versione 2 o successiva del CDK Toolkit. Questa versione è retrocompatibile con le app CDK v1. Pertanto, puoi utilizzare un'unica versione di CDK Toolkit installata a livello globale con tutti i tuoi AWS CDK progetti, indipendentemente dal fatto che utilizzino la v1 o la v2. Un'eccezione è che CDK Toolkit v2 crea solo progetti CDK v2.

Se è necessario creare progetti CDK v1 e v2, non installate CDK Toolkit v2 a livello globale. (Rimuovilo se lo hai già installato:.) `npm remove -g aws-cdk` Per richiamare CDK Toolkit, usate npx to run v1 o v2 del CDK Toolkit come desiderate.

```
npx aws-cdk@1.x init app --language typescript
npx aws-cdk@2.x init app --language typescript
```

Tip

Imposta gli alias della riga di comando in modo da poter utilizzare cdk1 i comandi cdk and per richiamare la versione desiderata di 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 $*
```

Aggiornamento delle dipendenze e delle importazioni

Aggiorna le dipendenze dell'app, quindi installa i nuovi pacchetti. Infine, aggiorna le importazioni nel codice.

TypeScript

Applicazioni

Per le app CDK, esegui l'aggiornamento `package.json` come segue. Rimuovi le dipendenze dai singoli moduli stabili in stile v1 e stabilisci la versione più bassa richiesta per la tua applicazione (2.0.0 qui). `aws-cdk-lib`

I costrutti sperimentali sono forniti in pacchetti separati, con versioni indipendenti, con nomi che terminano con e un numero di versione alfa. `alpha` Il numero di versione alpha corrisponde alla prima versione `aws-cdk-lib` con cui sono compatibili. Qui, abbiamo aggiunto la versione `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"
  }
}
```

Costruisci librerie

Per le librerie di costruzione, stabilisci la versione più bassa richiesta per la tua applicazione (2.0.0 qui) e aggiorna come segue. `aws-cdk-lib` `package.json`

Nota che `aws-cdk-lib` appare sia come dipendenza tra pari che come dipendenza di sviluppo.

```
{
  "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**
>
> Quando si rilascia una libreria compatibile con v2, è necessario eseguire un aumento importante della versione del numero di versione della libreria, poiché si tratta di una modifica fondamentale per gli utenti delle biblioteche. Non è possibile supportare sia CDK v1 che v2 con una singola libreria. Per continuare a supportare i clienti che utilizzano ancora la v1, puoi mantenere la versione precedente in parallelo o creare un nuovo pacchetto per la v2.
>
> Sei tu a decidere per quanto tempo vuoi continuare a supportare i clienti AWS CDK v1. Potresti prendere spunto dal ciclo di vita del CDK v1 stesso, che è entrato in manutenzione il 1° giugno 2022 e terminerà il 1° giugno 2023. end-of-life Per tutti i dettagli, consulta la Politica di manutenzione.AWS CDK

Sia le librerie che le app

Installa le nuove dipendenze eseguendo `npm install` o `yarn install`.

Modificate le importazioni in modo `Construct` da importare dal nuovo `constructs` modulo i tipi principali (ad esempio `App` e `Stack` dal livello superiore di `aws-cdk-lib`) e i moduli stabili di Construct Library per i servizi utilizzati dai namespace sottostanti. `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

Aggiorna come segue. `package.json` Rimuovi le dipendenze dai singoli moduli stabili in stile v1 e stabilisci la versione più bassa richiesta per la tua applicazione (2.0.0 qui). `aws-cdk-lib`

I costrutti sperimentali sono forniti in pacchetti separati, con versioni indipendenti, con nomi che terminano con e un numero di versione alfa. `alpha` Il numero di versione alpha corrisponde alla prima versione `aws-cdk-lib` con cui sono compatibili. Qui, abbiamo aggiunto la versione `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"
  }
}
```

Installa le nuove dipendenze eseguendo o. `npm install` `yarn install`

Modifica le importazioni dell'app per effettuare le seguenti operazioni:

- Importa `Construct` dal nuovo `constructs` modulo
- Importa i tipi principali, come `App` e`Stack`, dal livello superiore di `aws-cdk-lib`
- Importa AWS i moduli di Construct Library dai namespace in `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

Aggiorna `requirements.txt` la `install_requires` definizione come segue. `setup.py` Rimuovi le dipendenze dai singoli moduli stabili in stile v1.

I costrutti sperimentali sono forniti in pacchetti separati, con versioni indipendenti, con nomi che terminano con `alpha` e un numero di versione alfa. Il numero di versione alpha corrisponde alla prima versione `aws-cdk-lib` con cui sono compatibili. Qui, abbiamo bloccato la versione `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**
>
> Disinstalla qualsiasi altra versione dei AWS CDK moduli già installati nell'ambiente virtuale dell'app utilizzando. `pip uninstall` Quindi installa le nuove dipendenze con`python -m pip install -r requirements.txt`.

Modifica le importazioni della tua app per effettuare le seguenti operazioni:

- Importa `Construct` dal nuovo `constructs` modulo
- Importa i tipi principali, come `App` e`Stack`, dal livello superiore di `aws_cdk`
- Importa AWS i moduli di Construct Library dai namespace in `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

Inpom.xml, rimuovi tutte le software.amazon.awscdk dipendenze per i moduli stabili e sostituiscile con dipendenze da (for) e. software.constructs Construct software.amazon.awscdk

I costrutti sperimentali sono forniti in pacchetti separati, con versioni indipendenti, con nomi che terminano con alpha e un numero di versione alfa. Il numero di versione alpha corrisponde alla prima versione aws-cdk-lib con cui sono compatibili. Qui, abbiamo aggiunto la versione 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>
```

Installa le nuove dipendenze eseguendo. mvn package

Modifica il codice per eseguire le seguenti operazioni:

- Importa Construct dalla nuova software.constructs libreria
- Importa le classi principali, ad esempio Stack e App da software.amazon.awscdk
- Importa costrutti di servizio da 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#

Il modo più semplice per aggiornare le dipendenze di un'applicazione C# CDK consiste nel modificare il file manualmente. `.csproj` Rimuovi tutti i riferimenti stabili ai `Amazon.CDK.*` pacchetti e sostituiscili con riferimenti ai pacchetti and. `Amazon.CDK.Lib` `Constructs`

I costrutti sperimentali sono forniti in pacchetti separati, con versioni indipendenti, con nomi che terminano con `alpha` e un numero di versione alfa. Il numero di versione alpha corrisponde alla prima versione `aws-cdk-lib` con cui sono compatibili. Qui, abbiamo aggiunto la versione `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" />
```

Installa le nuove dipendenze eseguendo. `dotnet restore`

Modifica le importazioni nei tuoi file sorgente come segue.

```
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 per aggiornare le dipendenze alla versione più recente e aggiornare il `.mod` file del progetto.

Verifica dell'app migrata prima della distribuzione

Prima di distribuire gli stack, usali per verificare eventuali modifiche impreviste `cdk diff` alle risorse. Non sono previste modifiche agli ID logici (che causano la sostituzione delle risorse).

Le modifiche previste includono, a titolo esemplificativo ma non esaustivo:

- Modifiche alla `CDKMetadata` risorsa.
- Hash degli asset aggiornati.
- Modifiche relative alla sintesi dello stack di nuova concezione. Si applica se l'app utilizzava il sintetizzatore stack legacy nella v1. (CDK v2 non supporta il sintetizzatore stack legacy.)
- L'aggiunta di una regola. `CheckBootstrapVersion`

Le modifiche impreviste in genere non sono causate dall'aggiornamento alla AWS CDK v2 di per sé. Di solito, sono il risultato di un comportamento obsoleto che in precedenza era stato modificato dai flag di funzionalità. Questo è un sintomo dell'aggiornamento da una versione di CDK precedente alla 1.85.x circa. Le stesse modifiche si vedrebbero eseguendo l'aggiornamento all'ultima versione v1.x. Di solito è possibile risolvere questo problema effettuando le seguenti operazioni:

1. Aggiorna la tua app all'ultima versione v1.x
2. Rimuovi i contrassegni delle funzionalità
3. Se necessario, modificate il codice
4. Implementazione
5. Esegui l'aggiornamento alla v2

> **Note**
>
> Se l'app aggiornata non è più utilizzabile dopo l'aggiornamento in due fasi, segnala il problema.

Quando sei pronto per distribuire gli stack nella tua app, valuta la possibilità di distribuirne prima una copia in modo da poterla testare. Il modo più semplice per farlo è distribuirla in un'altra regione. Tuttavia, puoi anche modificare gli ID dei tuoi stack. Dopo il test, assicurati di distruggere la copia di prova concdk destroy.

Risoluzione dei problemi

TypeScript **`'from' expected`**o **`';' expected`** errore nelle importazioni

Aggiornamento alla versione TypeScript 3.8 o successiva.

Esegui 'cdk bootstrap'

Se vedi un errore come il seguente:

```
#  MyStack failed: Error: MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not
 found. Has the environment been bootstrapped? Please run 'cdk bootstrap' (see https://
docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html)
    at CloudFormationDeployments.validateBootstrapStackVersion (.../aws-cdk/lib/api/
cloudformation-deployments.ts:323:13)
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment
 been bootstrapped? Please run 'cdk bootstrap' (see https://docs.aws.amazon.com/cdk/
latest/guide/bootstrapping.html)
```

AWS CDK v2 richiede uno stack di bootstrap aggiornato e, inoltre, tutte le distribuzioni v2 richiedono risorse bootstrap. (Con la v1, puoi distribuire stack semplici senza bootstrap.) Per tutti i dettagli completi, consultare the section called “Bootstrapping (Processo di bootstrap)”.

Trovare gli stack v1

Durante la migrazione dell'applicazione CDK dalla v1 alla v2, potresti voler identificare gli stack distribuiti AWS CloudFormation che sono stati creati utilizzando v1. Per fare ciò, esegui il seguente comando:

```
npx awscdk-v1-stack-finder
```

Per i dettagli sull'utilizzo, consulta il file README di awscdk-v1-stack-finder.

Esegui la migrazione delle risorse e AWS CloudFormation dei modelli esistenti su AWS CDK

La funzionalità CDK Migrate è disponibile in anteprima AWS CDK ed è soggetta a modifiche.

Utilizzate l'interfaccia a riga di AWS Cloud Development Kit (AWS CDK) comando (AWS CDK CLI) per migrare AWS le risorse distribuite, gli AWS CloudFormation stack distribuiti e i modelli locali verso. AWS CloudFormation AWS CDK

Argomenti

Come funziona la migrazione

Utilizzate il AWS CDK CLI `cdk migrate` comando per migrare dalle seguenti fonti:

- Risorse distribuite AWS .
- Stack distribuiti AWS CloudFormation .
- Modelli locali AWS CloudFormation .

Risorse distribuite AWS

È possibile migrare AWS le risorse distribuite da un ambiente specifico (Account AWS e Regione AWS) che non sono associate a uno stack. AWS CloudFormation

AWS CDK CLIUtilizza il servizio di generazione IaC per cercare le risorse nell' AWS ambiente e raccogliere i dettagli delle risorse. Per ulteriori informazioni sul generatore IAc, consulta Generazione di modelli per risorse esistenti nella Guida per l'AWS CloudFormation utente.

Dopo aver raccolto i dettagli delle risorse, AWS CDK CLI crea una nuova app CDK che include un unico stack contenente le risorse migrate.

Pile distribuite AWS CloudFormation

Puoi migrare un singolo AWS CloudFormation stack in una nuova app. AWS CDK AWS CDK CLIRecupererà il AWS CloudFormation modello del tuo stack e creerà una nuova app CDK. L'app CDK sarà composta da un unico stack che contiene lo stack migrato. AWS CloudFormation

Modelli locali AWS CloudFormation

È possibile migrare da un AWS CloudFormation modello locale. I modelli locali possono contenere o meno risorse distribuite. AWS CDK CLICreerà una nuova app CDK che contiene un unico stack con le tue risorse.

Dopo la migrazione, puoi gestire, modificare e distribuire l'app CDK per fornire o aggiornare le tue AWS CloudFormation risorse.

Vantaggi di CDK Migrate

La migrazione delle risorse verso questa AWS CDK destinazione è stata storicamente un processo manuale che richiede tempo ed esperienza e persino per essere avviato. AWS CloudFormation AWS CDK Con CDK Migrate, AWS CDK CLI facilita la maggior parte delle operazioni di migrazione in una frazione del tempo. CDK Migrate ti consentirà di iniziare rapidamente a utilizzare lo strumento AWS CDK per sviluppare e gestire applicazioni nuove ed esistenti su. AWS

Considerazioni

Considerazioni generali

CDK Migrate e CDK Import

Il `cdk import` comando può importare le risorse distribuite in un'app CDK nuova o esistente. Durante l'importazione, ogni risorsa dovrà essere definita manualmente come un costrutto L1 nell'app. Ti consigliamo di `cdk import` utilizzarla per importare una o più risorse alla volta in

un'app CDK nuova o esistente. Per ulteriori informazioni, consulta Importazione di risorse esistenti in uno stack.

Il `cdk migrate` comando migra dalle risorse distribuite, dagli AWS CloudFormation stack distribuiti o dai AWS CloudFormation modelli locali a una nuova app CDK. Durante la migrazione, AWS CDK CLI utilizza `cdk import` per importare le risorse nella nuova app CDK. AWS CDK CLIInoltre genera per voi costrutti L1 per ogni risorsa. Si consiglia di utilizzarlo `cdk migrate` durante l'importazione da una fonte di migrazione supportata in una nuova app. AWS CDK

CDK Migrate crea solo costrutti L1

L'app CDK appena creata includerà solo costrutti L1. Puoi aggiungere costrutti di livello superiore all'app dopo la migrazione.

CDK Migrate crea app CDK che contengono un singolo stack

L'app CDK appena creata conterrà un singolo stack.

Durante la migrazione delle risorse distribuite, tutte le risorse migrate saranno contenute in un unico stack nella nuova app CDK.

Durante la migrazione degli AWS CloudFormation stack, è possibile migrare solo un singolo stack in un unico AWS CloudFormation stack nella nuova app CDK.

Migrazione delle risorse

Le risorse del progetto, come il AWS Lambda codice, non migreranno direttamente nella nuova app CDK. Dopo la migrazione, puoi specificare i valori degli asset per includerli nell'app CDK.

Migrazione di risorse stateful

Quando esegui la migrazione di risorse con stato, come database e bucket Amazon Simple Storage Service (Amazon S3), molto spesso desideri migrare la risorsa esistente anziché crearne una nuova.

Per migrare e preservare le risorse con stato, procedi come segue:

- Verifica che la risorsa stateful supporti l'importazione. Per ulteriori informazioni, consulta il supporto per i tipi di risorse nella Guida per l'AWS CloudFormation utente.
- Dopo la migrazione, verifica che l'ID logico della risorsa migrata nella nuova app CDK corrisponda all'ID logico della risorsa distribuita.
- Se esegui la migrazione da uno AWS CloudFormation stack, verifica che il nome dello stack nella nuova app CDK corrisponda allo stack. AWS CloudFormation
- Implementa l'app CDK utilizzando lo stesso AWS account e la risorsa migrata. Regione AWS

Considerazioni sulla migrazione da un modello AWS CloudFormation

CDK Migrate supporta la migrazione di un singolo modello

Durante la migrazione dei AWS CloudFormation modelli, è possibile selezionare un singolo modello da migrare. I modelli annidati non sono supportati.

Migrazione di modelli con funzioni intrinseche

Durante la migrazione da un AWS CloudFormation modello che utilizza funzioni intrinseche, AWS CDK CLI tenterà di migrare la logica nell'app CDK con la classe. `Fn` Per saperne di più, consulta la classe Fn nell'API Reference.AWS Cloud Development Kit (AWS CDK)

Considerazioni sulla migrazione dalle risorse distribuite

Limitazioni di scansione

Durante la scansione dell'ambiente alla ricerca di risorse, il generatore IaC presenta limitazioni specifiche sui dati che può recuperare e limiti di quota durante la scansione. Per ulteriori informazioni, consulta Considerazioni nella Guida per l'AWS CloudFormation utente.

Prerequisiti

Prima di utilizzare il `cdk migrate` comando, effettuate le seguenti operazioni:

1. Stabilisci l'autenticazione con AWS. Per istruzioni, consulta Fase 2: Configurazione dell'accesso programmatico.
2. Installa o aggiorna il AWS CDK CLI. Per le istruzioni di installazione, consulta Fase 3: Installare AWS CDKCLI.

Inizia a usare CDK Migrate

Per iniziare, esegui il AWS CDK CLI `cdk migrate` comando da una directory a tua scelta. Fornisci le opzioni obbligatorie e facoltative, a seconda del tipo di migrazione che stai eseguendo.

Per un elenco completo e una descrizione delle opzioni utilizzabili`cdk migrate`, consultacdk migrateriferimento al comando.

Di seguito sono riportate alcune opzioni importanti che potresti voler fornire.

Stack name (Nome stack)

L'unica opzione richiesta è`--stack-name`. Utilizzate questa opzione per specificare un nome per lo stack che verrà creato all'interno dell' AWS CDK app dopo la migrazione. Il nome dello stack verrà utilizzato anche come nome dello AWS CloudFormation stack al momento della distribuzione.

Lingua

Viene utilizzato `--language` per specificare il linguaggio di programmazione della nuova app CDK.

AWS account e Regione AWS

AWS CDK CLIrecupera l' AWS account e Regione AWS le informazioni da fonti predefinite. Per ulteriori informazioni, consulta Fase 2: Configurazione dell'accesso programmatico. È possibile utilizzare `--account` e utilizzare `--region` le opzioni `cdk migrate` per fornire altri valori.

Directory di output del nuovo progetto CDK

Per impostazione predefinita, AWS CDK CLI creerà un nuovo progetto CDK nella directory di lavoro e utilizzerà il valore fornito `--stack-name` per denominare la cartella del progetto. Se esiste attualmente una cartella con lo stesso nome, la AWS CDK CLI sovrascriverà.

È possibile specificare un percorso di output diverso per la nuova cartella di progetto CDK con l'`--output-path`opzione.

Fonte di migrazione

Fornisci un'opzione per specificare la fonte da cui stai migrando.

- `--from-path`— Eseguire la migrazione da un modello locale AWS CloudFormation .
- `--from-scan`— Esegui la migrazione dalle risorse distribuite in un AWS account e. Regione AWS
- `--from-stack`— Migrazione da uno stack. AWS CloudFormation

A seconda della fonte di migrazione, è possibile fornire opzioni aggiuntive per personalizzare il `cdk migrate` comando.

Esegui la migrazione da uno stack AWS CloudFormation

Per migrare da uno AWS CloudFormation stack distribuito, offri l'opzione. `--from-stack` Fornisci il nome dello stack distribuito AWS CloudFormation con. `--stack-name` Di seguito è riportato un esempio:

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

AWS CDK CLIFaranno quanto segue:

1. Recupera il AWS CloudFormation modello dello stack distribuito.
2. Esegui `cdk init` per inizializzare una nuova app CDK.
3. Crea uno stack all'interno dell'app CDK che contiene lo stack migrato. AWS CloudFormation

Quando esegui la migrazione da uno AWS CloudFormation stack distribuito, i AWS CDK CLI tentativi di abbinare gli ID logici delle risorse distribuite e il nome dello stack distribuito alle risorse e allo AWS CloudFormation stack migrati si accumulano nella nuova app CDK.

Dopo la migrazione, puoi gestire e modificare l'app CDK normalmente. Al momento della distribuzione, AWS CloudFormation identificherà la distribuzione come un aggiornamento AWS CloudFormation dello stack a causa del nome dello stack corrispondente AWS CloudFormation . Le risorse con ID logici corrispondenti verranno aggiornate. Per ulteriori informazioni sulla distribuzione, vedereGestisci e distribuisci la tua app CDK.

Esegui la migrazione da un modello AWS CloudFormation

CDK Migrate supporta la migrazione da AWS CloudFormation modelli formattati in o. `JSON` `YAML`

Per migrare da un AWS CloudFormation modello locale, utilizzate l'`--from-path`opzione e fornite un percorso al modello locale. È inoltre necessario fornire l'`--stack-name`opzione richiesta. Di seguito è riportato un esempio:

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

AWS CDK CLIFaranno quanto segue:

1. Recupera il tuo AWS CloudFormation modello locale.

2. Esegui `cdk init` per inizializzare una nuova app CDK.
3. Crea uno stack all'interno dell'app CDK che contenga il modello migrato. AWS CloudFormation

Dopo la migrazione, puoi gestire e modificare normalmente l'app CDK. Al momento della distribuzione, sono disponibili le seguenti opzioni:

- Aggiorna uno AWS CloudFormation stack: se il AWS CloudFormation modello locale è stato distribuito in precedenza, puoi aggiornare lo stack distribuito AWS CloudFormation .
- Distribuisci un nuovo AWS CloudFormation stack: se il AWS CloudFormation modello locale non è mai stato distribuito o se desideri creare un nuovo stack da un modello distribuito in precedenza, puoi distribuire un nuovo stack. AWS CloudFormation

Esegui la migrazione da un modello AWS SAM

Per migrare da un modello AWS Serverless Application Model (AWS SAM), devi prima convertirlo in un AWS CloudFormation modello o distribuirlo per creare uno stack. AWS CloudFormation

Per convertire un AWS SAM modello in AWS CloudFormation, puoi usare il comando. AWS SAM CLI `sam validate --debug` Potrebbe essere necessario `lint` impostarlo `false` nel `samconfig.toml` file prima di eseguire questo comando.

Per convertirlo in uno AWS CloudFormation stack, distribuisci il AWS SAM modello utilizzando. AWS SAM CLI Quindi esegui la migrazione dallo stack distribuito.

Esegui la migrazione dalle risorse distribuite

Per migrare dalle AWS risorse distribuite, offri l'opzione. `--from-scan` È inoltre necessario fornire l'opzione richiesta. `--stack-name` Di seguito è riportato un esempio:

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

AWS CDK CLIFaranno quanto segue:

1. Scansiona il tuo account per i dettagli di risorse e proprietà: AWS CDK CLI utilizza il generatore IaC per scansionare il tuo account e raccogliere dettagli.
2. Genera un AWS CloudFormation modello: dopo la scansione, AWS CDK CLI utilizza il generatore IAc per creare un modello. AWS CloudFormation

3. Inizializza una nuova app CDK e migra il tuo modello: AWS CDK CLI esegue `cdk init` l'inizializzazione di una nuova AWS CDK app e migra il AWS CloudFormation modello nell'app CDK come un unico stack.

Usa i filtri

Per impostazione predefinita, AWS CDK CLI eseguirà la scansione dell'intero AWS ambiente e migrerà le risorse fino al limite di quota massimo del generatore IAc. Puoi fornire filtri con lo scopo di specificare un criterio in base AWS CDK CLI al quale le risorse vengono migrate dal tuo account alla nuova app CDK. Per ulteriori informazioni, consulta `--filter`.

Ricerca di risorse con il generatore IAc

A seconda del numero di risorse presenti nell'account, la scansione potrebbe richiedere alcuni minuti. Durante il processo di scansione verrà visualizzata una barra di avanzamento.

Tipi di risorse supportati

AWS CDK CLIMigreranno le risorse supportate dal generatore IAc. Per un elenco completo, consulta il supporto per i tipi di risorse nella Guida per l'AWS CloudFormation utente.

Risolvi le proprietà di sola scrittura

Alcune risorse supportate contengono proprietà di sola scrittura. Queste proprietà possono essere scritte su, per configurare la proprietà, ma non possono essere lette dal generatore IAc o AWS CloudFormation per ottenere il valore. Ad esempio, una proprietà utilizzata per specificare una password di database può essere di sola scrittura per motivi di sicurezza.

Durante la scansione delle risorse durante la migrazione, il generatore IaC rileverà le risorse che possono contenere proprietà di sola scrittura e le classificherà in uno dei seguenti tipi:

- `MUTUALLY_EXCLUSIVE_PROPERTIES`— Si tratta di proprietà di sola scrittura per una risorsa specifica che sono intercambiabili e hanno uno scopo simile. Per configurare la risorsa è necessaria una delle proprietà che si escludono a vicenda. Ad esempio, le proprietà `S3BucketImageUri`, e di una `AWS::Lambda::Function` risorsa sono `ZipFile` proprietà di sola scrittura che si escludono a vicenda. È possibile utilizzare ognuna di esse per specificare le risorse funzionali, ma è necessario utilizzarne una.

- `MUTUALLY_EXCLUSIVE_TYPES`— Si tratta di proprietà di sola scrittura obbligatorie che accettano più tipi di configurazione. Ad esempio, la `Body` proprietà di una `AWS::ApiGateway::RestApi` risorsa accetta un oggetto o un tipo di stringa.
- `UNSUPPORTED_PROPERTIES`— Si tratta di proprietà di sola scrittura che non rientrano nelle altre due categorie. Sono proprietà opzionali o proprietà obbligatorie che accettano una serie di oggetti.

Per ulteriori informazioni sulle proprietà di sola scrittura e su come il generatore IaC le gestisce durante la scansione delle risorse distribuite e la creazione di AWS CloudFormation modelli, consultate il generatore IaC e le proprietà di sola scrittura nella Guida per l'utente.AWS CloudFormation

Dopo la migrazione, è necessario specificare i valori delle proprietà di sola scrittura nella nuova app CDK. AWS CDK CLIAggiungerà una sezione Warnings al `ReadMe` file del progetto CDK per documentare tutte le proprietà di sola scrittura identificate dal generatore IAc. Di seguito è riportato un esempio:

```
# 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
```

- Gli avvisi sono organizzati in titoli che identificano l'ID logico della risorsa a cui sono associati.
- Gli avvisi sono suddivisi in categorie per tipo. Questi tipi provengono direttamente dal generatore IAc.

Per risolvere le proprietà di sola scrittura

1. Identifica le proprietà di sola scrittura da risolvere nella sezione Avvertenze del file del tuo progetto CDK. `ReadMe` Qui puoi prendere nota delle risorse dell'app CDK che possono contenere proprietà di sola scrittura e identificare i tipi di proprietà di sola scrittura che sono stati scoperti.

   a. Ad esempio`MUTUALLY_EXCLUSIVE_PROPERTIES`, stabilite quale proprietà mutuamente esclusiva configurare nella vostra app. AWS CDK

   b. Per`MUTUALLY_EXCLUSIVE_TYPES`, determina quale tipo accettato utilizzerai per configurare la proprietà.

   c. Per`UNSUPPORTED_PROPERTIES`, determina se la proprietà è facoltativa o obbligatoria. Quindi, configurate se necessario.
2. Utilizzate le indicazioni del generatore IAc e delle proprietà di sola scrittura per fare riferimento al significato dei tipi di avviso.
3. Nell'app CDK, i valori delle proprietà di sola scrittura da risolvere verranno specificati anche nella sezione dell'app. `Props` Fornisci qui i valori corretti. Per le descrizioni e le indicazioni sulle proprietà, puoi fare riferimento all'AWS CDK API Reference.

   Di seguito è riportato un esempio della `Props` sezione all'interno di un'app CDK migrata con due proprietà di sola scrittura da risolvere:

```
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;
}
```

Una volta risolti tutti i valori delle proprietà di sola scrittura, sei pronto per prepararti alla distribuzione.

Il file migrate.json

Quindi AWS CDK CLI crea un `migrate.json` file nel tuo AWS CDK progetto durante la migrazione. Questo file contiene informazioni di riferimento sulle risorse distribuite. Quando si distribuisce l'app CDK per la prima volta, AWS CDK CLI utilizza questo file per fare riferimento alle risorse distribuite, le associa al nuovo AWS CloudFormation stack ed elimina il file.

Gestisci e distribuisci la tua app CDK

Durante la migrazione a AWS CDK, la nuova app CDK potrebbe non essere immediatamente pronta per l'implementazione. Questo argomento descrive le azioni da prendere in considerazione durante la gestione e la distribuzione della nuova app CDK.

Preparati per la distribuzione

Prima della distribuzione, devi preparare l'app CDK.

Sintetizza la tua app

Usa il `cdk synth` comando per sintetizzare lo stack dell'app CDK in un modello. AWS CloudFormation

Se hai eseguito la migrazione da uno AWS CloudFormation stack o da un modello distribuito, puoi confrontare il modello sintetizzato con il modello migrato per verificare i valori delle risorse e delle proprietà.

Per ulteriori informazioni su `cdk synth`, consulta Sintetizzazione degli stack.

Esegui un diff

Se hai eseguito la migrazione da uno AWS CloudFormation stack distribuito, puoi usare il comando cdk diff per eseguire il confronto con lo stack della tua nuova app CDK.

Per ulteriori informazioni su cdk diff, consulta. Confronto degli stack

Avvia il tuo ambiente

Se esegui la distribuzione da un AWS ambiente per la prima volta, usalo `cdk bootstrap` per preparare l'ambiente. Per ulteriori informazioni, consulta Bootstrapping (Processo di bootstrap).

Implementa la tua app CDK

Quando distribuisci un'app CDK, AWS CDK CLI utilizza il AWS CloudFormation servizio per fornire le tue risorse. Le risorse sono raggruppate in un unico stack nell'app CDK e vengono distribuite come un unico stack. AWS CloudFormation

A seconda della provenienza da cui è stata effettuata la migrazione, è possibile eseguire l'implementazione per creare un nuovo AWS CloudFormation stack o aggiornare uno stack esistente. AWS CloudFormation

Esegui la distribuzione per creare un nuovo stack AWS CloudFormation

Se hai eseguito la migrazione da risorse distribuite, AWS CDK CLI creerà automaticamente un nuovo AWS CloudFormation stack al momento della distribuzione. Le risorse distribuite verranno incluse nel nuovo stack. AWS CloudFormation

Se hai eseguito la migrazione da un AWS CloudFormation modello locale che non è mai stato distribuito, AWS CDK CLI creerà automaticamente un nuovo AWS CloudFormation stack al momento della distribuzione.

Se hai eseguito la migrazione da uno AWS CloudFormation stack distribuito o da un AWS CloudFormation modello locale distribuito in precedenza, puoi eseguire la distribuzione per creare un nuovo stack. AWS CloudFormation Per creare un nuovo stack, procedi come segue:

- Esegui la distribuzione in un nuovo AWS ambiente. Consiste nell'utilizzo di un AWS account diverso o nella distribuzione in un altro. Regione AWS
- Se desideri distribuire un nuovo stack nello stesso AWS ambiente dello stack o del modello migrato, devi modificare il nome dello stack nell'app CDK con un nuovo valore. È inoltre necessario modificare tutti gli ID logici delle risorse nell'app CDK. Quindi, puoi eseguire la distribuzione nello stesso ambiente per creare un nuovo stack e nuove risorse.

Esegui la distribuzione per aggiornare uno stack esistente AWS CloudFormation

Se hai eseguito la migrazione da uno AWS CloudFormation stack distribuito o da un AWS CloudFormation modello locale distribuito in precedenza, puoi eseguire la distribuzione per aggiornare lo stack esistente. AWS CloudFormation

Verifica che il nome dello stack nell'app CDK corrisponda al nome dello stack distribuito ed esegui la distribuzione nello stesso ambiente. AWS CloudFormation AWS

Lavorare con i AWS CDK linguaggi di programmazione supportati

Usa il AWS Cloud Development Kit (AWS CDK) per definire la tua Cloud AWS infrastruttura con un linguaggio di programmazione supportato.

Argomenti

Importazione della libreria AWS Construct

AWS CDK Include la AWS Construct Library, una raccolta di costrutti organizzati per servizio. AWS I costrutti stabili della libreria sono offerti in un singolo modulo, chiamato con il nome TypeScript del pacchetto:. `aws-cdk-lib` Il nome effettivo del pacchetto varia in base alla lingua.

TypeScript

Installa	`installazione di npm aws-cdk-lib`
Importa	`const cdk = require (''); aws-cdk-lib`

JavaScript

Installa	`installazione di npm aws-cdk-lib`

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

Python

Installa	`installazione python -m pip aws-cdk-lib`
Importa	`importa aws_cdk come cdk`

Java

Aggiungi a `pom.xml`	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Importa	`importare software.amazon.awscdk.app;` (for example)

C#

Installa	`dotnet aggiunge il pacchetto Amazon.cdk.lib`
Importa	`utilizzando Amazon.cdk;`

La classe `construct` base e il codice di supporto si trovano nel modulo. `constructs` I costrutti sperimentali, in cui l'API è ancora in fase di perfezionamento, vengono distribuiti come moduli separati.

Il riferimento all'API AWS CDK

L'AWS CDK API Reference fornisce una documentazione dettagliata dei costrutti (e di altri componenti) della libreria. Viene fornita una versione dell'API Reference per ogni linguaggio di programmazione supportato.

Il materiale di riferimento di ogni modulo è suddiviso nelle seguenti sezioni.

- Panoramica: materiale introduttivo necessario conoscere per utilizzare il servizio nel AWS CDK, inclusi concetti ed esempi.
- Costrutti: classi di libreria che rappresentano una o più risorse concrete. AWS Queste sono le risorse o i modelli «curati» (L2) (risorse L3) che forniscono un'interfaccia di alto livello con impostazioni predefinite valide.
- Classi: classi non costruite che forniscono funzionalità utilizzate dai costrutti del modulo.
- Strutture: strutture di dati (pacchetti di attributi) che definiscono la struttura di valori compositi come proprietà (l'`props`argomento dei costrutti) e opzioni.
- Interfacce: le interfacce, i cui nomi iniziano tutti con «I», definiscono la funzionalità minima assoluta per il costrutto o altra classe corrispondente. Il CDK utilizza interfacce di costruzione per rappresentare AWS risorse definite all'esterno dell' AWS CDK app e referenziate da metodi come. `Bucket.fromBucketArn()`
- Enumerazioni: raccolte di valori denominati da utilizzare per specificare determinati parametri di costruzione. L'utilizzo di un valore enumerato consente al CDK di verificare la validità di questi valori durante la sintesi.
- CloudFormation Risorse: questi costrutti L1, i cui nomi iniziano con «Cfn», rappresentano esattamente le risorse definite nelle specifiche. CloudFormation Vengono generati automaticamente in base a tale specifica con ogni versione CDK. Ogni costrutto L2 o L3 incapsula una o più risorse. CloudFormation
- CloudFormation Tipi di proprietà: la raccolta di valori denominati che definiscono le proprietà di ogni risorsa. CloudFormation

Interfacce confrontate con le classi di costruzione

AWS CDK Utilizza le interfacce in un modo specifico che potrebbe non essere ovvio anche se si ha familiarità con le interfacce come concetto di programmazione.

AWS CDK Supporta l'utilizzo di risorse definite all'esterno delle applicazioni CDK utilizzando metodi come. `Bucket.fromBucketArn()` Le risorse esterne non possono essere modificate e potrebbero non avere tutte le funzionalità disponibili con le risorse definite nell'app CDK utilizzando, ad esempio, la `Bucket` classe. Le interfacce, quindi, rappresentano la funzionalità minima disponibile nel CDK per un determinato tipo di AWS risorsa, incluse le risorse esterne.

Quando si istanziano le risorse nell'app CDK, quindi, è necessario utilizzare sempre classi concrete come. `Bucket` Quando specificate il tipo di argomento che accettate in uno dei vostri costrutti, usate il tipo di interfaccia, ad esempio `IBucket` se siete pronti a gestire risorse esterne (ovvero, non avrete bisogno di cambiarle). Se avete bisogno di un costrutto definito da CDK, specificate il tipo più generale che potete usare.

Alcune interfacce sono versioni minime di pacchetti di proprietà o di opzioni associati a classi specifiche, anziché costrutti. Tali interfacce possono essere utili quando si crea una sottoclasse per accettare argomenti da passare alla classe principale. Se avete bisogno di una o più proprietà aggiuntive, vi consigliamo di implementarle o derivarle da questa interfaccia o da un tipo più specifico.

> **Note**
>
> Alcuni linguaggi di programmazione supportati da AWS CDK non dispongono di una funzionalità di interfaccia. In questi linguaggi, le interfacce sono solo classi ordinarie. È possibile identificarli con i loro nomi, che seguono lo schema di una «I» iniziale seguita dal nome di qualche altro costrutto (ad esempio`IBucket`). Valgono le stesse regole.

Gestire le dipendenze

Le dipendenze per AWS CDK l'app o la libreria vengono gestite utilizzando strumenti di gestione dei pacchetti. Questi strumenti sono comunemente usati con i linguaggi di programmazione.

In genere, AWS CDK supporta lo strumento di gestione dei pacchetti standard o ufficiale del linguaggio, se presente. Altrimenti, AWS CDK supporterà la lingua più popolare o ampiamente supportata. Potresti anche essere in grado di utilizzare altri strumenti, specialmente se funzionano con gli strumenti supportati. Tuttavia, il supporto ufficiale per altri strumenti è limitato.

AWS CDK Supporta i seguenti gestori di pacchetti:

Lingua	Strumento di gestione dei pacchetti supportato
TypeScript/JavaScript	NPM (Node Package Manager) o Yarn
Python	PIP (Package Installer per Python)
Java	Maven

Lingua	Strumento di gestione dei pacchetti supportato
C#	NuGet
Go	Moduli Go

Quando crei un nuovo progetto utilizzando il AWS CDK CLI `cdk init` comando, le dipendenze per le librerie principali del CDK e i costrutti stabili vengono specificate automaticamente.

Per ulteriori informazioni sulla gestione delle dipendenze per i linguaggi di programmazione supportati, consulta quanto segue:

- Gestione delle dipendenze in TypeScript.
- Gestione delle dipendenze in JavaScript.
- Gestione delle dipendenze in Python.
- Gestione delle dipendenze in Java.
- Gestione delle dipendenze in C#.
- Gestione delle dipendenze in Go.

Confronto AWS CDKTypeScript con altri linguaggi

TypeScript è stata la prima lingua supportata per lo sviluppo di AWS CDK applicazioni. Pertanto, è scritta una notevole quantità di codice CDK di esempio. TypeScript Se state sviluppando in un'altra lingua, potrebbe essere utile confrontare il modo in cui il AWS CDK codice viene implementato TypeScript rispetto al linguaggio prescelto. Questo può aiutarvi a utilizzare gli esempi in tutta la documentazione.

Importazione di un modulo

TypeScript/JavaScript

TypeScript supporta l'importazione di un intero namespace o di singoli oggetti da un namespace. Ogni spazio dei nomi include costrutti e altre classi da utilizzare con un determinato servizio. 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

Ad esempio TypeScript, Python supporta le importazioni di moduli con namespace e le importazioni selettive. I namespace in Python assomigliano a aws_cdk. xxx, dove xxx rappresenta il nome AWS di un servizio, ad esempio s3 per Amazon S3. (Amazon S3 viene utilizzato in questi esempi).

```
# 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

Le importazioni di Java funzionano in modo diverso da quelle TypeScript di Java. Ogni istruzione di importazione importa un singolo nome di classe da un determinato pacchetto o tutte le classi definite in quel pacchetto (utilizzando*). È possibile accedere alle classi utilizzando il nome della classe stessa, se è stata importata, o il nome della classe qualificata che include il relativo pacchetto.

Le librerie hanno lo `software.amazon.awscdk.services.xxx` stesso nome della AWS Construct Library (la libreria principale è`software.amazon.awscdk`). L'ID del gruppo Maven per AWS CDK i pacchetti è. `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#

In C#, si importano i tipi con la direttiva. `using` Esistono due stili. Uno ti dà accesso a tutti i tipi nello spazio dei nomi specificato usando i loro nomi semplici. Con l'altro, puoi fare riferimento allo spazio dei nomi stesso usando un alias.

I pacchetti hanno lo stesso nome dei pacchetti `Amazon.CDK.AWS.xxx` AWS Construct Library. (Il modulo principale è`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

Ogni modulo AWS Construct Library viene fornito come pacchetto 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(...)
```

Istanziazione di un costrutto

AWS CDK le classi construct hanno lo stesso nome in tutte le lingue supportate. La maggior parte dei linguaggi usa la `new` parola chiave per istanziare una classe (Python e Go no). Inoltre, nella maggior parte delle lingue, la parola chiave `this` si riferisce all'istanza corrente. (Python usa per `self` convenzione.) Dovresti passare un riferimento all'istanza corrente come `scope` parametro per ogni costrutto che crei.

Il terzo argomento di un AWS CDK costrutto è `props` un oggetto contenente gli attributi necessari per creare il costrutto. Questo argomento può essere facoltativo, ma quando è richiesto, le lingue supportate lo gestiscono in modo idiomatico. I nomi degli attributi vengono inoltre adattati ai modelli di denominazione standard del linguaggio.

TypeScript/JavaScript

```
// Instantiate default Bucket
const bucket = new s3.Bucket(this, 'MyBucket');

// Instantiate Bucket with bucketName and versioned properties
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket',
   versioned: true,
});

// Instantiate Bucket with websiteRedirect, which has its own sub-properties
const bucket = new s3.Bucket(this, 'MyBucket', {
  websiteRedirect: {host: 'aws.amazon.com'}});
```

Python

Python non usa una `new` parola chiave per creare un'istanza di una classe. L'argomento delle proprietà è rappresentato utilizzando argomenti di parole chiave e gli argomenti sono denominati utilizzando. `snake_case`

Se un valore props è di per sé un insieme di attributi, è rappresentato da una classe che prende il nome dalla proprietà, che accetta argomenti di parole chiave per le sottoproprietà.

In Python, l'istanza corrente viene passata ai metodi come primo argomento, denominato `self` per convenzione.

```
# 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

In Java, l'argomento props è rappresentato da una classe denominata `XxxxProps` (ad esempio, `BucketProps` per gli oggetti di scena del `Bucket` costrutto). L'argomento props viene creato utilizzando un pattern builder.

Ogni `XxxxProps` classe ha un builder. C'è anche un comodo generatore per ogni costrutto che costruisce gli oggetti di scena e il costrutto in un unico passaggio, come mostrato nell'esempio seguente.

Gli oggetti di scena hanno lo stesso nome di, using. TypeScript `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#

In C#, gli oggetti di scena vengono specificati utilizzando un inizializzatore di oggetti in una classe denominata `XxxxProps` (ad esempio, `BucketProps` per gli oggetti di scena del `Bucket` costrutto).

Gli oggetti di scena sono denominati in modo simile a, tranne using. TypeScript `PascalCase`

È comodo usare la `var` parola chiave quando si crea un'istanza di un costrutto, quindi non è necessario digitare il nome della classe due volte. Tuttavia, la guida allo stile del codice locale può variare.

```
// 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

Per creare un costrutto in Go, chiama la funzione `NewXxxxxx` where `Xxxxxxx` è il nome del costrutto. Le proprietà dei costrutti sono definite come una struttura.

In Go, tutti i parametri di costruzione sono puntatori, inclusi valori come numeri, valori booleani e stringhe. Usa le comode funzioni come `jsii.String` per creare questi puntatori.

```
 // 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"),
 }})
```

Accesso ai membri

È comune fare riferimento ad attributi o proprietà di costrutti e altre AWS CDK classi e utilizzare questi valori come, ad esempio, input per creare altri costrutti. Le differenze di denominazione descritte in precedenza per i metodi si applicano anche qui. Inoltre, in Java, non è possibile accedere direttamente ai membri. Viene invece fornito un metodo getter.

TypeScript/JavaScript

I nomi sono `camelCase`.

```
bucket.bucketArn
```

Python

I nomi sono `snake_case`.

```
bucket.bucket_arn
```

Java

Per ogni proprietà viene fornito un metodo getter; questi nomi sono `camelCase`.

```
bucket.getBucketArn()
```

C#

I nomi sono`PascalCase`.

```
bucket.BucketArn
```

Go

I nomi sono`PascalCase`.

```
bucket.BucketArn
```

Costanti Enum

Le costanti Enum rientrano nell'ambito di una classe e hanno nomi in maiuscolo con caratteri di sottolineatura in tutte le lingue (a volte denominate). `SCREAMING_SNAKE_CASE` Poiché i nomi delle classi utilizzano lo stesso maiuscolo e minuscolo in tutte le lingue supportate tranne Go, anche i nomi enum qualificati sono gli stessi in queste lingue.

```
s3.BucketEncryption.KMS_MANAGED
```

In Go, le costanti enum sono attributi dello spazio dei nomi del modulo e sono scritte come segue.

```
awss3.BucketEncryption_KMS_MANAGED
```

Interfacce a oggetti

AWS CDK Utilizza interfacce a TypeScript oggetti per indicare che una classe implementa un insieme previsto di metodi e proprietà. È possibile riconoscere un'interfaccia a oggetti perché il suo nome inizia con. `I` Una classe concreta indica le interfacce che implementa utilizzando la `implements` parola chiave.

TypeScript/JavaScript

> **Note**
>
> JavaScript non ha una funzionalità di interfaccia. È possibile ignorare la `implements` parola chiave e i nomi delle classi che la seguono.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';

class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

Python non ha una funzionalità di interfaccia. Tuttavia, AWS CDK puoi indicare l'implementazione dell'interfaccia decorando la tua classe con. `@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

Le strutture Go non hanno bisogno di dichiarare esplicitamente quali interfacce implementano. Il compilatore Go determina l'implementazione in base ai metodi e alle proprietà disponibili nella struttura. Ad esempio, nel codice seguente, `MyAspect` implementa l'`IAspect`interfaccia perché fornisce un `Visit` metodo che accetta un costrutto.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
 fmt.Println("Visited", *node.Node().Path())
}
```

Lavorare con l' AWS CDK interno TypeScript

TypeScript è un linguaggio client completamente supportato per AWS Cloud Development Kit (AWS CDK) ed è considerato stabile. L'utilizzo dell' AWS CDK in TypeScript utilizza strumenti familiari, tra cui il TypeScript compilatore di Microsoft (`tsc`), Node.js e il Node Package Manager (`npm`). Puoi anche usare Yarn se preferisci, sebbene gli esempi in questa guida utilizzino NPM. I moduli che compongono la AWS Construct Library sono distribuiti tramite il repository NPM, npmjs.org.

Puoi usare qualsiasi editor o IDE. Molti AWS CDK sviluppatori utilizzano Visual Studio Code (o il suo equivalente open source VSCodium), che offre un supporto eccellente per. TypeScript

Argomenti

- Inizia con TypeScript
- Creare un progetto
- Utilizzo di locali e tsccdk
- Gestione dei moduli Construct Library AWS
- Gestione delle dipendenze in TypeScript
- AWS CDK idiomi in TypeScript

- Creazione, sintesi e distribuzione

Inizia con TypeScript

Per utilizzare AWS CDK, è necessario disporre di un AWS account e delle credenziali e aver installato Node.js e il AWS CDK Toolkit. Per informazioni, consulta Iniziare con AWS CDK.

È inoltre necessario disporre di TypeScript se stesso (versione 3.8 o successiva). Se non lo possiedi già, puoi installarlo utilizzandonpm.

```
npm install -g typescript
```

> **Note**
>
> Se ricevi un errore di autorizzazione e disponi dell'accesso come amministratore sul tuo sistema, provasudo npm install -g typescript.

TypeScript Tieniti aggiornato con un appuntamento abitualenpm update -g typescript.

> **Note**
>
> Lingua obsoleta di terze parti: la versione linguistica è supportata solo fino alla fine del ciclo di vita (EOL (End Of Life) condivisa dal fornitore o dalla community ed è soggetta a modifiche con preavviso.

Creare un progetto

È possibile creare un nuovo AWS CDK progetto cdk init richiamandolo in una directory vuota. Utilizzate l'--languageopzione e specificatetypescript:

```
mkdir my-project
cd my-project
cdk init app --language typescript
```

La creazione di un progetto installa anche il aws-cdk-libmodulo e le sue dipendenze.

`cdk init`utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. I trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Tuttavia, il nome dovrebbe altrimenti assumere la forma di un TypeScript identificatore; ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Utilizzo di locali e **`tsccdk`**

Per la maggior parte, questa guida presuppone l'installazione TypeScript e l'installazione di CDK Toolkit a livello globale (`npm install -g typescript aws-cdk`), e gli esempi di comandi forniti (ad esempio`cdk synth`) seguano questo presupposto. Questo approccio semplifica l'aggiornamento di entrambi i componenti e, poiché entrambi adottano un approccio rigoroso alla compatibilità con le versioni precedenti, in genere il rischio di utilizzare sempre le versioni più recenti è minimo.

Alcuni team preferiscono specificare tutte le dipendenze all'interno di ciascun progetto, inclusi strumenti come il TypeScript compilatore e il CDK Toolkit. Questa pratica consente di associare questi componenti a versioni specifiche e di garantire che tutti gli sviluppatori del team (e l'ambiente CI/CD) utilizzino esattamente quelle versioni. Ciò elimina una possibile fonte di cambiamento, contribuendo a rendere le build e le implementazioni più coerenti e ripetibili.

Il CDK include le dipendenze per entrambi TypeScript e il CDK Toolkit nei modelli di TypeScript progetto`package.json`, quindi se si desidera utilizzare questo approccio, non è necessario apportare alcuna modifica al progetto. Tutto quello che dovete fare è usare comandi leggermente diversi per creare l'app e per emettere comandi. `cdk`

Operazione	Usa strumenti globali	Usa strumenti locali
Inizializza il progetto	`cdk init --language typescript`	`npx aws-cdk init --language typescript`
Creazione	`tsc`	`npm run build`
Esegui il comando CDK Toolkit	`cdk ...`	`npm run cdk ...` or `npx aws-cdk ...`

`npx aws-cdk`esegue la versione di CDK Toolkit installata localmente nel progetto corrente, se ne esiste una, ricorrendo all'installazione globale, se presente. Se non esiste un'installazione globale, `npx` scarica una copia temporanea di CDK Toolkit e la esegue. È possibile specificare una versione arbitraria di CDK Toolkit utilizzando la sintassi: prints. `@` `npx aws-cdk@2.0 --version` `2.0.0`

> **Tip**
>
> Imposta un alias in modo da poter utilizzare il `cdk` comando con un'installazione locale di CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Gestione dei moduli Construct Library AWS

Utilizzate Node Package Manager (`npm`) per installare e aggiornare i moduli AWS Construct Library per l'uso da parte delle vostre app e degli altri pacchetti di cui avete bisogno. (Puoi usare `yarn` al posto di `npm` se preferisci.) `npm`installa inoltre automaticamente le dipendenze per quei moduli.

La maggior parte dei AWS CDK costrutti si trova nel pacchetto CDK principale, denominato`aws-cdk-lib`, che è una dipendenza predefinita nei nuovi progetti creati da. `cdk init` I moduli «sperimentali» di AWS Construct Library, in cui i costrutti di livello superiore sono ancora in fase di sviluppo, hanno lo stesso nome. `@aws-cdk/`*`SERVICE-NAME`*`-alpha` Il nome del servizio ha un prefisso aws-. Se non sei sicuro del nome di un modulo, cercalo su NPM.

> **Note**
>
> Il CDK API Reference mostra anche i nomi dei pacchetti.

Ad esempio, il comando seguente installa il modulo sperimentale per. AWS CodeStar

```
npm install @aws-cdk/aws-codestar-alpha
```

Il supporto di Construct Library di alcuni servizi è disponibile in più di un namespace. Ad esempio`aws-route53`, ci sono inoltre tre namespace Amazon Route 53 aggiuntivi,`aws-route53-targets`, `aws-route53-patterns` e. `aws-route53resolver`

Le dipendenze del progetto vengono mantenute in. `package.json` Puoi modificare questo file per bloccare alcune o tutte le tue dipendenze su una versione specifica o per consentire loro di essere aggiornate a versioni più recenti in base a determinati criteri. Per aggiornare le dipendenze NPM del progetto all'ultima versione consentita in base alle regole specificate in: `package.json`

```
npm update
```

In TypeScript, importi i moduli nel tuo codice con lo stesso nome che usi per installarli usando NPM. Consigliamo le seguenti pratiche per importare AWS CDK classi e moduli AWS Construct Library nelle applicazioni. Seguire queste linee guida contribuirà a rendere il codice coerente con altre AWS CDK applicazioni e più facile da comprendere.

- Usa `import` direttive in stile ES6, no. `require()`
- In genere, importa singole classi da. `aws-cdk-lib`

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

- Se hai bisogno di molte classi da`aws-cdk-lib`, puoi usare un alias dello spazio dei nomi di `cdk` invece di importare le singole classi. Evita di fare entrambe le cose.

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

- In genere, importate i costrutti dei AWS servizi utilizzando alias di namespace brevi.

  ```
  import { aws_s3 as s3 } from 'aws-cdk-lib';
  ```

Gestione delle dipendenze in TypeScript

Nei progetti TypeScript CDK, le dipendenze sono specificate nel `package.json` file nella directory principale del progetto. I AWS CDK moduli principali si trovano in un unico NPM pacchetto chiamato. `aws-cdk-lib`

Quando installate un pacchetto utilizzandonpm install, NPM registra il pacchetto al posto `package.json` vostro.

Se preferisci, puoi usare Yarn al posto di NPM. Tuttavia, il CDK non supporta la modalità di Yarn, che è la plug-and-play modalità predefinita in Yarn 2. Aggiungi quanto segue al `.yarnrc.yml` file del tuo progetto per disattivare questa funzionalità.

```
nodeLinker: node-modules
```

Applicazioni CDK

Di seguito è riportato un `package.json` file di esempio generato dal `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"
  }
}
```

Per le app CDK distribuibili, `aws-cdk-lib` deve essere specificato nella sezione di`dependencies`. `package.json` Potete utilizzare un identificatore del numero di versione con cursore (^) per indicare che accetterete versioni successive a quella specificata purché si trovino all'interno della stessa versione principale.

Per i costrutti sperimentali, specificate le versioni esatte per i moduli della libreria alpha construct, che hanno API che possono cambiare. Non utilizzate ^ o ~ poiché le versioni successive di questi moduli potrebbero apportare modifiche all'API che potrebbero danneggiare l'app.

Specificate le versioni delle librerie e degli strumenti necessari per testare l'app (ad esempio, il framework di `jest` test) nella `devDependencies` sezione di`package.json`. Facoltativamente, usa ^ per specificare che le versioni compatibili successive sono accettabili.

Librerie di costruzioni di terze parti

Se state sviluppando una libreria di costrutti, specificate le sue dipendenze utilizzando una combinazione delle `devDependencies` sezioni `peerDependencies` e, come mostrato nel seguente file di esempio. `package.json`

```
{
  "name": "my-package",
  "version": "0.0.1",
  "peerDependencies": {
    "aws-cdk-lib": "^2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "^10.0.0"
  },
  "devDependencies": {
    "aws-cdk-lib": "2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "10.0.0",
    "jsii": "^1.50.0",
    "aws-cdk": "^2.14.0"
  }
}
```

In`peerDependencies`, usa un accento circonflesso (^) per specificare la versione più bassa con `aws-cdk-lib` cui funziona la tua libreria. Ciò massimizza la compatibilità della tua libreria con una gamma di versioni CDK. Specificate le versioni esatte per i moduli della libreria alpha construct, che hanno API che possono cambiare. L'utilizzo `peerDependencies` assicura che vi sia una sola copia di tutte le librerie CDK nell'albero. `node_modules`

Nel`devDependencies`, specificate gli strumenti e le librerie necessari per il test, opzionalmente con ^ per indicare che le versioni compatibili successive sono accettabili. Specificate esattamente (senza ^ o ~) le versioni più basse `aws-cdk-lib` e gli altri pacchetti CDK con cui pubblicizzate la

compatibilità della vostra libreria. Questa pratica assicura che i test vengano eseguiti su tali versioni. In questo modo, se usi inavvertitamente una funzionalità presente solo nelle versioni più recenti, i tuoi test possono catturarla.

> **Warning**
>
> `peerDependencies`vengono installati automaticamente solo da NPM 7 e versioni successive. Se stai usando NPM 6 o versioni precedenti, o se stai usando Yarn, devi includere le dipendenze delle tue dipendenze in. `devDependencies` Altrimenti, non verranno installate e riceverai un avviso sulle dipendenze tra pari non risolte.

Installazione e aggiornamento delle dipendenze

Esegui il comando seguente per installare le dipendenze del tuo progetto.

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
```

Per aggiornare i moduli installati, è possibile utilizzare yarn upgrade i comandi precedenti npm install e. Entrambi i comandi aggiornano i pacchetti `node_modules` alle versioni più recenti che soddisfano le regole in`package.json`. Tuttavia, non `package.json` si aggiornano da soli, cosa che potresti voler fare per impostare una nuova versione minima. Se ospiti il pacchetto su GitHub, puoi configurare gli aggiornamenti della versione di Dependabot in modo che si aggiornino automaticamente. `package.json` In alternativa, utilizzare npm-check-updates.

> **Important**
>
> In base alla progettazione, quando installi o aggiorni le dipendenze, NPM e Yarn scelgono la versione più recente di ogni pacchetto che soddisfi i requisiti specificati in. `package.json` C'è sempre il rischio che queste versioni possano essere danneggiate (accidentalmente o intenzionalmente). Effettua un test approfondito dopo aver aggiornato le dipendenze del progetto.

AWS CDK idiomi in TypeScript

oggetti di scena

Tutte le classi di AWS Construct Library vengono istanziate utilizzando tre argomenti: l'ambito in cui viene definito il costrutto (il suo genitore nell'albero dei costrutti), un id e props. Argument props è un insieme di coppie chiave/valore che il costrutto utilizza per configurare le risorse che crea. AWS Anche altre classi e metodi utilizzano il modello «bundle of attributes» per gli argomenti.

In TypeScript, la forma di `props` è definita utilizzando un'interfaccia che indica gli argomenti obbligatori e facoltativi e i loro tipi. Tale interfaccia è definita per ogni tipo di `props` argomento, in genere specifica per un singolo costrutto o metodo. Ad esempio, il costrutto Bucket (in`aws-cdk-lib/aws-s3 module`) specifica un `props` argomento conforme all'interfaccia. BucketProps

Se una proprietà è essa stessa un oggetto, ad esempio la proprietà WebsiteRedirect di`BucketProps`, tale oggetto avrà una propria interfaccia alla quale la forma deve conformarsi, in questo caso. RedirectTarget

Se state sottoclassando una classe AWS Construct Library (o sovrascrivete un metodo che accetta un argomento simile a props), potete ereditare dall'interfaccia esistente crearne una nuova che specifichi tutti i nuovi oggetti di scena richiesti dal codice. Quando chiamate la classe principale o il metodo base, in genere potete passare l'intero argomento props ricevuto, poiché tutti gli attributi forniti nell'oggetto ma non specificati nell'interfaccia verranno ignorati.

Le future versioni di AWS CDK potrebbero aggiungere casualmente una nuova proprietà con un nome che hai usato per la tua proprietà. Il trasferimento del valore ricevuto nella catena di ereditarietà può quindi causare un comportamento imprevisto. È più sicuro passare una copia superficiale degli oggetti di scena ricevuti con la proprietà rimossa o impostata. `undefined` Per esempio:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

In alternativa, dai un nome alle tue proprietà in modo che sia chiaro che appartengono al tuo costrutto. In questo modo, è improbabile che entrino in collisione con le proprietà nelle AWS CDK versioni future. Se ce ne sono molte, utilizzate un unico oggetto dal nome appropriato per contenerle.

Valori mancanti

I valori mancanti in un oggetto (come gli oggetti di scena) hanno il valore in. `undefined` TypeScript La versione 3.7 del linguaggio ha introdotto operatori che semplificano l'utilizzo di questi valori, semplificando la specificazione dei valori predefiniti e il «cortocircuito» il concatenamento quando viene raggiunto un valore indefinito. Per ulteriori informazioni su queste funzionalità, consultate le Note di rilascio della versione TypeScript 3.7, in particolare le prime due funzionalità, Optional Chaining e Nullish Coalescing.

Creazione, sintesi e distribuzione

In genere, dovresti trovarti nella directory principale del progetto durante la creazione e l'esecuzione dell'applicazione.

Node.js non può essere eseguito TypeScript direttamente; l'applicazione viene invece convertita per JavaScript utilizzare il TypeScript compilatore,`tsc`. Il JavaScript codice risultante viene quindi eseguito.

Lo fa AWS CDK automaticamente ogni volta che è necessario eseguire l'app. Tuttavia, può essere utile compilare manualmente per verificare la presenza di errori ed eseguire test. Per compilare la tua TypeScript app manualmente, emetti. `npm run build` Potresti anche dover accedere `npm run watch` alla modalità watch, in cui il TypeScript compilatore ricostruisce automaticamente l'app ogni volta che salvi le modifiche a un file sorgente.

Gli stack definiti nell' AWS CDK app possono essere sintetizzati e distribuiti singolarmente o insieme utilizzando i comandi seguenti. In genere, dovresti trovarti nella directory principale del tuo progetto quando li pubblichi.

- `cdk synth`: sintetizza un AWS CloudFormation modello da uno o più stack dell'app. AWS CDK
- `cdk deploy`: distribuisce le risorse definite da uno o più stack dell'app su. AWS CDK AWS

È possibile specificare i nomi di più stack da sintetizzare o distribuire in un unico comando. Se l'app definisce solo uno stack, non è necessario specificarlo.

```
cdk synth                # app defines single stack
```

```
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Puoi anche usare i caratteri jolly * (qualsiasi numero di caratteri) e? (qualsiasi carattere singolo) per identificare le pile in base al modello. Quando usate i caratteri jolly, racchiudete il pattern tra virgolette. Altrimenti, la shell potrebbe provare ad espanderlo ai nomi dei file nella directory corrente prima che vengano passati al AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Non è necessario sintetizzare esplicitamente gli stack prima di distribuirli; `cdk deploy` esegue questo passaggio per assicurarsi che venga distribuito il codice più recente.

Per la documentazione completa del comando, consulta. `cdk` the section called “AWS CDK Kit di strumenti”

Lavorare con l' AWS CDK interno JavaScript

JavaScriptè un linguaggio client completamente supportato per AWS CDK ed è considerato stabile. L'utilizzo dell' AWS Cloud Development Kit (AWS CDK) in JavaScript utilizza strumenti familiari, tra cui Node.js e Node Package Manager (`npm`). Puoi anche usare Yarn se preferisci, sebbene gli esempi in questa guida utilizzino NPM. I moduli che compongono la AWS Construct Library sono distribuiti tramite il repository NPM, npmjs.org.

Puoi usare qualsiasi editor o IDE. Molti AWS CDK sviluppatori utilizzano Visual Studio Code (o il suo equivalente open source VSCodium), che offre un buon supporto per. JavaScript

Argomenti

- Inizia a usare JavaScript
- Creare un progetto
- Utilizzo del locale cdk
- Gestione dei moduli Construct Library AWS
- Gestione delle dipendenze in JavaScript
- AWS CDK idiomi in JavaScript

- Sintetizzazione e distribuzione
- Utilizzo TypeScript di esempi con JavaScript
- Migrazione a TypeScript

Inizia a usare JavaScript

Per utilizzare AWS CDK, è necessario disporre di un AWS account e delle credenziali e aver installato Node.js e il Toolkit. AWS CDK Per informazioni, consulta Iniziare con AWS CDK.

JavaScript AWS CDK le applicazioni non richiedono prerequisiti aggiuntivi oltre a questi.

> **Note**
>
> Obsoletamento linguistico di terze parti: la versione linguistica è supportata solo fino alla fine del ciclo di vita (EOL (End Of Life) condivisa dal fornitore o dalla community ed è soggetta a modifiche con preavviso.

Creare un progetto

È possibile creare un nuovo AWS CDK progetto `cdk init` richiamandolo in una directory vuota. Utilizzate l'`--language`opzione e specificate`javascript`:

```
mkdir my-project
cd my-project
cdk init app --language javascript
```

La creazione di un progetto installa anche il `aws-cdk-lib`modulo e le sue dipendenze.

`cdk init`utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. I trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Tuttavia, il nome dovrebbe altrimenti assumere la forma di un JavaScript identificatore; ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Utilizzo del locale **cdk**

Per la maggior parte, questa guida presuppone l'installazione di CDK Toolkit a livello globale (`npm install -g aws-cdk`) e gli esempi di comandi forniti (ad esempio`cdk synth`) seguano questo

presupposto. Questo approccio semplifica l'aggiornamento di CDK Toolkit e, poiché il CDK adotta un approccio rigoroso alla compatibilità con le versioni precedenti, il rischio di utilizzare sempre la versione più recente è generalmente minimo.

Alcuni team preferiscono specificare tutte le dipendenze all'interno di ciascun progetto, inclusi strumenti come CDK Toolkit. Questa pratica consente di associare tali componenti a versioni specifiche e di garantire che tutti gli sviluppatori del team (e dell'ambiente CI/CD) utilizzino esattamente quelle versioni. Ciò elimina una possibile fonte di cambiamento, contribuendo a rendere le build e le implementazioni più coerenti e ripetibili.

Il CDK include una dipendenza per il CDK Toolkit nei modelli di JavaScript progetto`package.json`, quindi se si desidera utilizzare questo approccio, non è necessario apportare alcuna modifica al progetto. Tutto quello che dovete fare è usare comandi leggermente diversi per creare l'app e per emettere comandi. `cdk`

Operazione	Usa CDK Toolkit globale	Usa CDK Toolkit locale
Inizializza il progetto	`cdk init --language javascript`	`npx aws-cdk init --linguaggio javascript`
Esegui il comando CDK Toolkit	`cdk...`	`npm esegui cdk...` or `npx aws-cdk...`

`npx aws-cdk`esegue la versione del CDK Toolkit installata localmente nel progetto corrente, se ne esiste una, ricorrendo all'installazione globale, se presente. Se non esiste un'installazione globale, `npx` scarica una copia temporanea di CDK Toolkit e la esegue. È possibile specificare una versione arbitraria di CDK Toolkit utilizzando la sintassi: prints. @ `npx aws-cdk@1.120 --version 1.120.0`

> **Tip**
>
> Imposta un alias in modo da poter utilizzare il `cdk` comando con un'installazione locale di CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```

Windows

```
doskey cdk=npx aws-cdk $*
```

Gestione dei moduli Construct Library AWS

Utilizzate Node Package Manager (`npm`) per installare e aggiornare i moduli AWS Construct Library per l'uso da parte delle vostre app e degli altri pacchetti di cui avete bisogno. (Puoi usare `yarn` al posto di `npm` se preferisci.) `npm`installa inoltre automaticamente le dipendenze per quei moduli.

La maggior parte dei AWS CDK costrutti si trova nel pacchetto CDK principale, denominato`aws-cdk-lib`, che è una dipendenza predefinita nei nuovi progetti creati da. `cdk init` I moduli «sperimentali» di AWS Construct Library, in cui i costrutti di livello superiore sono ancora in fase di sviluppo, hanno lo stesso nome. `aws-cdk-lib/`*`SERVICE-NAME`*`-alpha` Il nome del servizio ha un prefisso aws-. Se non sei sicuro del nome di un modulo, cercalo su NPM.

> **Note**
>
> Il CDK API Reference mostra anche i nomi dei pacchetti.

Ad esempio, il comando seguente installa il modulo sperimentale per. AWS CodeStar

```
npm install @aws-cdk/aws-codestar-alpha
```

Il supporto di Construct Library di alcuni servizi è disponibile in più di un namespace. Ad esempio`aws-route53`, ci sono inoltre tre namespace Amazon Route 53 aggiuntivi,`aws-route53-targets`, `aws-route53-patterns` e. `aws-route53resolver`

Le dipendenze del tuo progetto vengono mantenute in. `package.json` Puoi modificare questo file per bloccare alcune o tutte le tue dipendenze su una versione specifica o per consentire loro di essere aggiornate a versioni più recenti in base a determinati criteri. Per aggiornare le dipendenze NPM del progetto all'ultima versione consentita in base alle regole specificate in: `package.json`

```
npm update
```

In JavaScript, importi i moduli nel tuo codice con lo stesso nome che usi per installarli usando NPM. Consigliamo le seguenti pratiche per importare AWS CDK classi e moduli AWS Construct Library nelle applicazioni. Seguire queste linee guida contribuirà a rendere il codice coerente con altre AWS CDK applicazioni e più facile da comprendere.

- Utilizza`require()`, non direttive in stile ES6. `import` Le versioni precedenti di Node.js non supportano le importazioni ES6, quindi l'utilizzo della sintassi precedente è più ampiamente compatibile. (Se vuoi davvero utilizzare le importazioni ES6, usa esm per assicurarti che il tuo progetto sia compatibile con tutte le versioni supportate di Node.js.)
- In genere, importa singole classi da. `aws-cdk-lib`

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

- Se hai bisogno di molte classi da`aws-cdk-lib`, puoi usare un alias dello spazio dei nomi di `cdk` invece di importare le singole classi. Evita di fare entrambe le cose.

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

- In genere, importate le librerie AWS Construct utilizzando alias di namespace brevi.

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

Gestione delle dipendenze in JavaScript

Nei progetti JavaScript CDK, le dipendenze sono specificate nel `package.json` file nella directory principale del progetto. I AWS CDK moduli principali si trovano in un unico NPM pacchetto chiamato. `aws-cdk-lib`

Quando installate un pacchetto utilizzandonpm install, NPM registra il pacchetto al posto `package.json` vostro.

Se preferisci, puoi usare Yarn al posto di NPM. Tuttavia, il CDK non supporta la modalità di Yarn, che è la plug-and-play modalità predefinita in Yarn 2. Aggiungi quanto segue al `.yarnrc.yml` file del tuo progetto per disattivare questa funzionalità.

```
nodeLinker: node-modules
```

Applicazioni CDK

Di seguito è riportato un `package.json` file di esempio generato dal `cdk init --language typescript` comando. Il file generato per JavaScript è simile, solo senza le voci TypeScript relative.

```
{
  "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"
  }
}
```

Per le app CDK distribuibili, `aws-cdk-lib` deve essere specificato nella sezione di. `dependencies` `package.json` Potete utilizzare un identificatore del numero di versione con cursore (^) per indicare che accetterete versioni successive a quella specificata purché si trovino all'interno della stessa versione principale.

Per i costrutti sperimentali, specificate le versioni esatte per i moduli della libreria alpha construct, che hanno API che possono cambiare. Non utilizzate ^ o ~ poiché le versioni successive di questi moduli potrebbero apportare modifiche all'API che potrebbero danneggiare l'app.

Specificate le versioni delle librerie e degli strumenti necessari per testare l'app (ad esempio, il framework di `jest` test) nella `devDependencies` sezione di`package.json`. Facoltativamente, usa ^ per specificare che le versioni compatibili successive sono accettabili.

Librerie di costruzioni di terze parti

Se state sviluppando una libreria di costrutti, specificate le sue dipendenze utilizzando una combinazione delle `devDependencies` sezioni `peerDependencies` e, come mostrato nel seguente file di esempio. `package.json`

```
{
  "name": "my-package",
  "version": "0.0.1",
  "peerDependencies": {
    "aws-cdk-lib": "^2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "^10.0.0"
  },
  "devDependencies": {
    "aws-cdk-lib": "2.14.0",
    "@aws-cdk/aws-appsync-alpha": "2.10.0-alpha",
    "constructs": "10.0.0",
    "jsii": "^1.50.0",
    "aws-cdk": "^2.14.0"
  }
}
```

In`peerDependencies`, usa un accento circonflesso (^) per specificare la versione più bassa con `aws-cdk-lib` cui funziona la tua libreria. Ciò massimizza la compatibilità della tua libreria con una gamma di versioni CDK. Specificate le versioni esatte per i moduli della libreria alpha construct, che hanno API che possono cambiare. L'utilizzo `peerDependencies` assicura che vi sia una sola copia di tutte le librerie CDK nell'albero. `node_modules`

Nel`devDependencies`, specificate gli strumenti e le librerie necessari per il test, opzionalmente con ^ per indicare che le versioni compatibili successive sono accettabili. Specificate esattamente (senza ^ o ~) le versioni più basse `aws-cdk-lib` e gli altri pacchetti CDK con cui pubblicizzate la compatibilità della vostra libreria. Questa pratica assicura che i test vengano eseguiti su tali versioni. In questo modo, se usi inavvertitamente una funzionalità presente solo nelle versioni più recenti, i tuoi test possono catturarla.

> **Warning**
>
> `peerDependencies`vengono installati automaticamente solo da NPM 7 e versioni successive. Se stai usando NPM 6 o versioni precedenti, o se stai usando Yarn, devi includere le dipendenze delle tue dipendenze in. `devDependencies` Altrimenti, non verranno installate e riceverai un avviso sulle dipendenze tra pari non risolte.

Installazione e aggiornamento delle dipendenze

Esegui il comando seguente per installare le dipendenze del tuo progetto.

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
```

Per aggiornare i moduli installati, è possibile utilizzare yarn upgrade i comandi precedenti npm install e. Entrambi i comandi aggiornano i pacchetti `node_modules` alle versioni più recenti che soddisfano le regole in`package.json`. Tuttavia, non `package.json` si aggiornano da soli, cosa che potresti voler fare per impostare una nuova versione minima. Se ospiti il pacchetto su GitHub, puoi configurare gli aggiornamenti della versione di Dependabot in modo che si aggiornino automaticamente. `package.json` In alternativa, utilizzare npm-check-updates.

> **Important**
>
> In base alla progettazione, quando installi o aggiorni le dipendenze, NPM e Yarn scelgono la versione più recente di ogni pacchetto che soddisfi i requisiti specificati in. `package.json`

C'è sempre il rischio che queste versioni possano essere danneggiate (accidentalmente o intenzionalmente). Effettua un test approfondito dopo aver aggiornato le dipendenze del progetto.

AWS CDK idiomi in JavaScript

oggetti di scena

Tutte le classi di AWS Construct Library vengono istanziate utilizzando tre argomenti: l'ambito in cui viene definito il costrutto (l'elemento principale nell'albero dei costrutti), un id e props, un insieme di coppie chiave/valore che il costrutto utilizza per configurare le risorse che crea. AWS Anche altre classi e metodi utilizzano il modello «bundle of attributes» per gli argomenti.

L'uso di un IDE o di un editor con un buon JavaScript completamento automatico aiuterà a evitare errori di ortografia dei nomi delle proprietà. Se un costrutto prevede una `encryptionKeys` proprietà e la scriveteencryptionkeys, quando create un'istanza del costrutto non avete passato il valore desiderato. Ciò può causare un errore al momento della sintesi se la proprietà è obbligatoria o causare l'ignoramento silenzioso della proprietà se è facoltativa. In quest'ultimo caso, potresti ottenere un comportamento predefinito che intendevi sostituire. Fate particolare attenzione a questo punto.

Quando sottoclassi una classe AWS Construct Library (o si sovrascrive un metodo che accetta un argomento simile a props), potresti voler accettare proprietà aggiuntive per uso personale. Questi valori verranno ignorati dalla classe principale o dal metodo sovrascritto, perché non sono mai accessibili in quel codice, quindi in genere potete trasmettere tutti gli oggetti di scena che avete ricevuto.

Le future versioni di AWS CDK potrebbero aggiungere casualmente una nuova proprietà con un nome che hai usato per la tua proprietà. Il trasferimento del valore ricevuto nella catena di ereditarietà può quindi causare un comportamento imprevisto. È più sicuro passare una copia superficiale degli oggetti di scena ricevuti con la proprietà rimossa o impostata. `undefined` Per esempio:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

In alternativa, dai un nome alle tue proprietà in modo che sia chiaro che appartengono al tuo costrutto. In questo modo, è improbabile che entrino in collisione con le proprietà nelle AWS CDK versioni future. Se ce ne sono molte, utilizzate un unico oggetto dal nome appropriato per contenerle.

Valori mancanti

I valori mancanti in un oggetto (ad esempio`props`) hanno il valore in. `undefined` JavaScript Per affrontarli si applicano le tecniche usuali. Ad esempio, un linguaggio comune per accedere a una proprietà di un valore che può essere indefinito è il seguente:

```
// 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;
```

Tuttavia, se a potesse avere qualche altro valore «falso» oltre a questo`undefined`, è meglio rendere il test più esplicito. Qui, trarremo vantaggio dal fatto che `null` `undefined` siamo uguali per testarli entrambi contemporaneamente:

```
let c = a == null ? a : a.b;
```

> **Tip**
>
> Node.js 14.0 e versioni successive supportano nuovi operatori in grado di semplificare la gestione di valori non definiti. Per ulteriori informazioni, consulta le proposte opzionali di concatenamento e nullish coalescenza.

Sintetizzazione e distribuzione

Gli stack definiti nell' AWS CDK app possono essere sintetizzati e distribuiti singolarmente o insieme utilizzando i comandi seguenti. In genere, dovresti trovarti nella directory principale del tuo progetto quando li pubblichi.

- `cdk synth`: sintetizza un AWS CloudFormation modello da uno o più stack dell'app. AWS CDK
- `cdk deploy`: distribuisce le risorse definite da uno o più stack dell'app su. AWS CDK AWS

È possibile specificare i nomi di più stack da sintetizzare o distribuire in un unico comando. Se l'app definisce solo uno stack, non è necessario specificarlo.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

Puoi anche usare i caratteri jolly * (qualsiasi numero di caratteri) e? (qualsiasi carattere singolo) per identificare le pile in base allo schema. Quando usate i caratteri jolly, racchiudete il pattern tra virgolette. Altrimenti, la shell potrebbe provare ad espanderlo ai nomi dei file nella directory corrente prima che vengano passati al AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Non è necessario sintetizzare esplicitamente gli stack prima di distribuirli; `cdk deploy` esegue questo passaggio per assicurarsi che venga distribuito il codice più recente.

Per la documentazione completa del comando, consulta. `cdk` the section called “AWS CDK Kit di strumenti”

Utilizzo TypeScript di esempi con JavaScript

TypeScriptè il linguaggio che utilizziamo per sviluppare il AWS CDK, ed è stato il primo linguaggio supportato per lo sviluppo di applicazioni, quindi sono scritti molti esempi di AWS CDK codice disponibili TypeScript. Questi esempi di codice possono essere una buona risorsa per JavaScript gli sviluppatori; è sufficiente rimuovere le parti TypeScript specifiche del codice.

TypeScript Gli snippet utilizzano spesso il nuovo ECMAScript `import` e le `export` parole chiave per importare oggetti da altri moduli e dichiarare che gli oggetti devono essere resi disponibili all'esterno del modulo corrente. Node.js ha appena iniziato a supportare queste parole chiave nelle sue ultime versioni. A seconda della versione di Node.js che stai utilizzando (o che desideri supportare), potresti riscrivere le importazioni e le esportazioni per utilizzare la sintassi precedente.

Le importazioni possono essere sostituite con chiamate alla `require()` funzione.

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');
```

Le esportazioni possono essere assegnate all'`module.exports`oggetto.

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**
>
> Un'alternativa all'utilizzo delle importazioni e delle esportazioni vecchio stile consiste nell'utilizzare il `esm`modulo.

Dopo aver ordinato le importazioni e le esportazioni, puoi approfondire il codice effettivo. Potresti imbatterti in queste funzionalità di uso comune: TypeScript

- Digita le annotazioni
- Definizioni di interfaccia
- Conversioni di tipo/cast

- Modificatori di accesso

È possibile fornire annotazioni di tipo per variabili, membri della classe, parametri di funzione e tipi restituiti dalle funzioni. Per le variabili, i parametri e i membri, i tipi vengono specificati seguendo l'identificatore con i due punti e il tipo. I valori restituiti dalla funzione seguono la firma della funzione e sono costituiti da due punti e dal tipo.

Per convertire il codice annotato sul tipo in JavaScript, rimuovete i due punti e il tipo. I membri della classe devono inserire un valore JavaScript; impostatelo su `undefined` se hanno solo un'annotazione di tipo. 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) {
    // ...
}
```

In TypeScript, le interfacce vengono utilizzate per dare un nome a gruppi di proprietà obbligatorie e opzionali e ai relativi tipi. È quindi possibile utilizzare il nome dell'interfaccia come annotazione del

tipo. TypeScript si assicurerà che l'oggetto utilizzato come, ad esempio, argomento di una funzione abbia le proprietà richieste dei tipi corretti.

```
interface myFuncProps {
    code: lambda.Code,
    handler?: string
}
```

JavaScript non ha una funzionalità di interfaccia, quindi dopo aver rimosso le annotazioni di tipo, elimina completamente le dichiarazioni dell'interfaccia.

Quando una funzione o un metodo restituisce un tipo generico (ad esempio`object`), ma si desidera trattare tale valore come un tipo figlio più specifico per accedere a proprietà o metodi che non fanno parte dell'interfaccia del tipo più generale, TypeScript consente di eseguire il cast del valore utilizzando `as` seguito da un tipo o da un nome di interfaccia. JavaScript non lo supporta (o non ne ha bisogno), quindi è sufficiente rimuovere `as` e il seguente identificatore. Una sintassi cast meno comune consiste nell'utilizzare il nome di un tipo tra parentesi`<LikeThis>`; anche questi cast devono essere rimossi.

Infine, TypeScript supporta i modificatori `public` di accesso e per i membri delle classi. `protected` `private` Tutti i membri della classe JavaScript sono pubblici. Rimuovi semplicemente questi modificatori ovunque li vedi.

Sapere come identificare e rimuovere queste TypeScript funzionalità è fondamentale per adattare brevi TypeScript frammenti a. JavaScript Tuttavia, potrebbe non essere pratico convertire TypeScript esempi più lunghi in questo modo, poiché è più probabile che utilizzino altre funzionalità. TypeScript Per queste situazioni, consigliamo Sucrase. Sucrase non si lamenterà se il codice utilizza una variabile non definita, ad esempio, come farebbe. `tsc` Se è sintatticamente valido, con poche eccezioni, Sucrase può tradurlo in. JavaScript Ciò lo rende particolarmente utile per convertire frammenti che potrebbero non essere eseguibili da soli.

Migrazione a TypeScript

Molti JavaScript sviluppatori si trasferiscono TypeScriptman mano che i loro progetti diventano più grandi e complessi. TypeScript è un superset di JavaScript: tutto il JavaScript codice è un TypeScript codice valido, quindi non sono necessarie modifiche al codice, ed è anche un linguaggio supportato. AWS CDK Le annotazioni di tipo e altre TypeScript funzionalità sono opzionali e possono essere aggiunte all' AWS CDK app man mano che ne trovi il valore. TypeScript offre inoltre l'accesso

anticipato a nuove JavaScript funzionalità, come il concatenamento opzionale e il coalescenza nullish, prima che vengano finalizzate e senza richiedere l'aggiornamento di Node.js.

TypeScript, che definiscono pacchetti di proprietà obbligatorie e opzionali (e i relativi tipi) all'interno di un oggetto, consentono di rilevare gli errori più comuni durante la scrittura del codice e semplificano l'IDE nel fornire solidi consigli di completamento automatico e altri consigli di codifica in tempo reale.

La codifica TypeScript comporta un passaggio aggiuntivo: la compilazione dell'app con il compilatore,. TypeScript `tsc` Per AWS CDK le app tipiche, la compilazione richiede al massimo alcuni secondi.

Il modo più semplice per migrare un' JavaScript AWS CDK app esistente TypeScript è creare un nuovo TypeScript progetto utilizzando `cdk init app --language typescript` e quindi copiare i file sorgente (e qualsiasi altro file necessario, ad esempio risorse come il codice sorgente delle AWS Lambda funzioni) nel nuovo progetto. Rinomina i JavaScript file in modo che terminino `.ts` e inizi a svilupparli in. TypeScript

Lavorare con il AWS CDK in Python

Python è un linguaggio client completamente supportato per AWS Cloud Development Kit (AWS CDK) ed è considerato stabile. Lavorare con Python utilizza strumenti familiari, tra cui l'implementazione standard di Python (CPython), ambienti virtuali con `virtualenv` e il programma di installazione del pacchetto Python. AWS CDK `pip` I moduli che compongono la Construct Library sono distribuiti tramite pypi.org. AWS La versione Python di AWS CDK even utilizza identificatori in stile Python (ad esempio, i nomi dei metodi). `snake_case`

Puoi usare qualsiasi editor o IDE. Molti AWS CDK sviluppatori usano Visual Studio Code (o il suo equivalente open source VSCodium), che supporta bene Python tramite un'estensione ufficiale. L'editor IDLE incluso in Python sarà sufficiente per iniziare. I moduli Python per la AWS CDK hanno suggerimenti di tipo, utili per uno strumento di linting o un IDE che supporta la convalida dei tipi.

Argomenti

- Inizia a usare Python
- Creare un progetto
- Gestione dei moduli di AWS Construct Library
- Gestione delle dipendenze in Python
- AWS CDK idiomi in Python

- Sintetizzazione e distribuzione

Inizia a usare Python

Per utilizzare AWS CDK, è necessario disporre di un AWS account e delle credenziali e aver installato Node.js e il Toolkit. AWS CDK Per informazioni, consulta Iniziare con AWS CDK.

AWS CDK Le applicazioni Python richiedono Python 3.6 o successivo. Se non lo hai già installato, scarica una versione compatibile per il tuo sistema operativo su python.org. Se usi Linux, il tuo sistema potrebbe avere una versione compatibile, oppure puoi installarlo usando il gestore di pacchetti della tua distribuzione (`yum`apt`, ecc.). Gli utenti Mac potrebbero essere interessati a Homebrew, un gestore di pacchetti in stile Linux per macOS.

> **Note**
>
> Obsolitudine linguistica di terze parti: la versione linguistica è supportata solo fino alla fine del ciclo di vita (EOL (End Of Life) condivisa dal fornitore o dalla community ed è soggetta a modifiche con preavviso.

Sono richiesti anche il programma di installazione del pacchetto Python e il gestore dell'ambiente virtuale. `pip` `virtualenv` Le installazioni Windows di versioni Python compatibili includono questi strumenti. Su Linux, `pip` e `virtualenv` può essere fornito come pacchetti separati nel gestore di pacchetti. In alternativa, puoi installarli con i seguenti comandi:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

Se riscontrate un errore di autorizzazione, eseguite i comandi precedenti con il `--user` flag in modo che i moduli siano installati nella vostra directory utente, oppure usateli `sudo` per ottenere le autorizzazioni per installare i moduli a livello di sistema.

> **Note**
>
> È comune che le distribuzioni Linux utilizzino il nome eseguibile `python3` per Python 3.x e `python` facciano riferimento a un'installazione di Python 2.x. Alcune distribuzioni hanno un pacchetto opzionale che puoi installare che fa sì che il `python` comando faccia riferimento

a Python 3. In caso contrario, è possibile modificare il comando utilizzato per eseguire l'applicazione modificandolo `cdk.json` nella directory principale del progetto.

> **Note**
>
> Su Windows, potresti voler invocare Python (pipand) usando py l'eseguibile, il programma di avvio >Python per Windows. Tra le altre cose, il programma di avvio consente di specificare facilmente quale versione installata di Python si desidera utilizzare.
> Se digitando python nella riga di comando viene visualizzato un messaggio sull'installazione di Python da Windows Store, anche dopo aver installato una versione Windows di Python, apri il pannello delle impostazioni Manage App Execution Aliases di Windows e disattiva le due voci dell'App Installer per Python.

Creare un progetto

È possibile creare un nuovo AWS CDK progetto richiamandolo in una directory vuota. `cdk init` Utilizzate l'`--language`opzione e specificate`python`:

```
mkdir my-project
cd my-project
cdk init app --language python
```

`cdk init`utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. I trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Tuttavia, il nome dovrebbe altrimenti seguire la forma di un identificatore Python; ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Per lavorare con il nuovo progetto, attivare il suo ambiente virtuale. Ciò consente di installare le dipendenze del progetto localmente nella cartella del progetto, anziché globalmente.

```
source .venv/bin/activate
```

> **Note**
>
> Potresti riconoscerlo come il comando Mac/Linux per attivare un ambiente virtuale. I modelli Python includono un file batch, `source.bat`, che consente di utilizzare lo stesso comando

su Windows. Anche il comando tradizionale di Windows `.venv\Scripts\activate.bat`, funziona.
Se hai inizializzato il AWS CDK progetto utilizzando CDK Toolkit v1.70.0 o versione precedente, l'ambiente virtuale si trova invece nella directory. `.env` `.venv`

> **Important**
>
> Attiva l'ambiente virtuale del progetto ogni volta che inizi a lavorarci. Altrimenti, non avrai accesso ai moduli installati lì e i moduli che installerai andranno nella directory globale dei moduli di Python (o genereranno un errore di autorizzazione).

Dopo aver attivato l'ambiente virtuale per la prima volta, installa le dipendenze standard dell'app:

```
python -m pip install -r requirements.txt
```

Gestione dei moduli di AWS Construct Library

Usa il programma di installazione del pacchetto Python per installare e aggiornare i moduli di AWS Construct Library per l'uso da parte delle tue app e degli altri pacchetti di cui hai bisogno. pip pipinstalla inoltre automaticamente le dipendenze per tali moduli. Se il tuo sistema non riconosce pip come comando autonomo, invoca pip come modulo Python, in questo modo:

```
python -m pip PIP-COMMAND
```

La maggior parte dei AWS CDK costrutti sono disponibili. `aws-cdk-lib` I moduli sperimentali si trovano in moduli separati denominati come`aws-cdk.SERVICE-NAME.alpha`. Il nome del servizio include un prefisso aws. Se non sei sicuro del nome di un modulo, cercalo in PyPI. Ad esempio, il comando seguente installa la libreria. AWS CodeStar

```
python -m pip install aws-cdk.aws-codestar-alpha
```

I costrutti di alcuni servizi si trovano in più di un namespace. Ad esempio`aws-cdk.aws-route53`, ci sono inoltre tre namespace Amazon Route 53 aggiuntivi`aws-route53-targets`, `aws-route53-patterns` denominati e. `aws-route53resolver`

> **Note**
>
> L'edizione Python del CDK API Reference mostra anche i nomi dei pacchetti.

I nomi usati per importare i moduli AWS Construct Library nel codice Python sono i seguenti.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

Consigliamo le seguenti pratiche per importare AWS CDK classi e moduli AWS Construct Library nelle applicazioni. Seguire queste linee guida contribuirà a rendere il codice coerente con altre AWS CDK applicazioni e più facile da comprendere.

- In genere, importa singole classi dal livello superiore`aws_cdk`.

  ```
  from aws_cdk import App, Construct
  ```

- Se hai bisogno di molte classi da`aws_cdk`, puoi utilizzare un alias dello spazio dei nomi di `cdk` invece di importare singole classi. Evita di fare entrambe le cose.

  ```
  import aws_cdk as cdk
  ```

- In genere, importa le librerie AWS Construct utilizzando alias di namespace brevi.

  ```
  import aws_cdk.aws_s3 as s3
  ```

Dopo aver installato un modulo, aggiorna il `requirements.txt` file del progetto, che elenca le dipendenze del progetto. È meglio farlo manualmente piuttosto che usare`pip freeze`. `pip freeze`acquisisce le versioni correnti di tutti i moduli installati nell'ambiente virtuale Python, il che può essere utile quando si raggruppa un progetto da eseguire altrove.

Di solito, tuttavia, è `requirements.txt` necessario elencare solo le dipendenze di primo livello (moduli da cui dipende direttamente l'app) e non le dipendenze di tali librerie. Questa strategia semplifica l'aggiornamento delle dipendenze.

È possibile modificare `requirements.txt` per consentire gli aggiornamenti; è sufficiente sostituire il numero di versione == precedente con ~= per consentire l'aggiornamento a una versione compatibile

superiore o rimuovere completamente il requisito di versione per specificare l'ultima versione disponibile del modulo.

Se `requirements.txt` modificate in modo appropriato per consentire gli aggiornamenti, eseguite questo comando per aggiornare i moduli installati del progetto in qualsiasi momento:

```
pip install --upgrade -r requirements.txt
```

Gestione delle dipendenze in Python

In Python, si specificano le dipendenze inserendole nelle applicazioni o `setup.py` nelle `requirements.txt` librerie di costruzione. Le dipendenze vengono quindi gestite con lo strumento PIP. PIP viene richiamato in uno dei seguenti modi:

```
pip command options
python -m pip command options
```

L'python -m pipinvocazione funziona sulla maggior parte dei sistemi; pip richiede che l'eseguibile di PIP sia presente nel percorso di sistema. Se pip non funziona, prova a sostituirlo con. python -m pip

Il cdk init --language python comando crea un ambiente virtuale per il nuovo progetto. Ciò consente a ogni progetto di avere le proprie versioni di dipendenze e anche un `requirements.txt` file di base. È necessario attivare questo ambiente virtuale eseguendolo source .venv/bin/activate ogni volta che si inizia a lavorare con il progetto.

Applicazioni CDK

Di seguito è riportato un esempio del file `requirements.txt`. Poiché PIP non dispone di una funzionalità di blocco delle dipendenze, si consiglia di utilizzare l'operatore == per specificare le versioni esatte per tutte le dipendenze, come illustrato di seguito.

```
aws-cdk-lib==2.14.0
aws-cdk.aws-appsync-alpha==2.10.0a0
```

L'installazione di un modulo con pip install non lo aggiunge automaticamente a. `requirements.txt` Devi farlo tu stesso. Se desideri eseguire l'aggiornamento a una versione successiva di una dipendenza, modificane il numero di versione in`requirements.txt`.

Per installare o aggiornare le dipendenze del progetto dopo la creazione o la modifica`requirements.txt`, esegui quanto segue:

```
python -m pip install -r requirements.txt
```

> **Tip**
>
> Il pip freeze comando restituisce le versioni di tutte le dipendenze installate in un formato che può essere scritto in un file di testo. Questo può essere usato come file dei requisiti con. `pip install -r` Questo file è utile per aggiungere tutte le dipendenze (comprese quelle transitive) alle versioni esatte con cui hai testato. Per evitare problemi quando si aggiornano i pacchetti in un secondo momento, utilizzate un file separato, ad esempio (not). `freeze.txt` `requirements.txt` Quindi, rigeneralo quando aggiorni le dipendenze del progetto.

Librerie di costruzioni di terze parti

Nelle librerie, le dipendenze sono specificate in`setup.py`, in modo che le dipendenze transitive vengano scaricate automaticamente quando il pacchetto viene utilizzato da un'applicazione. Altrimenti, ogni applicazione che desidera utilizzare il pacchetto deve copiare le dipendenze al suo interno. `requirements.txt` Un esempio `setup.py` è mostrato qui.

```
from setuptools import setup

setup(
  name='my-package',
  version='0.0.1',
  install_requires=[
    'aws-cdk-lib==2.14.0',
  ],
  ...
)
```

Per lavorare sul pacchetto per lo sviluppo, crea o attiva un ambiente virtuale, quindi esegui il seguente comando.

```
python -m pip install -e .
```

Sebbene PIP installi automaticamente le dipendenze transitive, può esserci una sola copia installata di ogni pacchetto. Viene selezionata la versione specificata più in alto nell'albero delle dipendenze; le applicazioni hanno sempre l'ultima parola sulla versione dei pacchetti da installare.

AWS CDK idiomi in Python

Conflitti linguistici

In Python, `lambda` è una parola chiave del linguaggio, quindi non è possibile utilizzarla come nome per il modulo della libreria di AWS Lambda costruzioni o le funzioni Lambda. La convenzione di Python per tali conflitti consiste nell'utilizzare un carattere di sottolineatura finale, come in`lambda_`, nel nome della variabile.

Per convenzione, il secondo argomento dei AWS CDK costrutti viene denominato. `id` Quando scrivi i tuoi stack e i tuoi costrutti, chiamando un parametro `id` «shadows» la funzione incorporata in Python`id()`, che restituisce l'identificatore univoco di un oggetto. Questa funzione non viene usata molto spesso, ma se ti capita di averne bisogno nel tuo costrutto, rinomina l'argomento, ad esempio. `construct_id`

Argomenti e proprietà

Tutte le classi di AWS Construct Library vengono istanziate utilizzando tre argomenti: l'ambito in cui viene definito il costrutto (l'elemento principale nell'albero di costruzione), un id e props, un insieme di coppie chiave/valore che il costrutto utilizza per configurare le risorse che crea. Anche altre classi e metodi utilizzano il modello «bundle of attributes» per gli argomenti.

scope e id devono essere sempre passati come argomenti posizionali, non come argomenti di parole chiave, perché i loro nomi cambiano se il costrutto accetta una proprietà denominata scope o id.

In Python, gli oggetti di scena sono espressi come argomenti di parole chiave. Se un argomento contiene strutture di dati annidate, queste vengono espresse utilizzando una classe che utilizza i propri argomenti di parole chiave al momento dell'istanziazione. Lo stesso schema viene applicato ad altre chiamate di metodo che accettano un argomento strutturato.

Ad esempio, nel `add_lifecycle_rule` metodo di un bucket Amazon S3, la `transitions` proprietà è un elenco di istanze. `Transition`

```
bucket.add_lifecycle_rule(
  transitions=[
    Transition(
      storage_class=StorageClass.GLACIER,
      transition_after=Duration.days(10)
    )
  ]
```

```
)
```

Quando estendi una classe o sovrascrivi un metodo, potresti voler accettare argomenti aggiuntivi per i tuoi scopi che non sono compresi dalla classe madre. In questo caso dovreste accettare gli argomenti che non vi interessano usando l'`**kwargs`idioma e usare argomenti basati solo su parole chiave per accettare gli argomenti che vi interessano. Quando chiamate il costruttore del genitore o il metodo sovrascritto, passate solo gli argomenti che si aspetta (spesso solo). `**kwargs` Il passaggio di argomenti che la classe o il metodo principale non si aspetta genera un errore.

```
class MyConstruct(Construct):
    def __init__(self, id, *, MyProperty=42, **kwargs):
        super().__init__(self, id, **kwargs)
        # ...
```

Le future versioni di AWS CDK potrebbero aggiungere casualmente una nuova proprietà con un nome che hai usato per la tua proprietà. Ciò non causerà alcun problema tecnico agli utenti del costrutto o del metodo (poiché la proprietà non viene passata «all'inizio della catena», la classe principale o il metodo sovrascritto utilizzeranno semplicemente un valore predefinito) ma potrebbe causare confusione. Potete evitare questo potenziale problema denominando le proprietà in modo che appartengano chiaramente al vostro costrutto. Se sono presenti molte nuove proprietà, raggruppatele in una classe dal nome appropriato e passatela come argomento di una singola parola chiave.

Valori mancanti

Gli AWS CDK usi `None` per rappresentare valori mancanti o non definiti. Quando si lavora con`**kwargs`, utilizzate il `get()` metodo del dizionario per fornire un valore predefinito se non viene fornita una proprietà. Evitate di utilizzarlo`kwargs[...]`, poiché ciò genera `KeyError` valori mancanti.

```
encrypted = kwargs.get("encrypted")         # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)  # specify default of False if property is
 missing
```

È possibile che vengano restituiti alcuni AWS CDK metodi (`tryGetContext()`ad esempio per ottenere un valore di contesto di runtime)`None`, che dovrete verificare esplicitamente.

Utilizzo delle interfacce

Python non ha una funzionalità di interfaccia come altri linguaggi, sebbene abbia classi base astratte, che sono simili. (Se non hai familiarità con le interfacce, Wikipedia ha una buona introduzione.) TypeScript, il linguaggio in cui AWS CDK è implementato, fornisce interfacce e i costrutti e altri AWS CDK oggetti spesso richiedono un oggetto che aderisca a una particolare interfaccia, anziché ereditare da una particolare classe. Quindi AWS CDK fornisce la propria funzionalità di interfaccia come parte del livello JSII.

Per indicare che una classe implementa una particolare interfaccia, puoi usare il decoratore: `@jsii.implements`

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Digita le insidie

Python utilizza la tipizzazione dinamica, in cui tutte le variabili possono fare riferimento a un valore di qualsiasi tipo. I parametri e i valori restituiti possono essere annotati con tipi, ma questi sono «suggerimenti» e non vengono applicati. Ciò significa che in Python è facile passare il tipo di valore errato a un AWS CDK costrutto. Invece di ricevere un errore di tipo durante la compilazione, come faresti con un linguaggio tipizzato staticamente, potresti invece ricevere un errore di runtime quando il livello JSII (che traduce tra Python e il TypeScript core) non AWS CDKè in grado di gestire il tipo imprevisto.

In base alla nostra esperienza, gli errori di tipo commessi dai programmatori Python tendono a rientrare in queste categorie.

- Passare un singolo valore in cui un costrutto prevede un contenitore (elenco o dizionario Python) o viceversa.
- Passare un valore di un tipo associato a un costrutto di livello 1 (`CfnXxxxxx`) a un costrutto L2 o L3 o viceversa.

I moduli AWS CDK Python includono annotazioni di tipo, quindi puoi usare strumenti che li supportano per aiutarti con i tipi. Se non utilizzate un IDE che li supporti, ad esempio, potreste voler chiamare il validatore dei MyPytipi come fase del processo di compilazione. PyCharm Esistono anche correttori del tipo di runtime che possono migliorare i messaggi di errore per gli errori relativi ai tipi.

Sintetizzazione e distribuzione

Gli stack definiti nell' AWS CDK app possono essere sintetizzati e distribuiti singolarmente o insieme utilizzando i comandi seguenti. In genere, dovresti trovarti nella directory principale del tuo progetto quando li pubblichi.

- `cdk synth`: sintetizza un AWS CloudFormation modello da uno o più stack dell'app. AWS CDK
- `cdk deploy`: distribuisce le risorse definite da uno o più stack dell'app su. AWS CDK AWS

È possibile specificare i nomi di più stack da sintetizzare o distribuire in un unico comando. Se l'app definisce solo uno stack, non è necessario specificarlo.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Puoi anche usare i caratteri jolly * (qualsiasi numero di caratteri) e? (qualsiasi carattere singolo) per identificare le pile in base al modello. Quando usate i caratteri jolly, racchiudete il pattern tra virgolette. Altrimenti, la shell potrebbe provare ad espanderlo ai nomi dei file nella directory corrente prima che vengano passati al AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Non è necessario sintetizzare esplicitamente gli stack prima di distribuirli; `cdk deploy` esegue questo passaggio per assicurarsi che venga distribuito il codice più recente.

Per la documentazione completa del comando, consulta. `cdk` the section called “AWS CDK Kit di strumenti”

Lavorare con Java AWS CDK

Java è un linguaggio client completamente supportato per AWS CDK ed è considerato stabile. Puoi sviluppare AWS CDK applicazioni in Java utilizzando strumenti familiari, tra cui JDK (Oracle o una distribuzione OpenJDK come Amazon Corretto) e Apache Maven.

Supporta Java 8 e versioni successive. AWS CDK Tuttavia, consigliamo di utilizzare la versione più recente possibile, poiché le versioni successive del linguaggio includono miglioramenti particolarmente utili per lo sviluppo di AWS CDK applicazioni. Ad esempio, Java 9 introduce il `Map.of()` metodo (un modo conveniente per dichiarare hashmap che verrebbero scritte come oggetti letterali). TypeScript Java 10 introduce l'inferenza di tipo locale utilizzando la parola chiave. `var`

> **Note**
>
> La maggior parte degli esempi di codice in questa Guida per sviluppatori funziona con Java 8. Alcuni esempi di utilizzo`Map.of()`; questi esempi includono commenti che indicano che richiedono Java 9.

Puoi usare qualsiasi editor di testo o un IDE Java in grado di leggere i progetti Maven per lavorare sulle tue AWS CDK app. Forniamo suggerimenti per Eclipse in questa guida, ma IntelliJ IDEA e altri IDE possono importare progetti Maven e possono essere utilizzati per lo sviluppo di applicazioni in Java. NetBeans AWS CDK

È possibile scrivere AWS CDK applicazioni in linguaggi ospitati da JVM diversi da Java (ad esempio, Kotlin, Groovy, Clojure o Scala), ma l'esperienza potrebbe non essere particolarmente idiomatica e non siamo in grado di fornire alcun supporto per questi linguaggi.

Argomenti

Inizia a usare Java

Per utilizzare, è necessario disporre di un account e delle AWS CDK credenziali e aver installato Node.js e il Toolkit AWS . AWS CDK Per informazioni, consulta Iniziare con AWS CDK.

AWS CDK Le applicazioni Java richiedono Java 8 (v1.8) o versione successiva. Consigliamo Amazon Corretto, ma puoi utilizzare qualsiasi distribuzione OpenJDK o JDK di Oracle. Avrai anche bisogno di Apache Maven 3.5 o versione successiva. Puoi anche usare strumenti come Gradle, ma gli scheletri delle applicazioni generati dal Toolkit sono progetti Maven. AWS CDK

> **Note**
>
> Deprecazione linguistica di terze parti: la versione linguistica è supportata solo fino alla fine del ciclo di vita (EOL (End Of Life) condivisa dal fornitore o dalla community ed è soggetta a modifiche con preavviso.

Creare un progetto

È possibile creare un nuovo AWS CDK progetto `cdk init` richiamandolo in una directory vuota. Utilizzate l'`--language`opzione e specificate`java`:

```
mkdir my-project
cd my-project
cdk init app --language java
```

`cdk init`utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. I trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Tuttavia, il nome dovrebbe altrimenti assumere la forma di un identificatore Java; ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Il progetto risultante include un riferimento al pacchetto `software.amazon.awscdk` Maven. Esso e le sue dipendenze vengono installati automaticamente da Maven.

Se utilizzi un IDE, ora puoi aprire o importare il progetto. In Eclipse, ad esempio, scegli File > Importa > Maven > Progetti Maven esistenti. Assicuratevi che le impostazioni del progetto siano impostate per utilizzare Java 8 (1.8).

Gestione dei AWS moduli Construct Library

Usa Maven per installare i pacchetti AWS Construct Library, che fanno parte del gruppo. `software.amazon.awscdk` La maggior parte dei costrutti si trova nell'artefatto`aws-cdk-lib`, che viene aggiunto ai nuovi progetti Java per impostazione predefinita. I moduli per i servizi il cui supporto CDK di livello superiore è ancora in fase di sviluppo si trovano in pacchetti «sperimentali» separati, denominati con una versione breve (no o prefisso AWS Amazon) del nome del servizio. Cerca nel Maven Central Repository per trovare i nomi di tutte le librerie e di Construct Module. AWS CDK AWS

> **Note**
>
> L'edizione Java del CDK API Reference mostra anche i nomi dei pacchetti.

Il supporto di AWS Construct Library di alcuni servizi è disponibile in più di un namespace. Ad esempio, Amazon Route 53 ha le sue funzionalità suddivise in `software.amazon.awscdk.route53route53-patterns`,`route53resolver`, e`route53-targets`.

Il AWS CDK pacchetto principale viene importato in codice Java come`software.amazon.awscdk`. I moduli per i vari servizi della AWS Construct Library risiedono in `software.amazon.awscdk.services` e hanno un nome simile al nome del pacchetto Maven. Ad esempio, lo spazio dei nomi del modulo Amazon S3 è. `software.amazon.awscdk.services.s3`

Ti consigliamo di scrivere un'`import`istruzione Java separata per ogni classe di AWS Construct Library che usi in ciascuno dei tuoi file sorgente Java ed evitare le importazioni con caratteri jolly. Puoi sempre utilizzare il nome completo di un tipo (incluso il relativo spazio dei nomi) senza un'istruzione. `import`

Se la tua applicazione dipende da un pacchetto sperimentale, modifica il progetto `pom.xml` e aggiungi un nuovo `<dependency>` elemento nel contenitore. `<dependencies>` Ad esempio, il seguente `<dependency>` elemento specifica il modulo della libreria di costruzioni CodeStar sperimentali:

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>codestar-alpha</artifactId>
    <version>2.0.0-alpha.10</version>
</dependency>
```

> **Tip**
>
> Se si utilizza un IDE Java, probabilmente dispone di funzionalità per la gestione delle dipendenze di Maven. Tuttavia, ti consigliamo di modificare `pom.xml` direttamente, a meno che tu non sia assolutamente sicuro che la funzionalità dell'IDE corrisponda a quella che faresti a mano.

Gestione delle dipendenze in Java

In Java, le dipendenze vengono specificate `pom.xml` e installate utilizzando Maven. Il `<dependencies>` contenitore include un `<dependency>` elemento per ogni pacchetto. Di seguito è riportata una sezione `pom.xml` di una tipica app Java 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**
>
> Molti IDE Java hanno integrato il supporto Maven e `pom.xml` gli editor visivi, che potresti trovare utili per la gestione delle dipendenze.

Maven non supporta il blocco delle dipendenze. Sebbene sia possibile specificare intervalli di versioni`pom.xml`, ti consigliamo di utilizzare sempre versioni esatte per mantenere le build ripetibili.

Maven installa automaticamente le dipendenze transitive, ma può esserci solo una copia installata di ogni pacchetto. Viene selezionata la versione specificata più in alto nell'albero POM; le applicazioni hanno sempre l'ultima parola sulla versione dei pacchetti da installare.

Maven installa o aggiorna automaticamente le tue dipendenze ogni volta che crei (mvn compile) o pacchettizzi () il tuo progetto. mvn package CDK Toolkit lo fa automaticamente ogni volta che lo eseguite, quindi in genere non è necessario richiamare manualmente Maven.

AWS CDK idiomi in Java

oggetti di scena

Tutte le classi di AWS Construct Library vengono istanziate utilizzando tre argomenti: l'ambito in cui viene definito il costrutto (l'elemento principale nell'albero dei costrutti), un id e props, un insieme di coppie chiave/valore che il costrutto utilizza per configurare le risorse che crea. Anche altre classi e metodi utilizzano il modello «bundle of attributes» per gli argomenti.

In Java, gli oggetti di scena vengono espressi utilizzando il pattern Builder. Ogni tipo di costrutto ha un tipo di oggetto corrispondente; ad esempio, il `Bucket` costrutto (che rappresenta un bucket Amazon S3) prende come oggetto di scena un'istanza di. `BucketProps`

La `BucketProps` classe (come ogni classe props di AWS Construct Library) ha una classe interna chiamata. `Builder` Il `BucketProps.Builder` tipo offre metodi per impostare le varie proprietà di un'`BucketProps`istanza. Ogni metodo restituisce l'`Builder`istanza, quindi le chiamate al metodo possono essere concatenate per impostare più proprietà. Alla fine della catena, si chiama `build()` per produrre effettivamente l'`BucketProps`oggetto.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build());
```

I costrutti e le altre classi che utilizzano un oggetto simile a un oggetto simile a un oggetto props come argomento finale offrono una scorciatoia. La classe ha una `Builder` propria istanza che la istanzia e il relativo oggetto props in un unico passaggio. In questo modo, non è necessario istanziare esplicitamente (ad esempio) entrambi `BucketProps` e un, e non è necessaria un'`Bucket`importazione per il tipo props.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build();
```

Quando derivate il vostro costrutto da un costrutto esistente, potreste voler accettare proprietà aggiuntive. Ti consigliamo di seguire questi schemi di costruzione. Tuttavia, non è così semplice come sottoclassare una classe di costrutti. È necessario fornire personalmente le parti mobili delle due nuove `Builder` classi. Potresti preferire che il tuo costrutto accetti semplicemente uno o più argomenti aggiuntivi. È necessario fornire costruttori aggiuntivi quando un argomento è facoltativo.

Strutture generiche

In alcune API, AWS CDK utilizza JavaScript matrici o oggetti non tipizzati come input per un metodo. (Vedi, ad esempio, AWS CodeBuild il metodo.) `BuildSpec.fromObject()` In Java, questi oggetti sono rappresentati come`java.util.Map<String, Object>`. Nei casi in cui i valori sono tutte stringhe, puoi usare`Map<String, String>`.

Java non fornisce un modo per scrivere valori letterali per tali contenitori come fanno altri linguaggi. In Java 9 e versioni successive, è possibile `java.util.Map.of()`definire comodamente mappe con un massimo di dieci voci in linea con una di queste chiamate.

```
java.util.Map.of(
    "base-directory", "dist",
    "files", "LambdaStack.template.json"
 )
```

Per creare mappe con più di dieci voci, usa. `java.util.Map.ofEntries()`

Se utilizzi Java 8, potresti fornire metodi personalizzati simili a questi.

JavaScript gli array sono rappresentati come `List<Object>` o `List<String>` in Java. Il metodo `java.util.Arrays.asList` è utile per definire `List` s brevi.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

Valori mancanti

In Java, i valori mancanti in AWS CDK oggetti come gli oggetti di scena sono rappresentati da`null`. È necessario testare esplicitamente qualsiasi valore possibile `null` per assicurarsi che contenga un valore prima di utilizzarlo. Java non ha uno «zucchero sintattico» per aiutare a gestire i valori nulli come fanno altri linguaggi. Potresti trovare Apache ObjectUtil defaultIfNulle firstNonNullutili in alcune situazioni. In alternativa, scrivete i vostri metodi di supporto statici per semplificare la gestione di valori potenzialmente nulli e rendere il codice più leggibile.

Creazione, sintesi e distribuzione

Compila AWS CDK automaticamente l'app prima di eseguirla. Tuttavia, può essere utile creare l'app manualmente per verificare la presenza di errori ed eseguire test. Puoi farlo nel tuo IDE (ad esempio, premi Control-B in Eclipse) o eseguendo l'operazione `mvn compile` al prompt dei comandi mentre ti trovi nella directory principale del progetto.

Esegui tutti i test che hai scritto eseguendoli al prompt dei comandi`mvn test`.

Gli stack definiti nell' AWS CDK app possono essere sintetizzati e distribuiti singolarmente o insieme utilizzando i comandi seguenti. In genere, dovresti trovarti nella directory principale del tuo progetto quando li pubblichi.

- `cdk synth`: sintetizza un AWS CloudFormation modello da uno o più stack dell'app. AWS CDK
- `cdk deploy`: distribuisce le risorse definite da uno o più stack dell'app su. AWS CDK AWS

È possibile specificare i nomi di più stack da sintetizzare o distribuire in un unico comando. Se l'app definisce solo uno stack, non è necessario specificarlo.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Puoi anche usare i caratteri jolly * (qualsiasi numero di caratteri) e? (qualsiasi carattere singolo) per identificare le pile in base allo schema. Quando usate i caratteri jolly, racchiudete il pattern tra virgolette. Altrimenti, la shell potrebbe provare ad espanderlo ai nomi dei file nella directory corrente prima che vengano passati al AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Non è necessario sintetizzare esplicitamente gli stack prima di distribuirli; `cdk deploy` esegue questo passaggio per assicurarsi che venga distribuito il codice più recente.

Per la documentazione completa del comando, consulta. `cdk` the section called “AWS CDK Kit di strumenti”

Lavorare con il AWS CDK in C#

.NET è un linguaggio client completamente supportato per AWS CDK ed è considerato stabile. C# è il principale linguaggio.NET per il quale forniamo esempi e supporto. È possibile scegliere di scrivere AWS CDK applicazioni in altri linguaggi.NET, come Visual Basic o F#, ma AWS offre un supporto limitato per l'utilizzo di questi linguaggi con CDK.

È possibile sviluppare AWS CDK applicazioni in C# utilizzando strumenti familiari tra cui Visual Studio, Visual Studio Code, il `dotnet` comando e il NuGet gestore di pacchetti. I moduli che compongono la AWS Construct Library sono distribuiti tramite nuget.org.

Ti consigliamo di utilizzare Visual Studio 2019 (qualsiasi edizione) su Windows per sviluppare app in C#. AWS CDK

Argomenti

- Inizia a usare C#
- Creare un progetto
- Gestione dei moduli di AWS Construct Library
- Gestione delle dipendenze in C#
- AWS CDK idiomi in C#
- Creazione, sintesi e implementazione

Inizia a usare C#

Per utilizzare AWS CDK, è necessario disporre di un AWS account e delle credenziali e aver installato Node.js e il AWS CDK Toolkit. Per informazioni, consulta Iniziare con AWS CDK.

AWS CDK Le applicazioni C# richiedono .NET Core v3.1 o versione successiva, disponibile qui.

La toolchain .NET include `dotnet` uno strumento da riga di comando per la creazione e l'esecuzione di applicazioni.NET e la gestione dei pacchetti. NuGet Anche se lavori principalmente in Visual Studio, questo comando può essere utile per le operazioni in batch e per l'installazione dei pacchetti AWS Construct Library.

Creare un progetto

È possibile creare un nuovo AWS CDK progetto invocandolo `cdk init` in una directory vuota. Utilizzate l'`--language`opzione e specificate`csharp`:

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init`utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. I trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Tuttavia, il nome dovrebbe altrimenti assumere la forma di un identificatore C#; ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Il progetto risultante include un riferimento al `Amazon.CDK.Lib` NuGet pacchetto. Esso e le sue dipendenze vengono installati automaticamente da NuGet.

Gestione dei moduli di AWS Construct Library

L'ecosistema.NET utilizza il gestore di NuGet pacchetti. Il pacchetto CDK principale, che contiene le classi principali e tutti i costrutti di servizio stabili, è. `Amazon.CDK.Lib` I moduli sperimentali, in cui le nuove funzionalità sono in fase di sviluppo attivo`Amazon.CDK.AWS.`*`SERVICE-NAME`*`.Alpha`, hanno nomi simili, dove il nome del servizio è un nome breve senza prefisso AWS o Amazon. Ad esempio, il nome NuGet del pacchetto per il AWS IoT modulo è`Amazon.CDK.AWS.IoT.Alpha`. Se non riesci a trovare il pacchetto che desideri, cerca su NuGet.org.

> **Note**
>
> L'editione.NET del CDK API Reference mostra anche i nomi dei pacchetti.

Il supporto di AWS Construct Library di alcuni servizi è disponibile in più di un modulo. Ad esempio, AWS IoT ha un secondo modulo denominato. `Amazon.CDK.AWS.IoT.Actions.Alpha`

Il AWS CDK modulo principale, che ti servirà nella maggior parte delle AWS CDK app, viene importato nel codice C# come`Amazon.CDK`. I moduli per i vari servizi della AWS Construct Library sono disponibili in. `Amazon.CDK.AWS` Ad esempio, lo spazio dei nomi del modulo Amazon S3 è. `Amazon.CDK.AWS.S3`

Ti consigliamo di scrivere `using` direttive C# per i costrutti principali di CDK e per ogni AWS servizio che usi in ciascuno dei tuoi file sorgente C#. Potrebbe essere utile utilizzare un alias per un namespace o un tipo per risolvere i conflitti tra nomi. Puoi sempre utilizzare il nome completo di un tipo (incluso lo spazio dei nomi) senza una dichiarazione. `using`

Gestione delle dipendenze in C#

Nelle AWS CDK app C#, gestisci le dipendenze utilizzando. NuGet NuGet ha quattro interfacce standard, per lo più equivalenti. Usa quella più adatta alle tue esigenze e al tuo stile di lavoro. Puoi anche utilizzare strumenti compatibili, come Paket MyGeto persino modificare direttamente il `.csproj` file.

NuGet non consente di specificare intervalli di versioni per le dipendenze. Ogni dipendenza è associata a una versione specifica.

Dopo aver aggiornato le dipendenze, Visual Studio le utilizzerà NuGet per recuperare le versioni specificate di ogni pacchetto alla successiva compilazione. Se non usi Visual Studio, usa il dotnet restore comando per aggiornare le dipendenze.

Modifica diretta del file di progetto

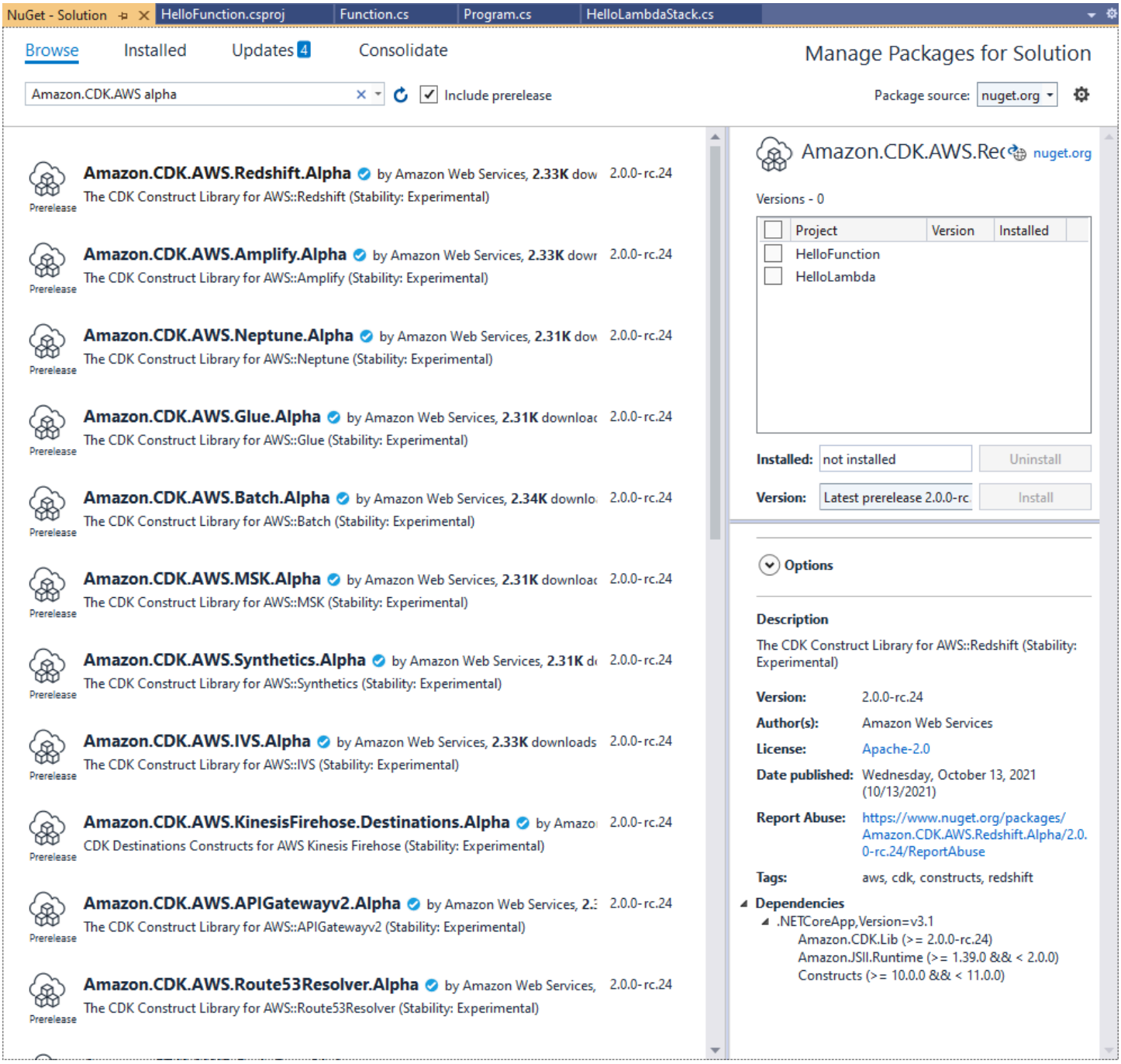
Il `.csproj` file del progetto contiene un `<ItemGroup>` contenitore che elenca le dipendenze come `<PackageReference` elementi.

```
<ItemGroup>
    <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
    <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

La GUI di Visual Studio NuGet

NuGet Gli strumenti di Visual Studio sono accessibili da Tools > NuGet Package Manager > Manage NuGet Packages for Solution. Utilizza la scheda Sfoglia per trovare i pacchetti AWS Construct Library che desideri installare. Potete scegliere la versione desiderata, comprese le versioni non definitive dei moduli, e aggiungerla a qualsiasi progetto aperto.

> **Note**
>
> Tutti i moduli di AWS Construct Library considerati «sperimentali» (vedithe section called “Controllo delle versioni”) sono contrassegnati come pre-release in e hanno un suffisso di nome. NuGet `alpha`



Consultate la pagina Aggiornamenti per installare nuove versioni dei pacchetti.

La NuGet console

La NuGet console è un'interfaccia PowerShell basata su di NuGet essa che funziona nel contesto di un progetto di Visual Studio. Puoi aprirlo in Visual Studio scegliendo Strumenti > NuGet Package Manager > Package Manager Console. Per ulteriori informazioni sull'utilizzo di questo strumento, vedi Installare e gestire i pacchetti con la console Package Manager in Visual Studio.

Il **dotnet** comando

Il `dotnet` comando è lo strumento da riga di comando principale per lavorare con i progetti di Visual Studio C#. È possibile richiamarlo da qualsiasi prompt dei comandi di Windows. Tra le sue numerose funzionalità, `dotnet` può aggiungere NuGet dipendenze a un progetto di Visual Studio.

Supponendo che ti trovi nella stessa directory del file di progetto di Visual Studio (`.csproj`), esegui un comando come il seguente per installare un pacchetto. Poiché la libreria CDK principale è inclusa quando crei un progetto, devi solo installare esplicitamente i moduli sperimentali. I moduli sperimentali richiedono di specificare un numero di versione esplicito.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

È possibile eseguire il comando da un'altra directory. A tale scopo, includi il percorso del file di progetto o della directory che lo contiene dopo la `add` parola chiave. L'esempio seguente presuppone che vi troviate nella directory principale del AWS CDK progetto.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Per installare una versione specifica di un pacchetto, includi il `-v` flag e la versione desiderata.

Per aggiornare un pacchetto, esegui lo stesso `dotnet add` comando che hai usato per installarlo. Per i moduli sperimentali, ancora una volta, è necessario specificare un numero di versione esplicito.

Per ulteriori informazioni sulla gestione dei pacchetti tramite il `dotnet` comando, consulta Installare e gestire i pacchetti utilizzando la CLI dotnet.

Il comando **nuget**

Lo strumento da riga di `nuget` comando può installare e aggiornare NuGet i pacchetti. Tuttavia, richiede che il progetto di Visual Studio sia configurato in modo diverso dal modo in cui `cdk init` vengono configurati i progetti. (Dettagli tecnici: `nuget` funziona con i `Packages.config` progetti, mentre `cdk init` crea un `PackageReference` progetto di nuovo stile.)

Si sconsiglia l'uso dello `nuget` strumento con AWS CDK progetti creati da. `cdk init` Se stai utilizzando un altro tipo di progetto e desideri utilizzarlo`nuget`, consulta la NuGet CLI Reference.

AWS CDK idiomi in C#

Oggetti di scena

Tutte le classi di AWS Construct Library vengono istanziate utilizzando tre argomenti: l'ambito in cui viene definito il costrutto (l'elemento principale nell'albero dei costrutti), un id e props, un insieme di coppie chiave/valore che il costrutto utilizza per configurare le risorse che crea. Anche altre classi e metodi utilizzano il modello «bundle of attributes» per gli argomenti.

In C#, gli oggetti di scena vengono espressi utilizzando un tipo di oggetti di scena. In stile C# idiomatico, possiamo usare un inizializzatore di oggetti per impostare le varie proprietà. Qui stiamo creando un bucket Amazon S3 usando il `Bucket` costrutto; il tipo di oggetto corrispondente è. `BucketProps`

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
    Versioned = true
});
```

> **Tip**
>
> Aggiungi il pacchetto `Amazon.JSII.Analyzers` al tuo progetto per ottenere i valori richiesti controllando le definizioni dei props all'interno di Visual Studio.

Quando estendi una classe o sovrascrivi un metodo, potresti voler accettare oggetti di scena aggiuntivi per i tuoi scopi che non sono compresi dalla classe principale. Per fare ciò, aggiungete una sottoclasse agli oggetti di scena appropriati e aggiungete i nuovi attributi.

```
// 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"
});
```

Quando chiamate l'inizializzatore o il metodo sovrascritto della classe principale, in genere potete passare gli oggetti di scena che avete ricevuto. Il nuovo tipo è compatibile con il suo genitore e gli oggetti di scena aggiuntivi aggiunti vengono ignorati.

Le future versioni di AWS CDK potrebbero aggiungere casualmente una nuova proprietà con un nome che hai usato per la tua proprietà. Ciò non causerà alcun problema tecnico nell'utilizzo del costrutto o del metodo (poiché la proprietà non viene trasmessa «all'inizio della catena», la classe principale o il metodo sovrascritto utilizzeranno semplicemente un valore predefinito) ma potrebbe creare confusione tra gli utenti del costrutto. Puoi evitare questo potenziale problema denominando le tue proprietà in modo che appartengano chiaramente al tuo costrutto. Se sono presenti molte nuove proprietà, raggruppatele in una classe dal nome appropriato e passatele come un'unica proprietà.

Strutture generiche

In alcune API, AWS CDK utilizza JavaScript matrici o oggetti non tipizzati come input per un metodo. (Vedi, ad esempio, AWS CodeBuild il metodo.) `BuildSpec.fromObject()` In C#, questi oggetti sono rappresentati come`System.Collections.Generic.Dictionary<String, Object>`. Nei casi in cui i valori sono tutte stringhe, puoi usare. `Dictionary<String, String>` JavaScript gli array sono rappresentati come `object[]` o tipi di `string[]` array in C#.

> **Tip**
>
> È possibile definire alias brevi per semplificare l'utilizzo di questi tipi di dizionario specifici.
>
> ```
> using StringDict = System.Collections.Generic.Dictionary<string, string>;
> using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
> ```

Valori mancanti

In C#, i valori mancanti in AWS CDK oggetti come props sono rappresentati da. `null` L'operatore null-conditional member access `?.` e l'operatore null coalescente sono utili per lavorare con questi valori. `??`

```
// 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";
```

Creazione, sintesi e implementazione

Compila AWS CDK automaticamente l'app prima di eseguirla. Tuttavia, può essere utile creare l'app manualmente per verificare la presenza di errori ed eseguire test. Puoi farlo premendo F6 in Visual Studio o eseguendo `dotnet build src` dalla riga di comando, dove si `src` trova la directory nella directory del progetto che contiene il file Visual Studio Solution (`.sln`).

Gli stack definiti nell' AWS CDK app possono essere sintetizzati e distribuiti singolarmente o insieme utilizzando i comandi seguenti. In genere, dovresti trovarti nella directory principale del tuo progetto quando li pubblichi.

- `cdk synth`: sintetizza un AWS CloudFormation modello da uno o più stack dell'app. AWS CDK
- `cdk deploy`: distribuisce le risorse definite da uno o più stack dell'app su. AWS CDK AWS

È possibile specificare i nomi di più stack da sintetizzare o distribuire in un unico comando. Se l'app definisce solo uno stack, non è necessario specificarlo.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Puoi anche usare i caratteri jolly * (qualsiasi numero di caratteri) e? (qualsiasi carattere singolo) per identificare le pile in base al modello. Quando usate i caratteri jolly, racchiudete il pattern tra virgolette. Altrimenti, la shell potrebbe provare ad espanderlo ai nomi dei file nella directory corrente prima che vengano passati al AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
```

```
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Non è necessario sintetizzare esplicitamente gli stack prima di distribuirli; `cdk deploy` esegue questo passaggio per assicurarsi che venga distribuito il codice più recente.

Per la documentazione completa del comando, consulta. `cdk` the section called “AWS CDK Kit di strumenti”

Lavorare con AWS CDK in Go

Go è un linguaggio client completamente supportato per AWS Cloud Development Kit (AWS CDK) ed è considerato stabile. Lavorare con AWS CDK in Go utilizza strumenti familiari. La versione Go di AWS CDK Even utilizza identificatori in stile Go.

A differenza degli altri linguaggi supportati dal CDK, Go non è un linguaggio di programmazione tradizionale orientato agli oggetti. Go utilizza la composizione laddove altri linguaggi spesso sfruttano l'ereditarietà. Abbiamo cercato di utilizzare il più possibile approcci Go idiomatici, ma ci sono punti in cui il CDK può differire.

Questo argomento fornisce una guida per l'utilizzo di in Go. AWS CDK Consulta il post sul blog dedicato all'annuncio per una descrizione dettagliata di un semplice progetto Go per il. AWS CDK

Argomenti

- Inizia a usare Go
- Creare un progetto
- Gestione dei moduli AWS di Construct Library
- Gestione delle dipendenze in Go
- AWS CDK idiomi in Go
- Creazione, sintesi e distribuzione

Inizia a usare Go

Per utilizzare AWS CDK, è necessario disporre di un AWS account e delle credenziali e aver installato Node.js e il Toolkit. AWS CDK Per informazioni, consulta Iniziare con AWS CDK.

I collegamenti Go AWS CDK utilizzano la toolchain Go standard, v1.18 o successiva. Puoi usare l'editor che preferisci.

> **Note**
>
> Lingua obsoleta di terze parti: la versione linguistica è supportata solo fino alla fine del ciclo di vita (EOL (End Of Life) condivisa dal fornitore o dalla community ed è soggetta a modifiche con preavviso.

Creare un progetto

È possibile creare un nuovo AWS CDK progetto `cdk init` richiamandolo in una directory vuota. Utilizzate l'`--language`opzione e specificatego:

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init`utilizza il nome della cartella del progetto per denominare vari elementi del progetto, tra cui classi, sottocartelle e file. I trattini nel nome della cartella vengono convertiti in caratteri di sottolineatura. Tuttavia, il nome dovrebbe altrimenti assumere la forma di un identificatore Go; ad esempio, non dovrebbe iniziare con un numero o contenere spazi.

Il progetto risultante include un riferimento al modulo AWS CDK Go principale,`github.com/aws/aws-cdk-go/awscdk/v2`, in`go.mod`. Problema go get per installare questo e altri moduli richiesti.

Gestione dei moduli AWS di Construct Library

Nella maggior parte della AWS CDK documentazione e degli esempi, la parola «modulo» è spesso usata per riferirsi ai moduli di AWS Construct Library, uno o più per AWS servizio, il che differisce dall'uso idiomatico del termine in Go. La CDK Construct Library è fornita in un modulo Go con i singoli moduli Construct Library, che supportano i vari AWS servizi, forniti come pacchetti Go all'interno di quel modulo.

Il supporto per AWS Construct Library di alcuni servizi è contenuto in più di un modulo Construct Library (pacchetto Go). Ad esempio, Amazon Route 53 dispone di tre moduli Construct Library oltre al `awsroute53` pacchetto principale, named `awsroute53patternsawsroute53resolver`, and`awsroute53targets`.

Il AWS CDK pacchetto principale, necessario nella maggior parte delle AWS CDK app, viene importato in Go code as`github.com/aws/aws-cdk-go/awscdk/v2`. I pacchetti per i vari servizi della AWS Construct Library sono disponibili in`github.com/aws/aws-cdk-go/awscdk/v2`. Ad esempio, lo spazio dei nomi del modulo Amazon S3 è. `github.com/aws/aws-cdk-go/awscdk/v2/awss3`

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

Dopo aver importato i moduli Construct Library (pacchetti Go) per i servizi che desideri utilizzare nella tua app, accedi ai costrutti di quel modulo utilizzando, ad esempio,. `awss3.Bucket`

Gestione delle dipendenze in Go

In Go, le versioni delle dipendenze sono definite in. `go.mod` L'impostazione predefinita `go.mod` è simile a quella mostrata qui.

```
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
)
```

I nomi dei Package (moduli, in gergo Go) sono specificati tramite URL con il numero di versione richiesto aggiunto. Il sistema di moduli di Go non supporta intervalli di versioni.

Emetti il go get comando per installare tutti i moduli richiesti e aggiornarligo.mod. Per visualizzare un elenco degli aggiornamenti disponibili per le tue dipendenze, invia. go list -m -u all

AWS CDK idiomi in Go

Nomi di campi e metodi

I nomi dei campi e dei metodi utilizzano camel casing (`likeThis`) in TypeScript, la lingua di origine del CDK. In Go, questi seguono le convenzioni Go, così come Pascal cased (). `LikeThis`

Pulizia

Nel tuo `main` metodo, usalo `defer jsii.Close()` per assicurarti che l'app CDK si pulisca da sola.

Valori mancanti e conversione del puntatore

In Go, i valori mancanti in AWS CDK oggetti come i pacchetti di proprietà sono rappresentati da. `nil` Go non ha tipi nullabili; l'unico tipo che può contenere `nil` è un puntatore. Per consentire ai valori di essere opzionali, quindi, tutte le proprietà CDK, gli argomenti e i valori restituiti sono puntatori, anche per i tipi primitivi. Questo vale sia per i valori obbligatori che per quelli facoltativi, quindi se un valore richiesto diventa successivamente facoltativo, non è necessario modificare radicalmente il tipo.

Quando passate valori o espressioni letterali, utilizzate le seguenti funzioni di supporto per creare puntatori ai valori.

- `jsii.String`
- `jsii.Number`
- `jsii.Bool`
- `jsii.Time`

Per motivi di coerenza, si consiglia di utilizzare i puntatori in modo simile quando si definiscono i propri costrutti, anche se può sembrare più comodo, ad esempio, ricevere i costrutti `id` come stringa anziché puntatore a una stringa.

Quando avete a che fare con AWS CDK valori opzionali, inclusi valori primitivi e tipi complessi, dovreste testare esplicitamente i puntatori per assicurarvi che non lo siano prima di fare qualcosa con essi. `nil` Go non ha «zucchero sintattico» per aiutare a gestire i valori vuoti o mancanti come fanno altri linguaggi. Tuttavia, è garantita l'esistenza dei valori richiesti nei pacchetti di proprietà e in strutture simili (altrimenti la costruzione fallisce), quindi non è necessario verificare questi valori. `nil`

Costrutti e oggetti di scena

I costrutti, che rappresentano una o più AWS risorse e i relativi attributi associati, sono rappresentati in Go come interfacce. Ad esempio, `awss3.Bucket` è un'interfaccia. Ogni costrutto ha una funzione di fabbrica`awss3.NewBucket`, ad esempio restituire una struttura che implementa l'interfaccia corrispondente.

Tutte le funzioni di fabbrica richiedono tre argomenti: quello `scope` in cui viene definito il costrutto (il suo elemento principale nell'albero dei costrutti)`id`, un e `props` un insieme di coppie chiave/valore che il costrutto utilizza per configurare le risorse che crea. Il modello «pacchetto di attributi» viene utilizzato anche altrove in. AWS CDK

In Go, gli oggetti di scena sono rappresentati da un tipo di struttura specifico per ogni costrutto. Ad esempio, an `awss3.Bucket` accetta un argomento props di tipo. `awss3.BucketProps` Usa una struttura letterale per scrivere argomenti props.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

Strutture generiche

In alcuni punti, AWS CDK utilizza JavaScript matrici o oggetti non tipizzati come input per un metodo. (Vedi, ad esempio, AWS CodeBuild il metodo.) `BuildSpec.fromObject()` In Go, questi oggetti sono rappresentati rispettivamente come sezioni e un'interfaccia vuota.

Il CDK fornisce funzioni di supporto variadiche, come `jsii.Strings` la creazione di sezioni contenenti tipi primitivi.

```
jsii.Strings("One", "Two", "Three")
```

Sviluppo di costrutti personalizzati

In Go, di solito è più semplice scrivere un nuovo costrutto che estenderne uno esistente. Innanzitutto, definisci un nuovo tipo di struttura, incorporando in modo anonimo uno o più tipi esistenti se desideri una semantica simile a un'estensione. Scrivi metodi per ogni nuova funzionalità che stai aggiungendo e i campi necessari per contenere i dati di cui hanno bisogno. Definisci un'interfaccia props se il tuo costrutto ne ha bisogno. Infine, scrivi una funzione di fabbrica `NewMyConstruct()` per restituire un'istanza del tuo costrutto.

Se state semplicemente modificando alcuni valori predefiniti su un costrutto esistente o aggiungete un comportamento semplice al momento dell'istanziazione, non avete bisogno di tutta quell'installazione idraulica. Invece, scrivi una funzione di fabbrica che richiami la funzione di fabbrica del costrutto che stai «estendendo». In altri linguaggi CDK, ad esempio, è possibile creare un `TypedBucket` costrutto che impone il tipo di oggetti in un bucket Amazon S3 sovrascrivendo il `s3.Bucket` tipo e, nell'inizializzatore del nuovo tipo, aggiungendo una policy di bucket che consenta di aggiungere al bucket solo estensioni di nomi di file specificate. In Go, è più semplice scrivere semplicemente un messaggio `NewTypedBucket` che restituisca una `s3.Bucket` (istanziata utilizzando`s3.NewBucket`) a cui è stata aggiunta una policy bucket appropriata. Non è necessario alcun nuovo tipo di costrutto perché la funzionalità è già disponibile nel costrutto bucket standard; il nuovo «costrutto» fornisce solo un modo più semplice per configurarlo.

Creazione, sintesi e distribuzione

Compila AWS CDK automaticamente l'app prima di eseguirla. Tuttavia, può essere utile creare l'app manualmente per verificare la presenza di errori ed eseguire test. Puoi farlo eseguendo il comando `go build` al prompt dei comandi mentre ti trovi nella directory principale del tuo progetto.

Esegui tutti i test che hai scritto eseguendoli `go test` al prompt dei comandi.

Gli stack definiti nell' AWS CDK app possono essere sintetizzati e distribuiti singolarmente o insieme utilizzando i comandi seguenti. In genere, dovresti trovarti nella directory principale del tuo progetto quando li pubblichi.

- `cdk synth`: sintetizza un AWS CloudFormation modello da uno o più stack dell'app. AWS CDK
- `cdk deploy`: distribuisce le risorse definite da uno o più stack dell'app su. AWS CDK AWS

È possibile specificare i nomi di più stack da sintetizzare o distribuire in un unico comando. Se l'app definisce solo uno stack, non è necessario specificarlo.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Puoi anche usare i caratteri jolly * (qualsiasi numero di caratteri) e? (qualsiasi carattere singolo) per identificare le pile in base allo schema. Quando usate i caratteri jolly, racchiudete il pattern tra virgolette. Altrimenti, la shell potrebbe provare ad espanderlo ai nomi dei file nella directory corrente prima che vengano passati al AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Non è necessario sintetizzare esplicitamente gli stack prima di distribuirli; `cdk deploy` esegue questo passaggio per assicurarsi che venga distribuito il codice più recente.

Per la documentazione completa del comando, consulta. `cdk` the section called “AWS CDK Kit di strumenti”

Sviluppo di AWS CDK applicazioni

Sviluppa AWS Cloud Development Kit (AWS CDK) applicazioni.

Argomenti

Personalizzazione dei costrutti dalla Construct Library AWS

Personalizza i costrutti della AWS Construct Library tramite escape hatch, sostituzioni raw e risorse personalizzate.

Argomenti

Usare le botole di fuga

La AWS Construct Library fornisce costrutti con diversi livelli di astrazione.

Al livello più alto, l' AWS CDK applicazione e gli stack in essa contenuti sono essi stessi astrazioni dell'intera infrastruttura cloud o parti significative di essa. Possono essere parametrizzati per distribuirli in ambienti diversi o per esigenze diverse.

Le astrazioni sono strumenti potenti per la progettazione e l'implementazione di applicazioni cloud. Ti AWS CDK dà il potere non solo di costruire con le sue astrazioni, ma anche di creare nuove astrazioni. Utilizzando i costrutti open source L2 e L3 esistenti come guida, è possibile creare costrutti L2 e L3 personalizzati che riflettano le migliori pratiche e le opinioni della propria organizzazione.

Nessuna astrazione è perfetta e anche le buone astrazioni non possono coprire tutti i possibili casi d'uso. Durante lo sviluppo, potresti trovare un costrutto che si adatta quasi alle tue esigenze, che richieda una personalizzazione piccola o grande.

Per questo motivo, AWS CDK fornisce modi per uscire dal modello di costruzione. Ciò include il passaggio a un'astrazione di livello inferiore o a un modello completamente diverso. Gli escape hatches ti consentono di sfuggire al AWS CDK paradigma e di personalizzarlo in base alle tue esigenze. Quindi, puoi racchiudere le modifiche in un nuovo costrutto per astrarre la complessità sottostante e fornire un'API pulita per altri sviluppatori.

Di seguito sono riportati alcuni esempi di situazioni in cui è possibile utilizzare le botole di fuga:

- Una funzionalità AWS di servizio è disponibile tramite AWS CloudFormation, ma non esistono costrutti L2 corrispondenti.
- Una funzionalità AWS di servizio è disponibile tramite AWS CloudFormation e sono presenti costrutti L2 per il servizio, ma questi non espongono ancora la funzionalità. Poiché i costrutti L2 sono curati dal team CDK, potrebbero non essere immediatamente disponibili per le nuove funzionalità.
- La funzionalità non è ancora disponibile affatto. AWS CloudFormation

  Per determinare se una funzionalità è disponibile tramite AWS CloudFormation, consulta AWS Resource and Property Types Reference.

Sviluppa portelli di fuga per i costrutti L1

Se i costrutti L2 non sono disponibili per il servizio, è possibile utilizzare i costrutti L1 generati automaticamente. Queste risorse possono essere riconosciute dal nome che inizia con`Cfn`, ad esempio o. `CfnBucket` `CfnRole` Le istanziate esattamente come usereste la risorsa equivalente AWS CloudFormation .

Ad esempio, per creare un'istanza di un bucket Amazon S3 di basso livello L1 con analisi abilitata, dovresti scrivere qualcosa come segue.

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",
        // ...
    }
});
```

Potrebbero esserci rari casi in cui desideri definire una risorsa che non ha una classe corrispondente. `CfnXxx` Potrebbe trattarsi di un nuovo tipo di risorsa che non è ancora stato pubblicato nelle specifiche delle AWS CloudFormation risorse. In casi come questo, è possibile creare `cdk.CfnResource` direttamente un'istanza e specificare il tipo e le proprietà della risorsa. Questo viene mostrato nell'esempio seguente.

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"
            }
        }
    }
```

```
});
```

Sviluppa portelli di fuga per costrutti L2

Se in un costrutto L2 manca una funzionalità o stai cercando di risolvere un problema, puoi modificare il costrutto L1 che è incapsulato dal costrutto L2.

Tutti i costrutti L2 contengono al loro interno il costrutto L1 corrispondente. Ad esempio, il costrutto di alto livello racchiude il `Bucket` costrutto di basso livello. `CfnBucket` Poiché `CfnBucket` corrisponde direttamente alla AWS CloudFormation risorsa, espone tutte le funzionalità disponibili tramite. AWS CloudFormation

L'approccio di base per accedere al costrutto L1 consiste nell'utilizzare (`construct.node.defaultChild`Python:`default_child`), convertirlo nel tipo corretto (se necessario) e modificarne le proprietà. Ancora una volta, prendiamo l'esempio di a. `Bucket`

TypeScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;

// Change its properties
cfnBucket.analyticsConfiguration = [
  {
    id: 'Config',
    // ...
  }
];
```

JavaScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild;

// Change its properties
cfnBucket.analyticsConfiguration = [
  {
    id: 'Config'
    // ...
  }
```

```
];
```

Python

```
# Get the CloudFormation resource
cfn_bucket = bucket.node.default_child

# Change its properties
cfn_bucket.analytics_configuration = [
    {
        "id": "Config",
        # ...
    }
]
```

Java

```
// Get the CloudFormation resource
CfnBucket cfnBucket = (CfnBucket)bucket.getNode().getDefaultChild();

cfnBucket.setAnalyticsConfigurations(
        Arrays.asList(java.util.Map.of(    // Java 9 or later
            "Id", "Config", // ...
        ));
```

C#

```
// Get the CloudFormation resource
var cfnBucket = (CfnBucket)bucket.Node.DefaultChild;

cfnBucket.AnalyticsConfigurations = new List<object> {
    new Dictionary<string, string>
    {
        ["Id"] = "Config",
        // ...
    }
};
```

È inoltre possibile utilizzare questo oggetto per modificare AWS CloudFormation opzioni come `Metadata` e`UpdatePolicy`.

TypeScript

```
cfnBucket.cfnOptions.metadata = {
  MetadataKey: 'MetadataValue'
};
```

JavaScript

```
cfnBucket.cfnOptions.metadata = {
  MetadataKey: 'MetadataValue'
};
```

Python

```
cfn_bucket.cfn_options.metadata = {
    "MetadataKey": "MetadataValue"
}
```

Java

```
cfnBucket.getCfnOptions().setMetadata(java.util.Map.of(    // Java 9+
    "MetadataKey", "Metadatavalue"));
```

C#

```
cfnBucket.CfnOptions.Metadata = new Dictionary<string, object>
{
    ["MetadataKey"] = "Metadatavalue"
};
```

Portelli Un-escape

Fornisce AWS CDK anche la possibilità di salire a un livello di astrazione, che potremmo chiamare un portello «un-escape». Se si dispone di un costrutto L1, ad esempio, è possibile creare un nuovo costrutto L2 (`Bucket`in questo caso) per avvolgere il costrutto L1. `CfnBucket`

Ciò è utile quando si crea una risorsa L1 ma si desidera utilizzarla con un costrutto che richiede una risorsa L2. È utile anche quando si desidera utilizzare metodi di praticità come `.grantXxxx()` quelli non disponibili nel costrutto L1.

Si passa al livello di astrazione superiore utilizzando un metodo statico sulla classe L2 chiamato, ad esempio, `.fromCfnXxxxx()` per i bucket Amazon `Bucket.fromCfnBucket()` S3. La risorsa L1 è l'unico parametro.

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);
```

I costrutti L2 creati dai costrutti L1 sono oggetti proxy che fanno riferimento alla risorsa L1, simili a quelli creati dai nomi delle risorse, dagli ARN o dalle ricerche. Le modifiche a questi costrutti non influiscono sul AWS CloudFormation modello sintetizzato finale (poiché si dispone della risorsa L1, tuttavia, è possibile modificarla). Per ulteriori informazioni sugli oggetti proxy, vedere. the section called “Riferimento alle risorse presenti nel tuo account AWS”

Per evitare confusione, non create più costrutti L2 che facciano riferimento allo stesso costrutto L1. Ad esempio, se estraete il `CfnBucket` da a `Bucket` usando la tecnica della sezione precedente, non

dovreste creare una seconda `Bucket` istanza chiamando con quella. `Bucket.fromCfnBucket()` `CfnBucket` In realtà funziona come previsto (ne `AWS::S3::Bucket` viene sintetizzata solo una) ma rende il codice più difficile da mantenere.

Raw sovrascrive

Se mancano delle proprietà nel costrutto L1, potete ignorare tutte le digitazioni utilizzando le sostituzioni non elaborate. Ciò consente anche di eliminare le proprietà sintetizzate.

Utilizzate uno dei `addOverride` metodi (Python:`add_override`), come mostrato nell'esempio seguente.

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");
```

Risorse personalizzate

Se la funzionalità non è disponibile tramite AWS CloudFormation, ma solo tramite una chiamata API diretta, devi scrivere una risorsa AWS CloudFormation personalizzata per effettuare la chiamata API necessaria. Puoi utilizzarlo AWS CDK per scrivere risorse personalizzate e inserirle in una normale interfaccia di costruzione. Dal punto di vista di un consumatore del tuo prodotto, l'esperienza sembrerà nativa.

La creazione di una risorsa personalizzata implica la scrittura di una funzione Lambda che risponde agli eventi della risorsa e `DELETE` del `CREATE` ciclo di `UPDATE` vita. Se la tua risorsa personalizzata deve effettuare solo una singola chiamata API, prendi in considerazione l'utilizzo di. [AwsCustomResource](#) In questo modo è possibile eseguire chiamate SDK arbitrarie durante una AWS

CloudFormation distribuzione. Altrimenti, dovresti scrivere la tua funzione Lambda per eseguire il lavoro che devi fare.

L'argomento è troppo ampio per essere trattato completamente in questa sede, ma i seguenti link dovrebbero aiutarti a iniziare:

- Risorse personalizzate
- Esempio di risorse personalizzate
- Per un esempio più completo, vedete la DnsValidatedCertificateclasse nella libreria standard CDK. Questa è implementata come risorsa personalizzata.

Ottenere un valore da una variabile di ambiente

Per ottenere il valore di una variabile di ambiente, utilizzate un codice come il seguente. Questo codice ottiene il valore della variabile di 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";
```

Usa un AWS CloudFormation valore

the section called “Parametri”Per informazioni sull'utilizzo AWS CloudFormation dei parametri con AWS CDK.

Per ottenere un riferimento a una risorsa in un AWS CloudFormation modello esistente, vederethe section called “Importa un AWS CloudFormation modello”.

Importazione di un AWS CloudFormation modello esistente

Importa risorse da un AWS CloudFormation modello nelle tue AWS Cloud Development Kit (AWS CDK) applicazioni utilizzando il `cloudformation-include.CfnInclude`costrutto per convertire le risorse in costrutti L1.

Dopo l'importazione, puoi utilizzare queste risorse nell'app nello stesso modo in cui faresti se fossero state originariamente definite nel codice. AWS CDK È inoltre possibile utilizzare questi costrutti L1 all'interno di costrutti di livello superiore AWS CDK . Ad esempio, questo può consentire di utilizzare i metodi di concessione delle autorizzazioni L2 con le risorse che definiscono.

Il `cloudformation-include.CfnInclude` costrutto aggiunge essenzialmente un wrapper AWS CDK API a qualsiasi risorsa del modello. AWS CloudFormation Utilizzate questa funzionalità per importare i AWS CloudFormation modelli esistenti AWS CDK un pezzo alla volta. In questo modo, puoi gestire le risorse esistenti utilizzando AWS CDK costrutti per sfruttare i vantaggi delle astrazioni di livello superiore. Puoi anche utilizzare questa funzionalità per vendere i tuoi AWS CloudFormation modelli agli AWS CDK sviluppatori fornendo un'API di costruzione. AWS CDK

> **Note**
>
> AWS CDK È inclusa anche la versione v1 `aws-cdk-lib.CfnInclude`, precedentemente utilizzata per lo stesso scopo generale. Tuttavia, manca gran parte della funzionalità di`cloudformation-include.CfnInclude`.

Argomenti

Importazione di un modello AWS CloudFormation

Di seguito è riportato un AWS CloudFormation modello di esempio che utilizzeremo per fornire esempi in questo argomento. Copia e salva il modello come `my-template.json` da seguire. Dopo aver esaminato questi esempi, puoi approfondire ulteriormente utilizzando uno dei AWS CloudFormation modelli distribuiti esistenti. Puoi ottenerli dalla AWS CloudFormation console.

```
{
  "Resources": {
    "MyBucket": {
      "Type": "AWS::S3::Bucket",
```

```
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

Puoi lavorare con modelli JSON o YAML. Consigliamo JSON, se disponibile, poiché i parser YAML possono variare leggermente in base a ciò che accettano.

Di seguito è riportato un esempio di come importare il modello di esempio nell'app utilizzando. AWS CDK `cloudformation-include` I modelli vengono importati nel contesto di uno stack 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"
            });
        }
    }
}
```

Per impostazione predefinita, l'importazione di una risorsa preserva l'ID logico originale della risorsa dal modello. Questo comportamento è adatto per importare un AWS CloudFormation modello in AWS CDK, dove devono essere conservati gli ID logici. AWS CloudFormation necessita di queste informazioni per riconoscere queste risorse importate come le stesse risorse del AWS CloudFormation modello.

Se state sviluppando un AWS CDK costrutto wrapper per il modello in modo che possa essere utilizzato da altri AWS CDK sviluppatori, chiedete invece di AWS CDK generare nuovi ID di risorse. In questo modo, il costrutto può essere utilizzato più volte in uno stack senza conflitti di nomi. A tale scopo, impostate la `preserveLogicalIds` proprietà su `false` quando importate il modello. Di seguito è riportato un esempio:

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
});
```

Per mettere le risorse importate sotto il controllo della tua AWS CDK app, aggiungi lo stack a: App

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");
        }
    }
}
```

Per verificare che non vi siano modifiche involontarie alle AWS risorse nello stack, puoi eseguire una differenza. Usa il AWS CDK CLI `cdk diff` comando e ometti qualsiasi AWS CDK metadato specifico. Di seguito è riportato un esempio:

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

Dopo aver importato un AWS CloudFormation modello, l' AWS CDK app dovrebbe diventare la fonte di riferimento per le risorse importate. Per apportare modifiche alle tue risorse, modificale nell' AWS CDK app e distribuiscile con il AWS CDK CLI cdk deploy comando.

Accesso alle risorse importate

Il nome `template` nel codice di esempio rappresenta il AWS CloudFormation modello importato. Per accedere a una risorsa da esso, utilizzate il `getResource()`metodo dell'oggetto. Per accedere alla risorsa restituita come tipo specifico di risorsa, trasmetti il risultato al tipo desiderato. Questo non è necessario in Python o. JavaScript Di seguito è riportato un esempio:

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");
```

Da questo esempio, ora `cfnBucket` è un'istanza della `aws-s3.CfnBucket`classe. Si tratta di un costrutto L1 che rappresenta la risorsa corrispondente AWS CloudFormation . Puoi trattarlo come qualsiasi altra risorsa del suo tipo. Ad esempio, è possibile ottenere il relativo valore ARN con la `bucket.attrArn` proprietà.

Per racchiudere invece la `CfnBucket` risorsa L1 in un'`aws-s3.Bucket`istanza L2, utilizzate i metodi `fromBucketArn()`statici o. `fromBucketAttributes()fromBucketName()` In genere, il `fromBucketName()` metodo è il più conveniente. Di seguito è riportato un esempio:

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);
```

Altri costrutti L2 hanno metodi simili per creare il costrutto da una risorsa esistente.

Quando racchiudete un costrutto L1 in un costrutto L2, non viene creata una nuova risorsa. Dal nostro esempio, non stiamo creando un secondo bucket S3;. Invece, la nuova `Bucket` istanza incapsula l'esistente. `CfnBucket`

Dall'esempio, ora `bucket` è un costrutto L2 che si comporta come `Bucket` qualsiasi altro costrutto L2. Ad esempio, è possibile concedere a una AWS Lambda funzione l'accesso in scrittura al bucket utilizzando il comodo metodo del bucket. `grantWrite()` Non è necessario definire manualmente la policy AWS Identity and Access Management (IAM) necessaria. Di seguito è riportato un esempio:

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

Sostituzione dei parametri

Se il AWS CloudFormation modello contiene parametri, è possibile sostituirli con valori di fase di compilazione al momento dell'importazione utilizzando la `parameters` proprietà. Nell'esempio seguente, sostituiamo il `UploadBucket` parametro con l'ARN di un bucket definito altrove nel nostro codice. AWS CDK

TypeScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

JavaScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
```

```
});
```

Python

```
template = cfn_inc.CfnInclude(self, "Template",
    template_file="my-template.json",
    parameters=dict(UploadBucket=bucket.bucket_arn)
)
```

Java

```
CfnInclude template = CfnInclude.Builder.create(this, "Template")
 .templateFile("my-template.json")
 .parameters(java.util.Map.of(    // Map.of requires Java 9+
   "UploadBucket", bucket.getBucketArn()))
 .build();
```

C#

```
var template = new cfnInc.CfnInclude(this, "Template", new cfnInc.CfnIncludeProps
{
    TemplateFile = "my-template.json",
    Parameters = new Dictionary<string, string>
    {
        { "UploadBucket", bucket.BucketArn }
    }
});
```

Altri elementi del modello

Puoi importare qualsiasi elemento AWS CloudFormation del modello, non solo risorse. Gli elementi importati diventano parte dello AWS CDK stack. Per importare questi elementi, utilizzate i seguenti metodi dell'`CfnInclude`oggetto:

- `getCondition()`— AWS CloudFormation condizioni.
- `getHook()`— AWS CloudFormation ganci per installazioni blu/verdi.
- `getMapping()`AWS CloudFormation— mappature.
- `getOutput()`— AWS CloudFormation uscite.
- `getParameter()`— AWS CloudFormation parametri.

- `getRule()`— AWS CloudFormation regole per i AWS Service Catalog modelli.

Ciascuno di questi metodi restituisce un'istanza di una classe che rappresenta il tipo specifico di AWS CloudFormation elemento. Questi oggetti sono mutabili. Le modifiche apportate ad essi verranno visualizzate nel modello generato dallo AWS CDK stack. Di seguito è riportato un esempio che importa un parametro dal modello e ne modifica il valore predefinito:

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";
```

Stack nidificati

È possibile importare pile annidate specificandole al momento dell'importazione del modello principale o in un momento successivo. Il modello nidificato deve essere archiviato in un file locale, ma deve essere utilizzato come `NestedStack` risorsa nel modello principale. Inoltre, il nome della

risorsa utilizzato nel AWS CDK codice deve corrispondere al nome utilizzato per lo stack nidificato nel modello principale.

Data questa definizione di risorsa nel modello principale, il codice seguente mostra come importare lo stack nidificato di riferimento in entrambi i modi.

```
"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'
});
```

È possibile importare più stack annidati con entrambi i metodi. Quando si importa il modello principale, si fornisce una mappatura tra il nome della risorsa di ogni stack nidificato e il relativo file modello. Questa mappatura può contenere un numero qualsiasi di voci. Per farlo dopo l'importazione iniziale, chiama `loadNestedStack()` una volta per ogni stack annidato.

Dopo aver importato uno stack nidificato, puoi accedervi utilizzando il metodo del modello principale. `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;
```

Il `getNestedStack()` metodo restituisce un'istanza. `IncludedNestedStack` Da questa istanza, è possibile accedere all' AWS CDK `NestedStack`istanza tramite la `stack` proprietà, come mostrato

nell'esempio. È inoltre possibile accedere all'oggetto AWS CloudFormation modello originale tramite`includedTemplate`, da cui è possibile caricare risorse e altri AWS CloudFormation elementi.

Ottieni un valore dal Systems Manager Parameter Store

AWS Cloud Development Kit (AWS CDK) Possono recuperare il valore degli attributi di AWS Systems Manager Parameter Store. Durante la sintesi, AWS CDK produce un token che viene risolto da AWS CloudFormation durante la distribuzione.

AWS CDK Supporta il recupero di valori semplici e sicuri. Puoi richiedere una versione specifica di entrambi i tipi di valore. Per valori semplici, puoi omettere la versione dalla tua richiesta per recuperare la versione più recente. Per i valori sicuri, è necessario specificare la versione quando si richiede il valore dell'attributo secure.

> **Note**
>
> Questo argomento mostra come leggere gli attributi dal AWS Systems Manager Parameter Store. Puoi anche leggere i segreti di AWS Secrets Manager (vediOttieni un valore da AWS Secrets Manager).

Argomenti

- Leggi i valori di Systems Manager al momento dell'implementazione
- Leggi i valori di Systems Manager al momento della sintesi
- Scrivi valori in Systems Manager

Leggi i valori di Systems Manager al momento dell'implementazione

Per leggere i valori dall'archivio dei parametri di Systems Manager, utilizzare il valueForStringparametro e valueForSecureStringParameteri metodi. Scegliete un metodo in base al fatto che l'attributo desiderato sia una stringa semplice o un valore di stringa sicuro. Questi metodi restituiscono i token, non il valore effettivo. Il valore viene risolto da AWS CloudFormation durante la distribuzione. Di seguito è riportato un esempio:

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
```

Attualmente questa funzionalità è supportata da un numero limitato di AWS servizi.

Leggi i valori di Systems Manager al momento della sintesi

A volte è utile fornire un parametro al momento della sintesi. In questo modo, il AWS CloudFormation modello utilizzerà sempre lo stesso valore invece di risolvere il valore durante la distribuzione.

Per leggere un valore dal Systems Manager Parameter Store al momento della sintesi, utilizzate il valueFromLookupmetodo (Python:`value_from_lookup`). Questo metodo restituisce il valore effettivo del parametro come the section called “Context” valore. Se il valore non è già stato memorizzato nella cache `cdk.json` o passato nella riga di comando, viene recuperato dall'account corrente AWS . Per questo motivo, lo stack deve essere sintetizzato con informazioni ambientali esplicite. AWS

Di seguito è riportato un esempio:

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");
```

È possibile recuperare solo stringhe di Systems Manager semplici. Le stringhe sicure non possono essere recuperate. Verrà sempre restituita la versione più recente. Non è possibile richiedere versioni specifiche.

> **Important**
>
> Il valore recuperato finirà nel modello sintetizzato AWS CloudFormation . Questo potrebbe rappresentare un rischio per la sicurezza, a seconda di chi ha accesso ai AWS CloudFormation modelli e del tipo di valore. In genere, non utilizzate questa funzionalità per password, chiavi o altri valori che desiderate mantenere privati.

Scrivi valori in Systems Manager

È possibile utilizzare la AWS CLI AWS Management Console, o un AWS SDK per impostare i valori dei parametri di Systems Manager. Negli esempi seguenti viene utilizzato il 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"
```

Quando aggiorni un valore SSM già esistente, includi anche l'opzione. `--overwrite`

```
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"
```

Ottieni un valore da AWS Secrets Manager

Per utilizzare i valori AWS Secrets Manager della tua AWS CDK app, usa il metodo fromSecretAttributes(). Rappresenta un valore recuperato da Secrets Manager e utilizzato al momento della AWS CloudFormation distribuzione. Di seguito è riportato un esempio:

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**
>
> Usa il comando AWS CLI CLI create-secret per creare un segreto dalla riga di comando, ad esempio durante il test:
>
> ```
> aws secretsmanager create-secret --name ImportedSecret --secret-string
>  mygroovybucket
> ```
>
> Il comando restituisce un ARN che è possibile utilizzare con l'esempio precedente.

Dopo aver creato un'`Secret`istanza, potete ottenere il valore del segreto dall'attributo dell'`secretValue`istanza. Il valore è rappresentato da un'`SecretValue`istanza, un tipo speciale dithe section called “Gettoni”. Poiché è un token, ha significato solo dopo la risoluzione. L'app CDK non ha bisogno di accedere al suo valore effettivo. L'app può invece passare l'`SecretValue`istanza (o la sua rappresentazione di stringa o numerica) a qualsiasi metodo CDK che richieda il valore.

Imposta una CloudWatch sveglia

Usa il pacchetto aws-cloudwatch per configurare gli allarmi Amazon CloudWatch sui parametri. CloudWatch Puoi utilizzare metriche predefinite o crearne di tue.

Argomenti

- Utilizzando una metrica esistente
- Creazione di una metrica personalizzata
- Creare l'allarme

Utilizzando una metrica esistente

Molti moduli di AWS Construct Library consentono di impostare un allarme su una metrica esistente passando il nome della metrica a un metodo pratico su un'istanza di un oggetto che dispone di metriche. Ad esempio, data una coda Amazon SQS, puoi ottenere la metrica ApproximateNumberOfMessagesVisibledal metodo metric () della coda:

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");
```

Creazione di una metrica personalizzata

Crea la tua metrica come segue, dove il valore dello spazio dei nomi dovrebbe essere qualcosa come AWS/SQS per una coda Amazon SQS. Devi anche specificare il nome e la dimensione della metrica:

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" }
    }
});
```

Creare l'allarme

Una volta che hai una metrica, esistente o definita da te, puoi creare un allarme. In questo esempio, l'allarme viene generato quando sono presenti più di 100 metriche in due degli ultimi tre periodi di valutazione. È possibile utilizzare confronti come less-than nei propri allarmi tramite la proprietà. `comparisonOperator` G reater-than-or-equal -to è l' AWS CDK impostazione predefinita, quindi non è necessario specificarlo.

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
});
```

Un modo alternativo per creare un allarme consiste nell'utilizzare il metodo createAlarm () della metrica, che assume essenzialmente le stesse proprietà del costruttore. `Alarm` Non è necessario inserire la metrica, perché è già nota.

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
});
```

Salva e recupera i valori delle variabili di contesto

È possibile specificare le variabili di contesto con AWS Cloud Development Kit (AWS CDK) CLI o nel `cdk.json` file. Quindi, utilizzate il `TryGetContext` metodo per recuperare i valori.

Argomenti

- Specificate le variabili di contesto
- Recupera i valori delle variabili di contesto

Specificate le variabili di contesto

È possibile specificare una variabile di contesto come parte di un AWS CDK CLI comando o in`cdk.json`.

Per creare una variabile di contesto della riga di comando, utilizzate l'opzione --context (-c), come illustrato nell'esempio seguente.

```
cdk synth -c bucket_name=mygroovybucket
```

Per specificare la stessa variabile di contesto e lo stesso valore nel `cdk.json` file, utilizzate il codice seguente.

```
{
  "context": {
    "bucket_name": "myotherbucket"
  }
}
```

Se specificate una variabile di contesto utilizzando AWS CDK CLI sia il `cdk.json` file che, il AWS CDK CLI valore ha la precedenza.

Recupera i valori delle variabili di contesto

Per ottenere il valore di una variabile di contesto nella tua app, usa il `TryGetContext` metodo nel contesto di un costrutto. (Cioè, quando`this`, o `self` in Python, è un'istanza di qualche costrutto.)

In questo esempio, recuperiamo il valore della variabile di contesto. `bucket_name` Se il valore richiesto non è definito, `TryGetContext` restituisce `undefined` (`None`in Python; `null` in Java e C#; `nil` in Go) anziché sollevare un'eccezione.

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");
```

Al di fuori del contesto di un costrutto, puoi accedere alla variabile di contesto dall'oggetto app, in questo modo.

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");
```

Per ulteriori dettagli sull'utilizzo delle variabili di contesto, the section called “Context” consultate.

Utilizzo delle risorse del registro AWS CloudFormation pubblico

Il registro AWS CloudFormation pubblico consente di gestire le estensioni, sia pubbliche che private, come risorse, moduli e hook disponibili per l'uso in. Account AWSÈ possibile utilizzare le estensioni delle risorse pubbliche nelle AWS Cloud Development Kit (AWS CDK) applicazioni con il `CfnResource`costrutto.

Per ulteriori informazioni sul registro AWS CloudFormation pubblico, consulta Uso del AWS CloudFormation registro nella Guida per l'AWS CloudFormation utente.

Tutte le estensioni pubbliche pubblicate da AWS sono disponibili per tutti gli account in tutte le regioni senza alcuna azione da parte dell'utente. Tuttavia, devi attivare ogni estensione di terze parti che desideri utilizzare, in ogni account e regione in cui desideri utilizzarla.

> **Note**
>
> Se utilizzi tipi AWS CloudFormation di risorse di terze parti, dovrai sostenere dei costi. I costi si basano sul numero di operazioni del gestore eseguite ogni mese e sulla durata delle operazioni del gestore. Consulta CloudFormation i prezzi per i dettagli completi.

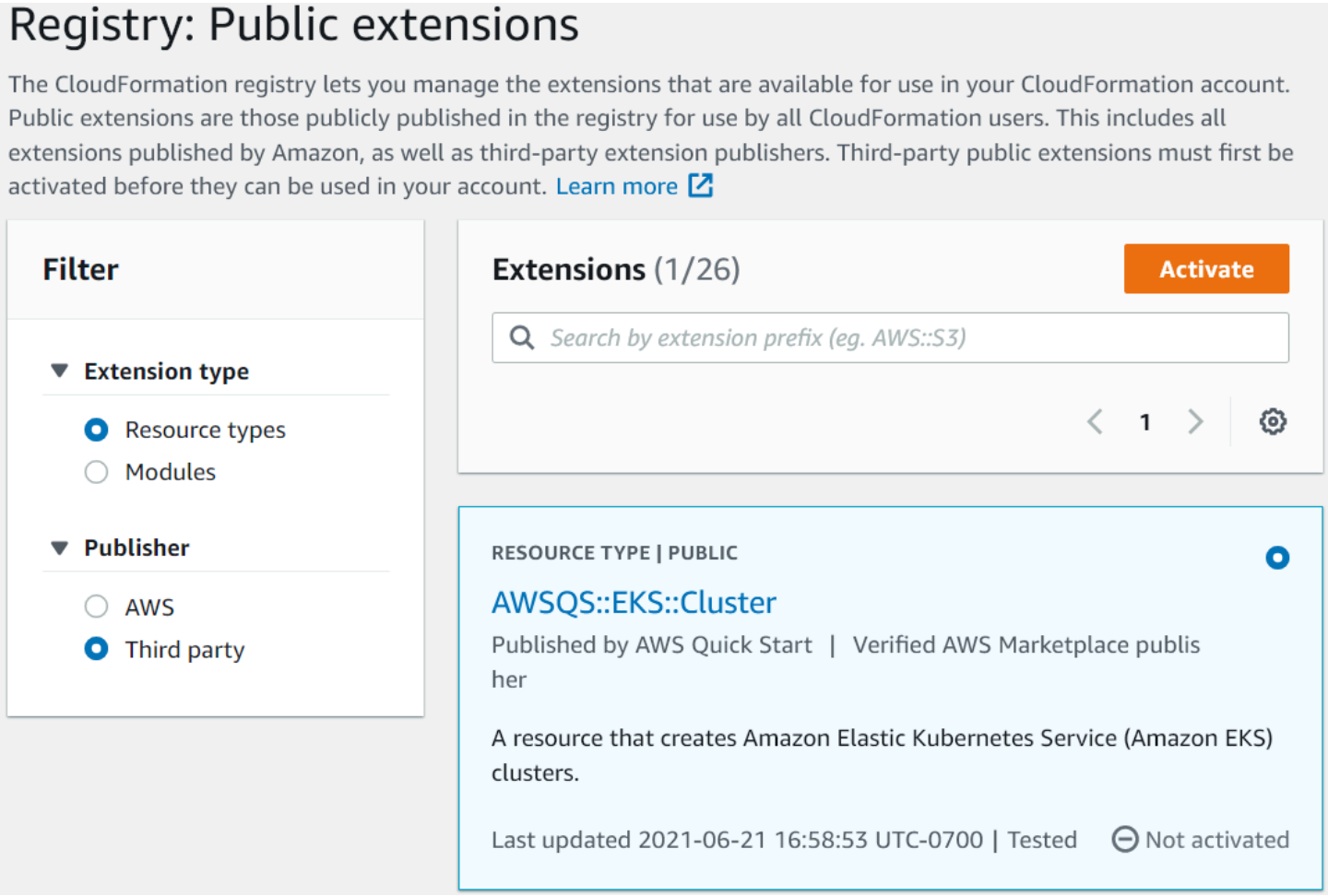
Per ulteriori informazioni sulle estensioni pubbliche, consulta la sezione Uso delle estensioni pubbliche CloudFormation nella Guida AWS CloudFormation per l'utente

Argomenti

Attivazione di una risorsa di terze parti nel tuo account e nella tua regione

Le estensioni pubblicate da AWS non richiedono l'attivazione. Sono sempre disponibili in ogni account e regione. Puoi attivare un'estensione di terze parti tramite AWS Management Console, tramite o distribuendo una AWS CloudFormation risorsa speciale. AWS Command Line Interface

Per attivare un'estensione di terze parti tramite AWS Management Console o scopri quali risorse sono disponibili



1. Accedi all' AWS account in cui desideri utilizzare l'estensione, quindi passa alla regione in cui desideri utilizzarla.
2. Accedi alla CloudFormation console tramite il menu Servizi.
3. Scegli Estensioni pubbliche nella barra di navigazione, quindi attiva il pulsante di opzione Terze parti in Publisher. Viene visualizzato un elenco delle estensioni pubbliche di terze parti disponibili. (Puoi anche AWSscegliere di visualizzare un elenco delle estensioni pubbliche pubblicate da AWS, anche se non è necessario attivarle).
4. Sfoglia l'elenco e trova l'estensione che desideri attivare. In alternativa, cercala, quindi attiva il pulsante di opzione nell'angolo in alto a destra della scheda dell'estensione.
5. Scegli il pulsante Attiva nella parte superiore dell'elenco per attivare l'estensione selezionata. Viene visualizzata la pagina Attiva dell'estensione.

6. Nella pagina Activate, è possibile sovrascrivere il nome predefinito dell'estensione e specificare un ruolo di esecuzione e una configurazione di registrazione. Puoi anche scegliere se aggiornare automaticamente l'estensione quando viene rilasciata una nuova versione. Dopo aver impostato queste opzioni come preferisci, scegli Attiva estensione nella parte inferiore della pagina.

Per attivare un'estensione di terze parti utilizzando il AWS CLI

- Utilizza il comando `activate-type`. Sostituisci l'ARN del tipo personalizzato che desideri utilizzare dove indicato.

  Di seguito è riportato un esempio:

  ```
  aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
  ```

Per attivare un'estensione di terze parti tramite CloudFormation il nostro CDK

- Implementate una risorsa di tipo `AWS::CloudFormation::TypeActivation` e specificate le seguenti proprietà:

  a. `TypeName`- Il nome del tipo, ad esempio`AWSQS::EKS::Cluster`.

  b. `MajorVersion`- Il numero di versione principale dell'estensione desiderata. Omettilo se desideri la versione più recente.

  c. `AutoUpdate`- Se aggiornare automaticamente questa estensione quando l'editore rilascia una nuova versione secondaria. (Gli aggiornamenti delle versioni principali richiedono una modifica esplicita della `MajorVersion` proprietà.)

  d. `ExecutionRoleArn`- L'ARN del ruolo IAM in base al quale verrà eseguita questa estensione.

  e. `LoggingConfig`- La configurazione di registrazione per l'estensione.

  La `TypeActivation` risorsa può essere distribuita dal CDK utilizzando il costrutto. `CfnResource` Questo è illustrato per le estensioni effettive nella sezione seguente.

Aggiungere una risorsa dal registro AWS CloudFormation pubblico all'app CDK

Utilizzate il `CfnResource`costrutto per includere una risorsa del registro AWS CloudFormation pubblico nella vostra applicazione. Questo costrutto si trova nel modulo del CDK. `aws-cdk-lib`

Ad esempio, supponiamo che esista una risorsa pubblica denominata `MY::S5::UltimateBucket` che desideri utilizzare nell'applicazione. AWS CDK Questa risorsa ha una proprietà: il nome del bucket. L'`CfnResource`istanziazione corrispondente ha questo aspetto.

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"
    }
});
```

Implementazione di applicazioni AWS CDK

Implementa AWS Cloud Development Kit (AWS CDK) applicazioni.

Argomenti

AWS CDK convalida delle politiche al momento della sintesi

Argomenti

Convalida delle politiche al momento della sintesi

Se tu o la tua organizzazione utilizzate uno strumento di convalida delle politiche, come AWS CloudFormation Guardl'OPA, per definire i vincoli sul AWS CloudFormation modello, potete integrarli con at synthesis. AWS CDK Utilizzando il plug-in di convalida delle politiche appropriato, potete fare in modo che l' AWS CDK applicazione verifichi il AWS CloudFormation modello generato rispetto alle vostre politiche subito dopo la sintesi. In caso di violazioni, la sintesi avrà esito negativo e un rapporto verrà stampato sulla console.

La convalida eseguita AWS CDK al momento della sintesi convalida i controlli a un certo punto del ciclo di vita dell'implementazione, ma non può influire sulle azioni che avvengono al di fuori della sintesi. Gli esempi includono le azioni intraprese direttamente nella console o tramite le API di servizio. Non resistono all'alterazione dei AWS CloudFormation modelli dopo la sintesi. Qualche altro meccanismo per convalidare lo stesso set di regole in modo più autorevole dovrebbe essere impostato indipendentemente, come hook o.AWS CloudFormationAWS Config Tuttavia, la capacità di valutare il set AWS CDK di regole durante lo sviluppo è ancora utile in quanto migliorerà la velocità di rilevamento e la produttività degli sviluppatori.

L'obiettivo della convalida delle AWS CDK policy è ridurre al minimo la quantità di configurazione necessaria durante lo sviluppo e renderlo il più semplice possibile.

> **Note**
>
> Questa funzionalità è considerata sperimentale e sia l'API del plug-in che il formato del rapporto di convalida sono soggetti a modifiche in futuro.

Argomenti

Per gli sviluppatori di applicazioni

Per utilizzare uno o più plugin di convalida nell'applicazione, utilizzate la `policyValidationBeta1` proprietà di: `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: [...],
});
```

Immediatamente dopo la sintesi, tutti i plugin registrati in questo modo verranno richiamati per convalidare tutti i modelli generati nell'ambito definito. In particolare, se si registrano i modelli nell'Appoggetto, tutti i modelli saranno soggetti a convalida.

> **Warning**
>
> Oltre a modificare l'assembly cloud, i plugin possono fare tutto ciò che può fare l' AWS CDK applicazione. Possono leggere i dati dal filesystem, accedere alla rete ecc. È tua responsabilità, in qualità di consumatore di un plug-in, verificare che sia sicuro da usare.

AWS CloudFormation Guard plugin

L'utilizzo del CfnGuardValidatorplug-in consente di AWS CloudFormation Guardeseguire convalide delle politiche. Il `CfnGuardValidator` plugin è dotato di un set selezionato di controlli AWS Control Tower proattivi integrati. L'attuale set di regole è disponibile nella documentazione del progetto. Come accennato inConvalida delle politiche al momento della sintesi, raccomandiamo alle organizzazioni di impostare un metodo di convalida più autorevole utilizzando gli hook.AWS CloudFormation

Per AWS Control Towerі clienti, questi stessi controlli proattivi possono essere implementati in tutta l'organizzazione. Quando abiliti i controlli AWS Control Tower proattivi nel tuo AWS Control Tower ambiente, i controlli possono impedire l'implementazione di risorse non conformi distribuite tramite. AWS CloudFormationPer ulteriori informazioni sui controlli proattivi gestiti e su come funzionano, consulta la documentazione.AWS Control Tower

Questi controlli AWS CDK raggruppati e i controlli AWS Control Tower proattivi gestiti vengono utilizzati al meglio insieme. In questo scenario puoi configurare questo plug-in di convalida con gli stessi controlli proattivi attivi nel tuo ambiente cloud. AWS Control Tower Potrai quindi acquisire rapidamente la certezza che la tua AWS CDK applicazione supererà i AWS Control Tower controlli `cdk synth` eseguendo localmente.

Rapporto di convalida

Quando sintetizzi l' AWS CDK app, verranno richiamati i plugin di convalida e i risultati verranno stampati. Di seguito è riportato un esempio di rapporto.

```
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
```

Per impostazione predefinita, il rapporto verrà stampato in un formato leggibile dall'uomo. Se desideri un report in formato JSON, abilitalo utilizzando la `@aws-cdk/core:validationReportJson` CLI o passandolo direttamente all'applicazione:

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

In alternativa, puoi impostare questa coppia chiave-valore di contesto usando `cdk.context.json` i file `cdk.json` o nella directory del tuo progetto (vedi). Contesto di runtime

Se scegli il formato JSON, AWS CDK stamperà il rapporto di convalida delle politiche in un file chiamato `policy-validation-report.json` nella directory di cloud assembly. Per il formato predefinito, leggibile dall'uomo, il report verrà stampato sullo standard output.

Per gli autori di plugin

Plug-in

Il framework AWS CDK principale è responsabile della registrazione e dell'invocazione dei plugin e della successiva visualizzazione del rapporto di convalida formattato. La responsabilità del plugin è quella di fungere da livello di traduzione tra il AWS CDK framework e lo strumento di convalida delle politiche. Un plugin può essere creato in qualsiasi lingua supportata da AWS CDK. Se stai creando un plug-in che potrebbe essere utilizzato da più lingue, ti consigliamo di crearlo in `TypeScript` modo da poter utilizzare JSII per pubblicare il plug-in in ogni AWS CDK lingua.

Creazione di plugin

Il protocollo di comunicazione tra il modulo AWS CDK principale e lo strumento di policy è definito dall'`IPolicyValidationPluginBeta1`interfaccia. Per creare un nuovo plugin è necessario scrivere una classe che implementi questa interfaccia. Ci sono due cose che devi implementare: il nome del plugin (sovrascrivendo la `name` proprietà) e il `validate()` metodo.

Il framework chiamerà`validate()`, passando un `IValidationContextBeta1` oggetto. La posizione dei modelli da convalidare è data da`templatePaths`. Il plugin dovrebbe restituire un'istanza di. `ValidationPluginReportBeta1` Questo oggetto rappresenta il rapporto che l'utente riceverà alla fine della sintesi.

```
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',
      }],
    }],
  };
```

```
}
```

Nota che i plugin non sono autorizzati a modificare nulla nell'assembly cloud. Qualsiasi tentativo in tal senso comporterà un errore di sintesi.

Se il tuo plugin dipende da uno strumento esterno, tieni presente che alcuni sviluppatori potrebbero non avere ancora installato questo strumento nelle loro workstation. Per ridurre al minimo l'attrito, ti consigliamo vivamente di fornire uno script di installazione insieme al pacchetto del plugin, per automatizzare l'intero processo. Meglio ancora, esegui lo script come parte dell'installazione del pacchetto. Con`npm`, ad esempio, puoi aggiungerlo allo `postinstall` script nel `package.json` file.

Gestione delle esenzioni

Se l'organizzazione dispone di un meccanismo per la gestione delle esenzioni, può essere implementato come parte del plugin di convalida.

Uno scenario di esempio per illustrare un possibile meccanismo di esenzione:

- Un'organizzazione ha una regola secondo cui i bucket Amazon S3 pubblici non sono consentiti, tranne in determinati scenari.
- Uno sviluppatore sta creando un bucket Amazon S3 che rientra in uno di questi scenari e richiede un'esenzione (ad esempio, crea un ticket).
- Gli strumenti di sicurezza sanno leggere dal sistema interno che registra le esenzioni

In questo scenario lo sviluppatore richiederebbe un'eccezione nel sistema interno e quindi avrà bisogno di un modo per «registrare» tale eccezione. Oltre all'esempio del plugin guard, potresti creare un plug-in che gestisca le esenzioni filtrando le violazioni che hanno un'esenzione corrispondente in un sistema di ticketing interno.

Vedi i plugin esistenti per esempio per le implementazioni.

- @cdklabs/cdk-validator-cfnguard

Integrazione e distribuzione continue (CI/CD) con CDK Pipelines

Utilizzate il modulo CDK Pipelines AWS della Construct Library per configurare la distribuzione continua delle applicazioni. AWS CDK Quando esegui il commit del codice sorgente della tua app

CDK in AWS CodeCommit, o GitHub AWS CodeStar, CDK Pipelines può creare, testare e distribuire automaticamente la tua nuova versione.

Le CDK Pipelines si aggiornano automaticamente. Se aggiungi stadi o stack di applicazioni, la pipeline si riconfigura automaticamente per distribuire quei nuovi stadi o stack.

> **Note**
>
> CDK Pipelines supporta due API. Una è l'API originale resa disponibile nella CDK Pipelines Developer Preview. L'altra è un'API moderna che incorpora il feedback dei clienti CDK ricevuto durante la fase di anteprima. Gli esempi in questo argomento utilizzano l'API moderna. Per i dettagli sulle differenze tra le due API supportate, consulta l'API originale di CDK Pipelines nel repository aws-cdk. GitHub

Argomenti

AWS Avvia i tuoi ambienti

Prima di poter utilizzare CDK Pipelines, è necessario avviare AWS l'ambiente in cui verranno distribuiti gli stack.

Una CDK Pipeline coinvolge almeno due ambienti. Il primo ambiente è quello in cui viene effettuato il provisioning della pipeline. Il secondo ambiente è quello in cui si desidera distribuire gli stack o le fasi dell'applicazione (le fasi sono gruppi di stack correlati). Questi ambienti possono essere uguali, ma una buona pratica è quella di isolare le fasi l'una dall'altra in ambienti diversi.

> **Note**
>
> the section called “Bootstrapping (Processo di bootstrap)”Per ulteriori informazioni sui tipi di risorse create dal bootstrap e su come personalizzare lo stack di bootstrap, vedi.

L'implementazione continua con CDK Pipelines richiede l'inclusione di quanto segue nello stack CDK Toolkit:

- Un bucket Amazon Simple Storage Service (Amazon S3).
- Un repository Amazon ECR.
- Ruoli IAM per fornire alle varie parti di una pipeline le autorizzazioni di cui hanno bisogno.

Il CDK Toolkit aggiornerà lo stack di bootstrap esistente o ne creerà uno nuovo, se necessario.

Per avviare un ambiente in grado di effettuare il provisioning di una AWS CDK pipeline, `cdk bootstrap` richiamate come mostrato nell'esempio seguente. Se necessario, richiamando il AWS CDK Toolkit tramite il `npx` comando, lo si installa temporaneamente. Utilizzerà anche la versione del Toolkit installata nel progetto corrente, se ne esiste una.

--cloudformation-execution-policiesspecifica l'ARN di una policy in base alla quale verranno eseguite le future implementazioni di CDK Pipelines. La `AdministratorAccess` politica predefinita assicura che la pipeline possa distribuire ogni tipo di risorsa. AWS Se utilizzi questa politica, assicurati di considerare attendibile tutto il codice e le dipendenze che compongono la tua app. AWS CDK

La maggior parte delle organizzazioni impone controlli più rigorosi sui tipi di risorse che possono essere distribuite mediante l'automazione. Rivolgiti al reparto competente all'interno dell'organizzazione per determinare la politica da utilizzare nella tua pipeline.

È possibile omettere l'--profileopzione se il AWS profilo predefinito contiene la configurazione di autenticazione necessaria e. Regione 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
```

Per avviare ambienti aggiuntivi in cui AWS CDK le applicazioni verranno distribuite dalla pipeline, utilizzate invece i seguenti comandi. L'--trustopzione indica quale altro account deve disporre delle autorizzazioni per distribuire applicazioni in questo ambiente. AWS CDK Per questa opzione, specifica l'ID dell'account della pipeline. AWS

Ancora una volta, puoi omettere l'--profileopzione se il tuo AWS profilo predefinito contiene la configurazione di autenticazione necessaria e. Regione 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**
>
> Utilizzate le credenziali amministrative solo per il bootstrap e per il provisioning della pipeline iniziale. In seguito, utilizzate la pipeline stessa, non il computer locale, per distribuire le modifiche.

Se stai aggiornando un ambiente con bootstrap legacy, il bucket Amazon S3 precedente rimane orfano quando viene creato il nuovo bucket. Eliminalo manualmente utilizzando la console Amazon S3.

Inizializza un progetto

Crea un nuovo GitHub progetto vuoto e clonalo sulla tua workstation nella directory. `my-pipeline` (I nostri esempi di codice in questo argomento utilizzano. GitHub Puoi anche usare AWS CodeStar o AWS CodeCommit.)

```
git clone GITHUB-CLONE-URL my-pipeline
cd my-pipeline
```

> **Note**
>
> Puoi usare un nome diverso `my-pipeline` da quello della directory principale dell'app. Tuttavia, in tal caso, sarà necessario modificare i nomi dei file e delle classi più avanti in questo argomento. Questo perché il AWS CDK Toolkit basa alcuni nomi di file e classi sul nome della directory principale.

Dopo la clonazione, inizializza il progetto come al solito.

TypeScript

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

JavaScript

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

Python

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

Dopo aver creato l'app, inserisci anche i due comandi seguenti. Questi attivano l'ambiente virtuale Python dell'app e installano le dipendenze AWS CDK principali.

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

Java

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

Se utilizzi un IDE, ora puoi aprire o importare il progetto. In Eclipse, ad esempio, scegli File > Importa > Maven > Progetti Maven esistenti. Assicuratevi che le impostazioni del progetto siano impostate per utilizzare Java 8 (1.8).

C#

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

Se utilizzi Visual Studio, apri il file della soluzione nella `src` directory.

Go

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

Dopo aver creato l'app, inserisci anche il seguente comando per installare i moduli AWS Construct Library richiesti dall'app.

```
go get
```

> **Important**
>
> Assicurati di affidare i tuoi `cdk.context.json` file `cdk.json` e i tuoi file al controllo del codice sorgente. Le informazioni di contesto (come i flag delle funzionalità e i valori memorizzati nella cache recuperati dal tuo AWS account) fanno parte dello stato del progetto. I valori possono essere diversi in un altro ambiente, il che può causare cambiamenti imprevisti nei risultati. Per ulteriori informazioni, consulta the section called “Context”.

Definire una pipeline

L'applicazione CDK Pipelines includerà almeno due stack: uno che rappresenta la pipeline stessa e uno o più stack che rappresentano l'applicazione distribuita attraverso di essa. Gli stack possono anche essere raggruppati in fasi, che è possibile utilizzare per distribuire copie degli stack di

infrastruttura in ambienti diversi. Per ora, prenderemo in considerazione la pipeline e in seguito approfondiremo l'applicazione che verrà implementata.

Il costrutto `CodePipeline`è il costrutto che rappresenta una CDK Pipeline che utilizza come motore di distribuzione. AWS CodePipeline Quando si crea un'istanza `CodePipeline` in uno stack, si definisce la posizione di origine della pipeline (ad esempio un repository). GitHub Definisci anche i comandi per creare l'app.

Ad esempio, quanto segue definisce una pipeline la cui origine è archiviata in un GitHub repository. Include anche una fase di compilazione per un'applicazione TypeScript CDK. Inserisci le informazioni sul tuo GitHub repo dove indicato.

> **Note**
>
> Per impostazione predefinita, la pipeline si autentica GitHub utilizzando un token di accesso personale memorizzato in Secrets Manager sotto il nome. `github-token`

Dovrai anche aggiornare l'istanza dello stack di pipeline per specificare l'account e la regione. AWS

TypeScript

In `lib/my-pipeline-stack.ts` (può variare se la cartella del progetto non ha un 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']
      })
    });
  }
```

```
}
```

In `bin/my-pipeline.ts` (può variare se la cartella del progetto non ha un 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

In `lib/my-pipeline-stack.js` (può variare se la cartella del progetto non ha un 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 }
```

In `bin/my-pipeline.js` (può variare se la cartella del progetto non ha un 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

In `my-pipeline/my-pipeline-stack.py` (può variare se la cartella del progetto non ha un 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"]
                        )
```

```
                    )
```

In `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

In `src/main/java/com/myorg/MyPipelineStack.java` (può variare se la cartella del progetto non ha un 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();
    }
}
```

In `src/main/java/com/myorg/MyPipelineApp.java` (può variare se la cartella del progetto non ha un 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#

In `src/MyPipeline/MyPipelineStack.cs` (può variare se la cartella del progetto non ha un 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" }
                })
            });
        }
    }
}
```

In `src/MyPipeline/Program.cs` (può variare se la cartella del progetto non ha un 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();
        }
    }
}
```

È necessario distribuire una pipeline manualmente una volta. Dopodiché, la pipeline si aggiorna dal repository del codice sorgente. Quindi assicurati che il codice nel repository sia il codice che desideri distribuire. Controlla le modifiche e invia a GitHub, quindi distribuisci:

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

> **Tip**
>
> Ora che hai completato la distribuzione iniziale, il tuo AWS account locale non necessita più dell'accesso amministrativo. Questo perché tutte le modifiche all'app verranno distribuite tramite la pipeline. Tutto quello che devi fare è premere su. GitHub

Fasi della candidatura

Per definire un' AWS applicazione multi-stack che può essere aggiunta alla pipeline tutta in una volta, definite una sottoclasse di. `Stage` (Questo è diverso dal modulo `CdkStage` CDK Pipelines.)

Lo stage contiene gli stack che compongono l'applicazione. Se esistono dipendenze tra gli stack, gli stack vengono aggiunti automaticamente alla pipeline nell'ordine corretto. Gli stack che non dipendono l'uno dall'altro vengono distribuiti in parallelo. Puoi aggiungere una relazione di dipendenza tra gli stack chiamando. `stack1.addDependency(stack2)`

Gli stadi accettano un `env` argomento predefinito, che diventa l'ambiente predefinito per gli stack al suo interno. (Gli stack possono comunque avere il proprio ambiente specificato.).

Un'applicazione viene aggiunta alla pipeline chiamando `addStage()` con istanze di. `Stage` Una fase può essere istanziata e aggiunta alla pipeline più volte per definire fasi diverse della pipeline di applicazioni DTAP o multiregione.

Creeremo uno stack contenente una semplice funzione Lambda e lo posizioneremo in uno stage. Quindi aggiungeremo lo stage alla pipeline in modo che possa essere distribuito.

TypeScript

Crea il nuovo file `lib/my-pipeline-lambda-stack.ts` per contenere il nostro stack di applicazioni contenente una funzione 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";')
      });
    }
}
```

Crea il nuovo file `lib/my-pipeline-app-stage.ts` per ospitare il nostro 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');
    }
}
```

Modifica `lib/my-pipeline-stack.ts` per aggiungere lo stage alla nostra 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

Crea il nuovo file `lib/my-pipeline-lambda-stack.js` per contenere il nostro stack di applicazioni contenente una funzione 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 }
```

Crea il nuovo file `lib/my-pipeline-app-stage.js` per ospitare il nostro 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 };
```

Modifica `lib/my-pipeline-stack.ts` per aggiungere lo stage alla nostra 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

Crea il nuovo file `my_pipeline/my_pipeline_lambda_stack.py` per contenere il nostro stack di applicazioni contenente una funzione 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';")
        )
```

Crea il nuovo file `my_pipeline/my_pipeline_app_stage.py` per ospitare il nostro 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")
```

Modifica `my_pipeline/my-pipeline-stack.py` per aggiungere lo stage alla nostra 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

Crea il nuovo file `src/main/java/com.myorg/MyPipelineLambdaStack.java` per contenere il nostro stack di applicazioni contenente una funzione 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();

    }

}
```

Crea il nuovo file `src/main/java/com.myorg/MyPipelineAppStage.java` per ospitare il nostro 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");
    }

}
```

Modifica `src/main/java/com.myorg/MyPipelineStack.java` per aggiungere lo stage alla nostra 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#

Crea il nuovo file `src/MyPipeline/MyPipelineLambdaStack.cs` per contenere il nostro stack di applicazioni contenente una funzione 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';")
            });
        }
    }
}
```

Crea il nuovo file `src/MyPipeline/MyPipelineAppStage.cs` per ospitare il nostro 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");
        }
    }
}
```

Modifica `src/MyPipeline/MyPipelineStack.cs` per aggiungere lo stage alla nostra 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"
                }
            }));
        }
    }
}
```

Ogni fase dell'applicazione aggiunta da `addStage()` comporta l'aggiunta di una fase della pipeline corrispondente, rappresentata da un'StageDeploymentistanza restituita dalla `addStage()` chiamata. È possibile aggiungere azioni di pre-distribuzione o post-distribuzione alla fase chiamandone `addPre()` il metodo or. `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"));
```

Puoi aggiungere fasi a un Wave per distribuirle in parallelo, ad esempio quando distribuisci una fase su più account o regioni.

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"
    }
}));
```

Test delle implementazioni

Puoi aggiungere passaggi a una pipeline CDK per convalidare le distribuzioni che stai eseguendo. Ad esempio, puoi utilizzare le librerie CDK Pipeline per eseguire attività come `ShellStep` le seguenti:

- Tentativo di accesso a un Amazon API Gateway appena distribuito supportato da una funzione Lambda
- Verifica dell'impostazione di una risorsa distribuita mediante l'emissione di un comando AWS CLI

Nella sua forma più semplice, l'aggiunta di azioni di convalida è la seguente:

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'" }
}));
```

Molte AWS CloudFormation implementazioni comportano la generazione di risorse con nomi imprevedibili. Per questo motivo, CDK Pipelines fornisce un modo per AWS CloudFormation leggere gli output dopo una distribuzione. In questo modo è possibile passare (ad esempio) l'URL generato di un sistema di bilanciamento del carico a un'azione di test.

Per utilizzare gli output, esponi l'`CfnOutput`oggetto che ti interessa. Quindi, passalo nella `envFromCfnOutputs` proprietà di un passaggio per renderlo disponibile come variabile di ambiente all'interno di quel passaggio.

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" }
}));
```

Puoi scrivere semplici test di convalida direttamente in`ShellStep`, ma questo approccio diventa complicato quando il test dura più di poche righe. Per test più complessi, potete inserire file aggiuntivi (come script di shell completi o programmi in altri linguaggi) nella proprietà tramite the. `ShellStep` `inputs` Gli input possono essere qualsiasi passaggio che abbia un output, inclusa una fonte (come un GitHub repository) o un'altra. `ShellStep`

L'inserimento di file dal repository dei sorgenti è appropriato se i file sono direttamente utilizzabili nel test (ad esempio, se sono essi stessi eseguibili). In questo esempio, dichiariamo il nostro GitHub repository come `source` (anziché istanziarlo in linea come parte di). `CodePipeline` Quindi, passiamo questo set di file sia alla pipeline che al test di convalida.

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" }
}));
```

Ottenere i file aggiuntivi dalla fase di sintetizzazione è appropriato se i test devono essere compilati, operazione che viene eseguita come parte della sintesi.

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" }
}));
```

Note sulla sicurezza

Qualsiasi forma di distribuzione continua presenta rischi intrinseci per la sicurezza. In base al modello di responsabilità AWS condivisa, sei responsabile della sicurezza delle tue informazioni nel AWS cloud. La libreria CDK Pipelines ti offre un vantaggio iniziale incorporando impostazioni predefinite sicure e best practice di modellazione.

Tuttavia, per sua stessa natura, una libreria che necessita di un elevato livello di accesso per soddisfare lo scopo previsto non può garantire una sicurezza completa. Esistono molti vettori di attacco esterni AWS all'organizzazione.

In particolare, tenete presente quanto segue:

- Fai attenzione al software da cui dipendi. Controlla tutti i software di terze parti che utilizzi nella tua pipeline, perché possono modificare l'infrastruttura che viene implementata.
- Utilizza il blocco delle dipendenze per evitare aggiornamenti accidentali. CDK Pipelines `package-lock.json` rispetta `yarn.lock` e si assicura che le vostre dipendenze siano quelle che vi aspettate.
- CDK Pipelines funziona con risorse create nel tuo account e la configurazione di tali risorse è controllata dagli sviluppatori che inviano il codice tramite la pipeline. Pertanto, CDK Pipelines da solo non può proteggere da sviluppatori malintenzionati che cercano di aggirare i controlli di conformità. Se il tuo modello di minaccia include sviluppatori che scrivono codice CDK, dovresti disporre di meccanismi di conformità esterni come AWS CloudFormation Hooks (preventivo) o AWS Config (reattivo) che l' AWS CloudFormation Execution Role non dispone delle autorizzazioni per disabilitare.
- Le credenziali per gli ambienti di produzione dovrebbero avere vita breve. Dopo il bootstrap e il provisioning iniziale, non è affatto necessario che gli sviluppatori dispongano delle credenziali

dell'account. Le modifiche possono essere implementate tramite la pipeline. Riduci la possibilità di perdita delle credenziali evitando innanzitutto di averne bisogno.

Risoluzione dei problemi

I seguenti problemi si riscontrano comunemente quando si inizia a usare CDK Pipelines.

Pipeline: errore interno

```
CREATE_FAILED  | AWS::CodePipeline::Pipeline | Pipeline/Pipeline
Internal Failure
```

Controlla il tuo token di GitHub accesso. Potrebbe mancare o potrebbe non disporre delle autorizzazioni per accedere al repository.

Chiave: La policy contiene una dichiarazione con uno o più principi non validi

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey
Policy contains a statement with one or more invalid principals.
```

Uno degli ambienti di destinazione non è stato avviato con il nuovo stack bootstrap. Assicurati che tutti gli ambienti di destinazione siano avviati.

Lo stack è nello stato ROLLBACK_COMPLETE e non può essere aggiornato.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
ID: ...)
```

Lo stack non è riuscito nella distribuzione precedente ed è in uno stato non riutilizzabile. Elimina lo stack dalla AWS CloudFormation console e riprova la distribuzione.

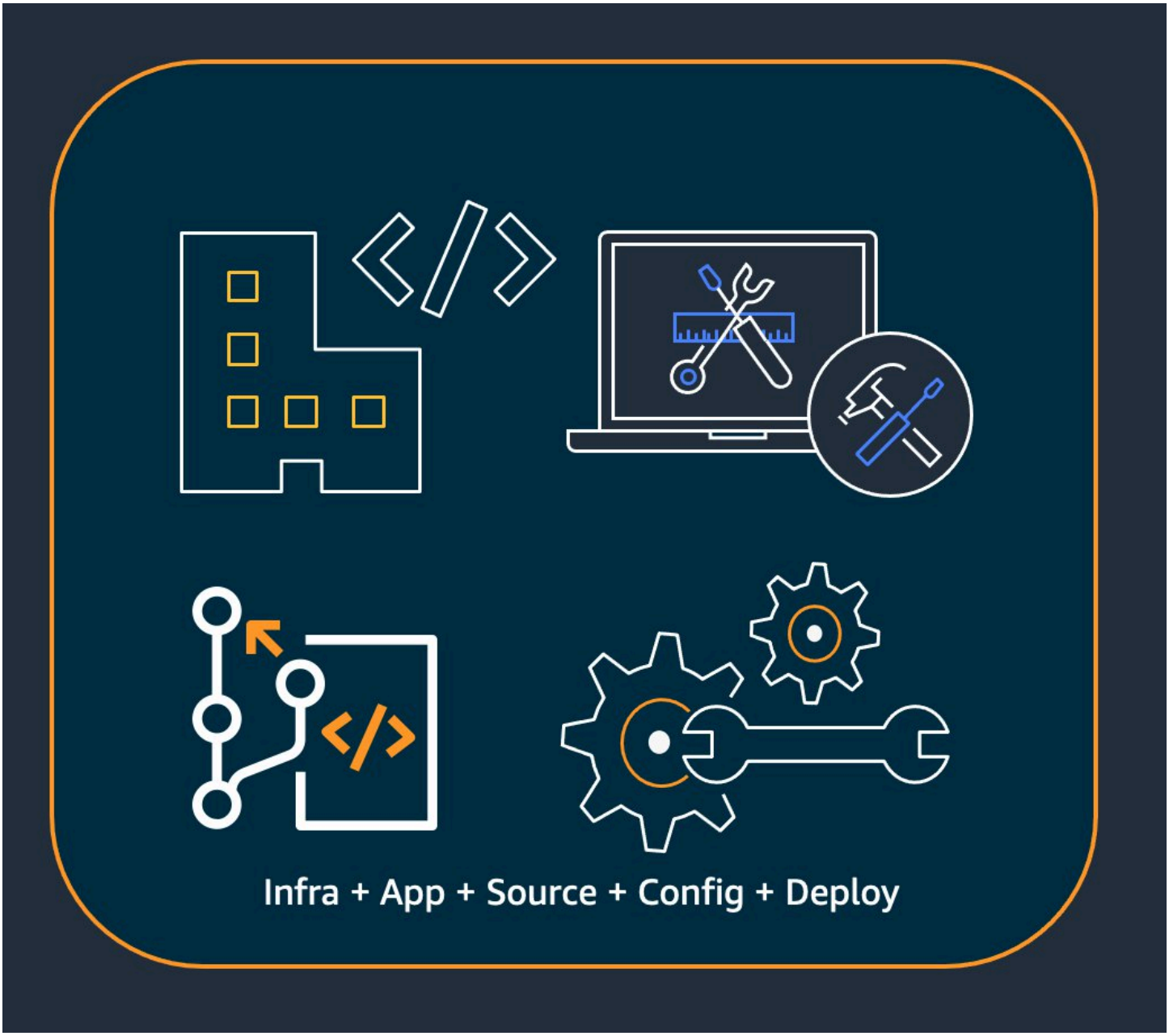
Le migliori pratiche per lo sviluppo e l'implementazione dell'infrastruttura cloud con AWS CDK

Con AWS CDK, gli sviluppatori o gli amministratori possono definire la propria infrastruttura cloud utilizzando un linguaggio di programmazione supportato. Le applicazioni CDK devono essere organizzate in unità logiche, come API, database e risorse di monitoraggio, e opzionalmente disporre di una pipeline per le implementazioni automatizzate. Le unità logiche devono essere implementate come costrutti, tra cui:

- Infrastruttura (ad esempio bucket Amazon S3, database Amazon RDS o una rete Amazon VPC)
- Codice di runtime (ad esempio funzioni) AWS Lambda
- Codice di configurazione

Gli stack definiscono il modello di implementazione di queste unità logiche. Per un'introduzione più dettagliata ai concetti alla base del CDK, vedere. Nozioni di base

AWS CDK Ciò riflette un'attenta considerazione delle esigenze dei nostri clienti e dei team interni e dei modelli di errore che spesso si verificano durante l'implementazione e la manutenzione continua di applicazioni cloud complesse. Abbiamo scoperto che gli errori sono spesso correlati a "out-of-band" modifiche apportate a un'applicazione che non sono state completamente testate, come le modifiche alla configurazione. Pertanto, lo abbiamo sviluppato AWS CDK attorno a un modello in cui l'intera applicazione è definita in codice, non solo la logica aziendale ma anche l'infrastruttura e la configurazione. In questo modo, le modifiche proposte possono essere esaminate attentamente, testate in modo completo in ambienti che assomigliano a vari livelli alla produzione e annullate completamente se qualcosa va storto.



Al momento dell'implementazione, AWS CDK sintetizza un assembly cloud che contiene quanto segue:

- AWS CloudFormation modelli che descrivono l'infrastruttura in tutti gli ambienti di destinazione
- Risorse di file che contengono il codice di runtime e i relativi file di supporto

Con il CDK, ogni commit nel ramo di controllo della versione principale dell'applicazione può rappresentare una versione completa, coerente e distribuibile dell'applicazione. L'applicazione può quindi essere distribuita automaticamente ogni volta che viene apportata una modifica.

La filosofia alla base AWS CDK ci porta alle nostre migliori pratiche consigliate, che abbiamo suddiviso in quattro grandi categorie.

- the section called “Best Practice dell'organizzazione”
- the section called “Le migliori pratiche di codifica”
- the section called “Costruisci le migliori pratiche”
- the section called “Le migliori pratiche applicative”

> **Tip**
>
> Considerate anche le best practice AWS CloudFormation e i singoli AWS servizi che utilizzate, ove applicabile all'infrastruttura definita da CDK.

Best Practice dell'organizzazione

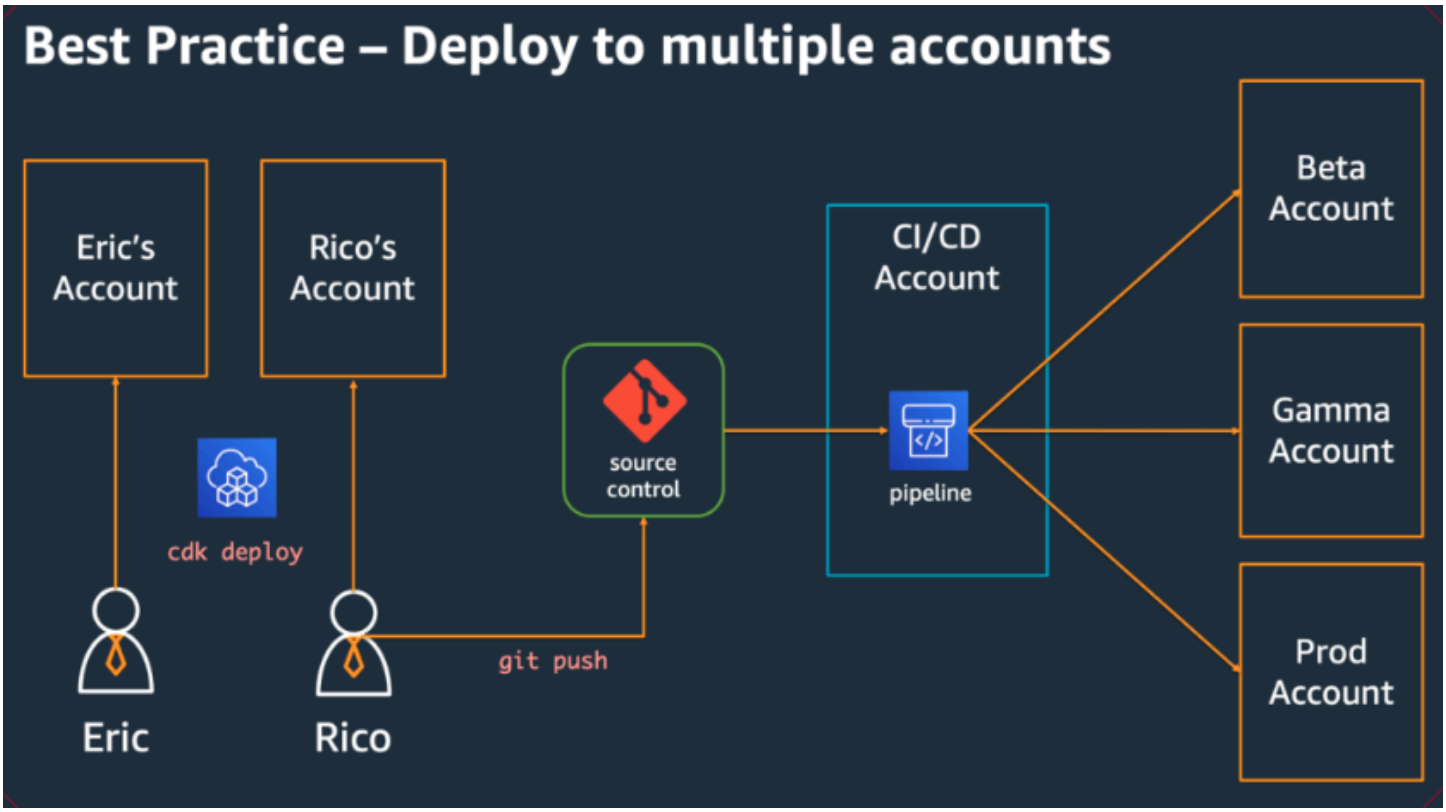
Nelle fasi iniziali dell' AWS CDK adozione, è importante considerare come preparare l'organizzazione per il successo. È consigliabile disporre di un team di esperti responsabile della formazione e della guida del resto dell'azienda nell'adozione del CDK. Le dimensioni di questo team possono variare, da una o due persone in una piccola azienda a un vero e proprio Cloud Center of Excellence (CCoE) presso un'azienda più grande. Questo team è responsabile della definizione degli standard e delle politiche per l'infrastruttura cloud dell'azienda e anche della formazione e del tutoraggio degli sviluppatori.

Il CCoE potrebbe fornire indicazioni su quali linguaggi di programmazione dovrebbero essere utilizzati per l'infrastruttura cloud. I dettagli variano da un'organizzazione all'altra, ma una buona politica aiuta a garantire che gli sviluppatori possano comprendere e gestire l'infrastruttura cloud dell'azienda.

Il CCoE crea anche una «landing zone» che definisce le unità organizzative all'interno. AWS Una landing zone è un AWS ambiente preconfigurato, sicuro, scalabile e multi-account basato su modelli di best practice. Per unire i servizi che compongono la tua landing zone, puoi utilizzare AWS Control Tower, che configura e gestisce l'intero sistema multi-account da un'unica interfaccia utente.

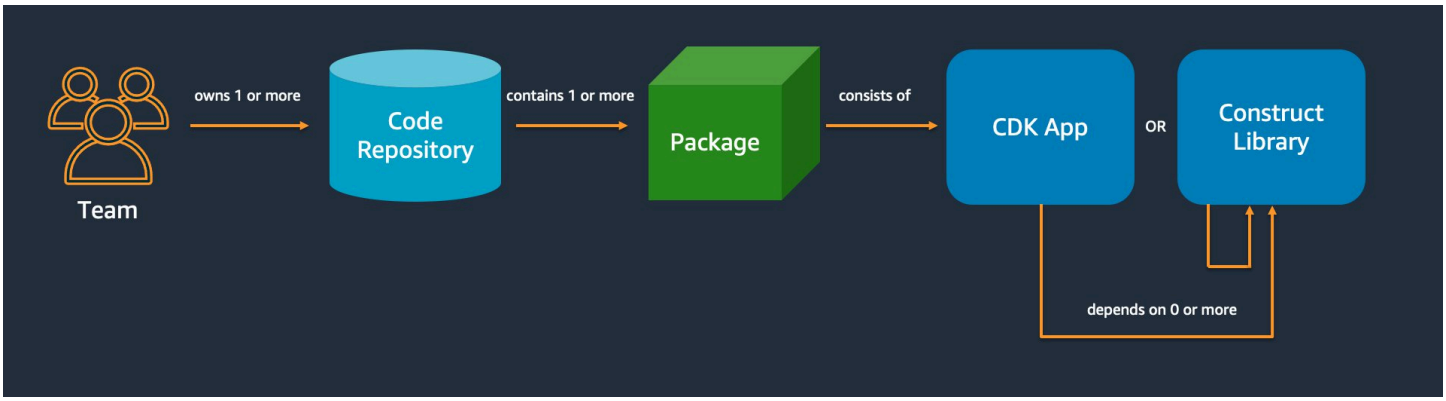
I team di sviluppo dovrebbero essere in grado di utilizzare i propri account per testare e distribuire nuove risorse in questi account, se necessario. I singoli sviluppatori possono trattare queste risorse come estensioni della propria workstation di sviluppo. Utilizzando CDK Pipelines, AWS CDK le applicazioni possono quindi essere distribuite tramite un account CI/CD in ambienti di test,

integrazione e produzione (ciascuno isolato nella propria regione o account). AWS Questo viene fatto unendo il codice degli sviluppatori nel repository canonico dell'organizzazione.



Le migliori pratiche di codifica

Questa sezione presenta le migliori pratiche per organizzare il AWS CDK codice. Il diagramma seguente mostra la relazione tra un team e gli archivi di codice, i pacchetti, le applicazioni e le librerie di costruzione del team.



Inizia in modo semplice e aggiungi complessità solo quando ne hai bisogno

Il principio guida della maggior parte delle nostre best practice è quello di mantenere le cose il più semplici possibile, ma non di più. Aggiungete complessità solo quando le vostre esigenze impongono una soluzione più complicata. Con AWS CDK, puoi rifattorizzare il codice secondo necessità per supportare nuovi requisiti. Non è necessario progettare in anticipo tutti gli scenari possibili.

Allineamento con il AWS Well-Architected Framework

Il AWS Well-Architected Framework definisce un componente come il codice, la configurazione AWS e le risorse che insieme soddisfano un requisito. Un componente è spesso l'unità di proprietà tecnica ed è disaccoppiato dagli altri componenti. Il termine carico di lavoro viene utilizzato per identificare un insieme di componenti che insieme forniscono valore aziendale. Un carico di lavoro è in genere il livello di dettaglio di cui comunicano i leader aziendali e tecnologici.

Un' AWS CDK applicazione è mappata a un componente come definito dal AWS Well-Architected Framework. AWS CDK le app sono un meccanismo per codificare e fornire le migliori pratiche per le applicazioni cloud Well-Architected. Puoi anche creare e condividere componenti come librerie di codice riutilizzabili tramite repository di artefatti, come. AWS CodeArtifact

Ogni applicazione inizia con un singolo pacchetto in un unico repository

Un singolo pacchetto è il punto di ingresso della tua AWS CDK app. Qui definisci come e dove distribuire le diverse unità logiche dell'applicazione. È inoltre necessario definire la pipeline CI/CD per distribuire l'applicazione. I costrutti dell'app definiscono le unità logiche della soluzione.

Utilizzate pacchetti aggiuntivi per i costrutti utilizzati in più di un'applicazione. (I costrutti condivisi dovrebbero inoltre avere il proprio ciclo di vita e la propria strategia di test.) Le dipendenze tra i pacchetti nello stesso repository sono gestite dagli strumenti di compilazione del repository.

Sebbene sia possibile, non consigliamo di inserire più applicazioni nello stesso repository, specialmente quando si utilizzano pipeline di distribuzione automatizzate. In questo modo si aumenta il «raggio d'azione» delle modifiche durante la distribuzione. Quando in un repository sono presenti più applicazioni, le modifiche apportate a un'applicazione attivano la distribuzione delle altre (anche se le altre non sono state modificate). Inoltre, un'interruzione in un'applicazione impedisce la distribuzione delle altre applicazioni.

Sposta il codice nei repository in base al ciclo di vita del codice o alla proprietà del team

Quando i pacchetti iniziano a essere utilizzati in più applicazioni, spostali nel rispettivo repository. In questo modo, i pacchetti possono essere referenziati dai sistemi di compilazione delle applicazioni che li utilizzano e possono anche essere aggiornati con cadenze indipendenti dai cicli di vita delle applicazioni. Tuttavia, all'inizio potrebbe avere senso mettere tutti i costrutti condivisi in un unico repository.

Inoltre, sposta i pacchetti nel proprio repository quando diversi team stanno lavorando su di essi. Questo aiuta a rafforzare il controllo degli accessi.

Per utilizzare i pacchetti oltre i confini del repository, è necessario un archivio di pacchetti privato, simile a NPM o Maven Central PyPi, ma interno all'organizzazione. È inoltre necessario un processo di rilascio che compili, verifichi e pubblichi il pacchetto nell'archivio privato dei pacchetti. CodeArtifactpuò ospitare pacchetti per i linguaggi di programmazione più diffusi.

Le dipendenze dai pacchetti nel repository dei pacchetti sono gestite dal gestore di pacchetti della lingua, come NPM for TypeScript o le applicazioni. JavaScript Il tuo gestore di pacchetti aiuta a garantire che le build siano ripetibili. Lo fa registrando le versioni specifiche di ogni pacchetto da cui dipende l'applicazione. Consente inoltre di aggiornare tali dipendenze in modo controllato.

I pacchetti condivisi richiedono una strategia di test diversa. Per una singola applicazione, potrebbe essere sufficiente distribuire l'applicazione in un ambiente di test e confermare che funzioni ancora. Tuttavia, i pacchetti condivisi devono essere testati indipendentemente dall'applicazione che li utilizza, come se fossero rilasciati al pubblico. (L'organizzazione potrebbe scegliere di rilasciare effettivamente alcuni pacchetti condivisi al pubblico.)

Tenete presente che un costrutto può essere arbitrariamente semplice o complesso. A `Bucket` è un costrutto, ma `CameraShopWebsite` potrebbe essere anche un costrutto.

L'infrastruttura e il codice di runtime risiedono nello stesso pacchetto

Oltre a generare AWS CloudFormation modelli per l'implementazione dell'infrastruttura, raggruppa AWS CDK anche risorse di runtime come funzioni Lambda e immagini Docker e le distribuisce insieme all'infrastruttura. In questo modo è possibile combinare il codice che definisce l'infrastruttura e il codice che implementa la logica di runtime in un unico costrutto. È consigliabile eseguire questa operazione. Non è necessario che questi due tipi di codice risiedano in repository o addirittura in pacchetti separati.

Per far evolvere insieme i due tipi di codice, è possibile utilizzare un costrutto autonomo che descriva completamente una funzionalità, incluse l'infrastruttura e la logica. Con un costrutto autonomo, puoi testare i due tipi di codice separatamente, condividere e riutilizzare il codice tra i progetti e creare una versione sincronizzata di tutto il codice.

Costruisci le migliori pratiche

Questa sezione contiene le migliori pratiche per lo sviluppo di costrutti. I costrutti sono moduli riutilizzabili e componibili che incapsulano risorse. Sono gli elementi costitutivi delle app. AWS CDK

Modella con costrutti, implementa con pile

Gli stack sono l'unità di distribuzione: tutto in uno stack viene distribuito insieme. Pertanto, quando create le unità logiche di livello superiore dell'applicazione a partire da più AWS risorse, rappresentate ogni unità logica come una, non come una `Construct`. `Stack` Utilizzate gli stack solo per descrivere come devono essere composti e collegati i costrutti per i vari scenari di implementazione.

Ad esempio, se una delle tue unità logiche è un sito Web, i costrutti che lo compongono (come un bucket Amazon S3, API Gateway, funzioni Lambda o tabelle Amazon RDS) devono essere composti in un unico costrutto di alto livello. Quindi quel costrutto deve essere istanziato in uno o più stack per la distribuzione.

Utilizzando costrutti per la costruzione e stack per l'implementazione, si migliora il potenziale di riutilizzo dell'infrastruttura e si ottiene una maggiore flessibilità nel modo in cui viene implementata.

Configura con proprietà e metodi, non con variabili di ambiente

Le ricerche delle variabili di ambiente all'interno di costrutti e pile sono un anti-pattern comune. Sia i costrutti che gli stack devono accettare un oggetto properties per consentire la completa configurabilità nel codice. In caso contrario si crea una dipendenza dalla macchina su cui verrà eseguito il codice, il che crea ulteriori informazioni di configurazione che è necessario monitorare e gestire.

In generale, le ricerche delle variabili di ambiente devono essere limitate al livello più alto di un'app. AWS CDK Dovrebbero essere utilizzati anche per trasmettere le informazioni necessarie per l'esecuzione in un ambiente di sviluppo. Per ulteriori informazioni, consulta the section called “Ambienti”.

Esegui un test unitario della tua infrastruttura

Per eseguire in modo coerente una suite completa di unit test in fase di compilazione in tutti gli ambienti, evita le ricerche in rete durante la sintesi e modella tutte le fasi di produzione in codice. (Queste best practice verranno illustrate più avanti). Se ogni singolo commit restituisce sempre lo stesso modello generato, puoi fidarti degli unit test che scrivi per confermare che i modelli generati abbiano l'aspetto che ti aspetti. Per ulteriori informazioni, consulta Costrutti di test.

Non modificate l'ID logico delle risorse stateful

La modifica dell'ID logico di una risorsa comporta la sostituzione della risorsa con una nuova alla successiva distribuzione. Per risorse statiche come database e bucket S3 o infrastrutture persistenti come Amazon VPC, questo è raramente ciò che si desidera. Fai attenzione a eventuali refactoring del AWS CDK codice che potrebbero causare la modifica dell'ID. Scrivi test unitari che affermino che gli ID logici delle tue risorse stateful rimangono statici. L'ID logico deriva da quello specificato quando si `id` crea un'istanza del costrutto e dalla posizione del costrutto nell'albero dei costrutti. Per ulteriori informazioni, consulta the section called “ID logici”.

I costrutti non sono sufficienti per la conformità

Molti clienti aziendali scrivono i propri wrapper per i costrutti L2 (i costrutti «curati» che rappresentano AWS risorse individuali con impostazioni predefinite e best practice integrate). Questi wrapper applicano le migliori pratiche di sicurezza come la crittografia statica e politiche IAM specifiche. Ad esempio, potresti crearne uno `MyCompanyBucket` da utilizzare nelle tue applicazioni al posto del solito costrutto Amazon S3`Bucket`. Questo modello è utile per far emergere le linee guida sulla sicurezza nelle prime fasi del ciclo di vita dello sviluppo del software, ma non utilizzarlo come unico mezzo di applicazione.

Utilizza invece AWS funzionalità come le politiche di controllo dei servizi e i limiti di autorizzazione per far rispettare le barriere di sicurezza a livello di organizzazione. Usa the section called “Aspetti” strumenti come CloudFormation Guard per fare affermazioni sulle proprietà di sicurezza degli elementi dell'infrastruttura prima della distribuzione. Usala AWS CDK per ciò che sa fare meglio.

Infine, tenete presente che scrivere i vostri costrutti «L2+» potrebbe impedire agli sviluppatori di sfruttare AWS CDK pacchetti come AWS Solutions Constructs o costrutti di terze parti di Construct Hub. Questi pacchetti sono in genere costruiti su AWS CDK costrutti standard e non saranno in grado di utilizzare i costrutti wrapper.

Le migliori pratiche applicative

In questa sezione spieghiamo come scrivere le AWS CDK applicazioni, combinando costrutti per definire il modo in cui le AWS risorse sono connesse.

Prendi decisioni al momento della sintesi

Sebbene AWS CloudFormation consenta di prendere decisioni al momento dell'implementazione (utilizzando `Conditions{ Fn::If }`, e`Parameters`) e consenta di accedere a questi meccanismi, si consiglia di non utilizzarli. AWS CDK I tipi di valori che è possibile utilizzare e i tipi di operazioni che è possibile eseguire su di essi sono limitati rispetto a quelli disponibili in un linguaggio di programmazione generico.

Provate invece a prendere tutte le decisioni, ad esempio il costrutto da istanziare, nell' AWS CDK applicazione utilizzando le istruzioni del linguaggio di `if` programmazione e altre funzionalità. Ad esempio, un linguaggio CDK comune, che esegue un'iterazione su un elenco e istanzia un costrutto con i valori di ogni elemento dell'elenco, semplicemente non è possibile utilizzare le espressioni. AWS CloudFormation

Consideralo un dettaglio di implementazione che AWS CDK utilizza per solide implementazioni cloud, non AWS CloudFormation come un obiettivo linguistico. Non stai scrivendo AWS CloudFormation modelli in TypeScript Python, stai scrivendo codice CDK che viene utilizzato CloudFormation per la distribuzione.

Usa nomi di risorse generate, non nomi fisici

I nomi sono una risorsa preziosa. Ogni nome può essere usato una sola volta. Pertanto, se codifichi un nome di tabella o un nome di bucket nell'infrastruttura e nell'applicazione, non puoi implementare quella parte di infrastruttura due volte nello stesso account. (Il nome di cui stiamo parlando qui è il nome specificato, ad esempio, dalla `bucketName` proprietà su un costrutto di bucket Amazon S3.)

Quel che è peggio, non è possibile apportare modifiche alla risorsa che richiedono la sua sostituzione. Se una proprietà può essere impostata solo al momento della creazione `KeySchema` della risorsa, ad esempio una tabella Amazon DynamoDB, tale proprietà è immutabile. La modifica di questa proprietà richiede una nuova risorsa. Tuttavia, la nuova risorsa deve avere lo stesso nome per essere una vera sostituta. Ma non può avere lo stesso nome mentre la risorsa esistente utilizza ancora quel nome.

Un approccio migliore consiste nello specificare il minor numero possibile di nomi. Se ometti i nomi delle risorse, li AWS CDK genererà automaticamente in modo da non causare problemi. Supponiamo di avere una tabella come risorsa. È quindi possibile passare il nome della tabella generata come variabile di ambiente alla AWS Lambda funzione. Nell' AWS CDK applicazione, è possibile fare riferimento al nome della tabella come`table.tableName`. In alternativa, puoi generare un file di configurazione sull'istanza Amazon EC2 all'avvio o scrivere il nome effettivo della tabella nel AWS Systems Manager Parameter Store in modo che l'applicazione possa leggerlo da lì.

Se il posto in cui ti serve è un'altra AWS CDK pila, è ancora più semplice. Supponendo che uno stack definisca la risorsa e un altro stack debba utilizzarla, vale quanto segue:

- Se i due stack si trovano nella stessa AWS CDK app, passa un riferimento tra i due stack. Ad esempio, salvate un riferimento al costrutto della risorsa come attributo dello stack di definizione (). `this.stack.uploadBucket = myBucket` Quindi, passa quell'attributo al costruttore dello stack che necessita della risorsa.
- Quando i due stack si trovano in AWS CDK app diverse, utilizza un `from` metodo statico per utilizzare una risorsa definita esternamente in base all'ARN, al nome o ad altri attributi. (Ad esempio, utilizzare `Table.fromArn()` per una tabella DynamoDB). Utilizzate il `CfnOutput` costrutto per stampare l'ARN o un altro valore richiesto nell'output `cdk deploy` di, oppure cercate in. AWS Management Console In alternativa, la seconda app può leggere il CloudFormation modello generato dalla prima app e recuperare quel valore dalla sezione. `Outputs`

Definisci le politiche di rimozione e la conservazione dei log

I AWS CDK tentativi di impedirti di perdere dati adottando per impostazione predefinita politiche che conservano tutto ciò che crei. Ad esempio, la politica di rimozione predefinita per le risorse che contengono dati (come i bucket Amazon S3 e le tabelle del database) prevede di non eliminare la risorsa quando viene rimossa dallo stack. Invece, la risorsa rimane orfana dallo stack. Analogamente, l'impostazione predefinita del CDK è quella di conservare tutti i log per sempre. Negli ambienti di produzione, queste impostazioni predefinite possono comportare rapidamente l'archiviazione di grandi quantità di dati che non sono effettivamente necessari e la relativa fattura. AWS

Valuta attentamente quali politiche desideri siano per ogni risorsa di produzione e specificale di conseguenza. the section called “Aspetti”Utilizzatele per convalidare le politiche di rimozione e registrazione nello stack.

Separa l'applicazione in più stack in base ai requisiti di implementazione

Non esiste una regola fissa per stabilire il numero di stack necessari all'applicazione. In genere si finisce per basare la decisione sui modelli di distribuzione. Tieni a mente le seguenti linee guida:

- In genere è più semplice tenere quante più risorse possibile nello stesso stack, quindi tienile insieme a meno che tu non sappia di volerle separare.
- Prendi in considerazione la possibilità di conservare le risorse con stato (come i database) in uno stack separato dalle risorse stateless. È quindi possibile attivare la protezione dalla terminazione nello stack stateful. In questo modo, puoi distruggere o creare liberamente più copie dello stack stateless senza il rischio di perdita di dati.
- Le risorse stateful sono più sensibili alla ridenominazione dei costrutti: la ridenominazione comporta la sostituzione delle risorse. Pertanto, non inserite risorse con stato all'interno di costrutti che potrebbero essere spostati o rinominati (a meno che lo stato non possa essere ricostruito in caso di perdita, come una cache). Questa è un'altra buona ragione per inserire le risorse stateful nel proprio stack.

Impegnati a evitare comportamenti **`cdk.context.json`** non deterministici

Il determinismo è fondamentale per implementazioni di successo. AWS CDK Un' AWS CDK app dovrebbe avere essenzialmente lo stesso risultato ogni volta che viene distribuita in un determinato ambiente.

Poiché AWS CDK l'app è scritta in un linguaggio di programmazione generico, può eseguire codice arbitrario, utilizzare librerie arbitrarie ed effettuare chiamate di rete arbitrarie. Ad esempio, puoi utilizzare un AWS SDK per recuperare alcune informazioni dal tuo account mentre sintetizzi l'app. AWS Riconosci che così facendo comporterai requisiti aggiuntivi per la configurazione delle credenziali, una maggiore latenza e una possibilità, per quanto piccola, di fallire ogni volta che esegui. `cdk synth`

Non modificate mai il vostro AWS account o le vostre risorse durante la sintesi. La sintesi di un'app non dovrebbe avere effetti collaterali. Le modifiche all'infrastruttura dovrebbero avvenire solo nella fase di implementazione, dopo la generazione del AWS CloudFormation modello. In questo modo, se c'è un problema, AWS CloudFormation puoi ripristinare automaticamente la modifica. Per apportare modifiche che non possono essere apportate facilmente all'interno del AWS CDK framework, utilizza risorse personalizzate per eseguire codice arbitrario al momento della distribuzione.

Anche le chiamate strettamente di sola lettura non sono necessariamente sicure. Considerate cosa succede se il valore restituito da una chiamata di rete cambia. Su quale parte della tua infrastruttura avrà questo impatto? Cosa accadrà alle risorse già impiegate? Di seguito sono riportati due esempi di situazioni in cui una modifica improvvisa dei valori potrebbe causare un problema.

- Se effettui il provisioning di un Amazon VPC in tutte le zone di disponibilità disponibili in una regione specifica e il numero di AZ è due il giorno dell'implementazione, lo spazio IP viene diviso a metà. Se il giorno successivo AWS viene avviata una nuova zona di disponibilità, la distribuzione successiva tenta di suddividere lo spazio IP in tre parti, richiedendo la ricreazione di tutte le sottoreti. Questo probabilmente non sarà possibile perché le tue istanze Amazon EC2 sono ancora in esecuzione e dovrai pulirle manualmente.
- Se richiedi l'immagine più recente della macchina Amazon Linux e distribuisci un'istanza Amazon EC2 e il giorno successivo viene rilasciata una nuova immagine, una distribuzione successiva rileva la nuova AMI e sostituisce tutte le tue istanze. Potrebbe non essere quello che ti aspettavi che accadesse.

Queste situazioni possono essere pericolose perché il AWS cambiamento potrebbe verificarsi dopo mesi o anni di implementazioni riuscite. Improvvisamente le vostre implementazioni falliscono «senza motivo» e molto tempo fa avete dimenticato cosa avete fatto e perché.

Fortunatamente, AWS CDK include un meccanismo chiamato context providers per registrare un'istantanea di valori non deterministici. Ciò consente alle future operazioni di sintesi di produrre esattamente lo stesso modello utilizzato quando sono state implementate per la prima volta. Le uniche modifiche al nuovo modello sono le modifiche apportate al codice. Quando si utilizza il `.fromLookup()` metodo di un costrutto, il risultato della chiamata viene memorizzato nella cache. `cdk.context.json` Dovresti affidarlo al controllo della versione insieme al resto del codice per assicurarti che le future esecuzioni della tua app CDK utilizzino lo stesso valore. Il CDK Toolkit include comandi per gestire la cache contestuale, in modo da poter aggiornare voci specifiche quando necessario. Per ulteriori informazioni, consulta the section called “Context”.

Se avete bisogno di un valore (proveniente AWS o altrove) per il quale non esiste un provider di contesto CDK nativo, vi consigliamo di scrivere uno script separato. Lo script dovrebbe recuperare il valore e scriverlo in un file, quindi leggerlo nell'app CDK. Esegui lo script solo quando desideri aggiornare il valore memorizzato, non come parte del normale processo di compilazione.

Lascia che AWS CDK gestiscano ruoli e gruppi di sicurezza

Con i metodi di `grant()` praticità della libreria AWS CDK construct, puoi creare AWS Identity and Access Management ruoli che garantiscono l'accesso a una risorsa da parte di un'altra utilizzando autorizzazioni con ambito minimo. Ad esempio, considera una riga come la seguente:

```
myBucket.grantRead(myLambda)
```

Questa singola riga aggiunge una policy al ruolo della funzione Lambda (anch'essa creata per te). Quel ruolo e le relative politiche sono più di una dozzina di righe CloudFormation che non devi scrivere. AWS CDK Concede solo le autorizzazioni minime richieste alla funzione per leggere dal bucket.

Se si richiede agli sviluppatori di utilizzare sempre ruoli predefiniti creati da un team di sicurezza, la AWS CDK codifica diventa molto più complicata. I vostri team potrebbero perdere molta flessibilità nel modo in cui progettano le loro applicazioni. Un'alternativa migliore consiste nell'utilizzare le politiche di controllo dei servizi e i limiti di autorizzazione per assicurarsi che gli sviluppatori rimangano entro i limiti.

Modella tutte le fasi di produzione in codice

Negli AWS CloudFormation scenari tradizionali, l'obiettivo è produrre un singolo artefatto parametrizzato in modo che possa essere distribuito in vari ambienti di destinazione dopo aver applicato valori di configurazione specifici a tali ambienti. Nel CDK, è possibile e si deve inserire tale configurazione nel codice sorgente. Crea uno stack per il tuo ambiente di produzione e crea uno stack separato per ciascuna delle altre fasi. Quindi, inserisci i valori di configurazione per ogni stack nel codice. Utilizzate servizi come Secrets Manager e Systems Manager Parameter Store per valori sensibili che non desiderate trasferire al controllo del codice sorgente, utilizzando i nomi o gli ARN di tali risorse.

Quando sintetizzi l'applicazione, l'assembly cloud creato nella `cdk.out` cartella contiene un modello separato per ogni ambiente. L'intera build è deterministica. Non ci sono out-of-band modifiche all'applicazione e ogni commit restituisce sempre lo stesso identico AWS CloudFormation modello e le stesse risorse di accompagnamento. Ciò rende i test unitari molto più affidabili.

Misura tutto

Il raggiungimento dell'obiettivo di un'implementazione completa e continua, senza intervento umano, richiede un elevato livello di automazione. Tale automazione è possibile solo con un'ampia quantità di

monitoraggio. Per misurare tutti gli aspetti delle risorse distribuite, crea metriche, allarmi e dashboard. Non limitarti a misurare cose come l'utilizzo della CPU e lo spazio su disco. Registra anche le metriche aziendali e utilizza tali misurazioni per automatizzare le decisioni di implementazione, come i rollback. La maggior parte dei costrutti L2 include metodi pratici per aiutarti a creare metriche, come il metodo sulla classe DynamoDB.Table. AWS CDK`metricUserErrors()`

AWS CDK riferimento

Questa sezione contiene informazioni di riferimento per AWS Cloud Development Kit (AWS CDK).

Argomenti

Riferimento API

L'API Reference contiene informazioni sulla AWS Construct Library e altre API fornite da. AWS Cloud Development Kit (AWS CDK) La maggior parte della AWS Construct Library è contenuta in un singolo pacchetto chiamato con il suo TypeScript nome:. `aws-cdk-lib` Il nome effettivo del pacchetto varia in base alla lingua. Sono disponibili versioni separate del riferimento API per ogni linguaggio di programmazione supportato.

Il riferimento all'API CDK è organizzato in sottomoduli. Esistono uno o più sottomoduli per ciascuno. Servizio AWS

Ogni sottomodulo ha una panoramica che include informazioni su come utilizzare le sue API. Ad esempio, la panoramica di S3 mostra come impostare la crittografia predefinita su un bucket Amazon Simple Storage Service (Amazon S3).

AWS CDK controllo delle versioni

Questo argomento fornisce informazioni di riferimento su come il sistema AWS Cloud Development Kit (AWS CDK) gestisce il controllo delle versioni.

I numeri di versione sono costituiti da tre parti numeriche: principale. minore. correggi e aderisci rigorosamente al modello di versionamento semantico. Ciò significa che le modifiche sostanziali alle API stabili sono limitate alle versioni principali.

Le versioni secondarie e le patch sono retrocompatibili. Il codice scritto in una versione precedente con la stessa versione principale può essere aggiornato a una versione più recente all'interno della stessa versione principale. Inoltre, continuerà a essere compilato ed eseguito, producendo lo stesso risultato.

Argomenti

AWS CDKCLIcompatibilità

AWS CDK CLIè sempre compatibile con le librerie di costruzione con un numero di versione semanticamente inferiore o uguale. Pertanto, è sempre sicuro aggiornarli AWS CDK CLI all'interno della stessa versione principale.

Non AWS CDK CLI è sempre compatibile con le librerie di costruzione di una versione semanticamente superiore. La compatibilità dipende dal fatto che i due componenti utilizzino la stessa versione dello schema di assemblaggio cloud. Il AWS CDK framework genera un assemblaggio cloud durante la sintesi e AWS CDK CLI lo utilizza per la distribuzione. Lo schema che definisce il formato dell'assembly cloud è rigorosamente specificato e versionato.

AWS le librerie di build che utilizzano una determinata versione dello schema di assemblaggio cloud sono compatibili con AWS CDK CLI le versioni che utilizzano quella versione dello schema o successive. Ciò potrebbe includere versioni di AWS CDK CLI che sono precedenti a una determinata versione della libreria di costruzioni.

Quando la versione di cloud assembly richiesta dalla libreria di costruzione non è compatibile con la versione supportata da AWS CDK CLI, viene visualizzato un messaggio di errore simile al seguente:

```
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.
```

Per risolvere questo errore, aggiornalo AWS CDK CLI a una versione compatibile con la versione di cloud assembly richiesta o all'ultima versione disponibile. L'alternativa (il downgrade dei moduli della libreria di costruzione utilizzati dall'app) è generalmente sconsigliata.

> **Note**
>
> Per maggiori dettagli sullo schema di assemblaggio cloud, consulta Cloud Assembly Versioning.

AWS Controllo delle versioni di Construct Library

I moduli della AWS Construct Library attraversano varie fasi man mano che vengono sviluppati dall'idea all'API matura. Le diverse fasi offrono diversi gradi di stabilità delle API nelle versioni successive di. AWS CDK

Le API nella AWS CDK libreria principale sono stabili e la libreria dispone di versioni semantiche complete. `aws-cdk-lib` Questo pacchetto include costrutti AWS CloudFormation (L1) per tutti i AWS servizi e tutti i moduli stabili di livello superiore (L2 e L3). (Include anche le classi CDK principali come e). `App` `Stack` Le API non verranno rimosse da questo pacchetto (sebbene possano essere obsolete) fino alla prossima versione principale del CDK. Nessuna singola API subirà mai modifiche sostanziali. Quando è necessaria una modifica sostanziale, verrà aggiunta un'API completamente nuova.

Le nuove API in fase di sviluppo per un servizio già incorporato `aws-cdk-lib` vengono identificate utilizzando un `Beta`*`N`* suffisso, che `N` parte da 1 e viene incrementato a ogni modifica sostanziale apportata alla nuova API. `Beta`*`N`* Le API non vengono mai rimosse, ma solo obsolete, quindi l'app esistente continua a funzionare con le versioni più recenti di. `aws-cdk-lib` Quando l'API viene considerata stabile, viene aggiunta una nuova API senza il suffisso. `Beta`*`N`*

Quando si iniziano a sviluppare API di livello superiore (L2 o L3) per un AWS servizio che in precedenza aveva solo API L1, tali API vengono inizialmente distribuite in un pacchetto separato. Il nome di tale pacchetto ha il suffisso «Alpha» e la sua versione corrisponde alla prima versione compatibile con una versione secondaria. `aws-cdk-lib` `alpha` Quando il modulo supporta i casi d'uso previsti, vengono aggiunte le sue API. `aws-cdk-lib`

Stabilità del legame linguistico

Nel corso del tempo, potremmo aggiungere il supporto AWS CDK per altri linguaggi di programmazione. Sebbene l'API descritta in tutte le lingue sia la stessa, il modo in cui l'API viene espressa varia a seconda della lingua e potrebbe cambiare con l'evoluzione del supporto linguistico. Per questo motivo, le associazioni linguistiche sono considerate sperimentali per un certo periodo, fino a quando non vengono considerate pronte per l'uso in produzione.

Language	Stability
TypeScript	Stable
JavaScript	Stable

Language	Stability
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK tutorial

Questa sezione contiene tutorial per. AWS Cloud Development Kit (AWS CDK)

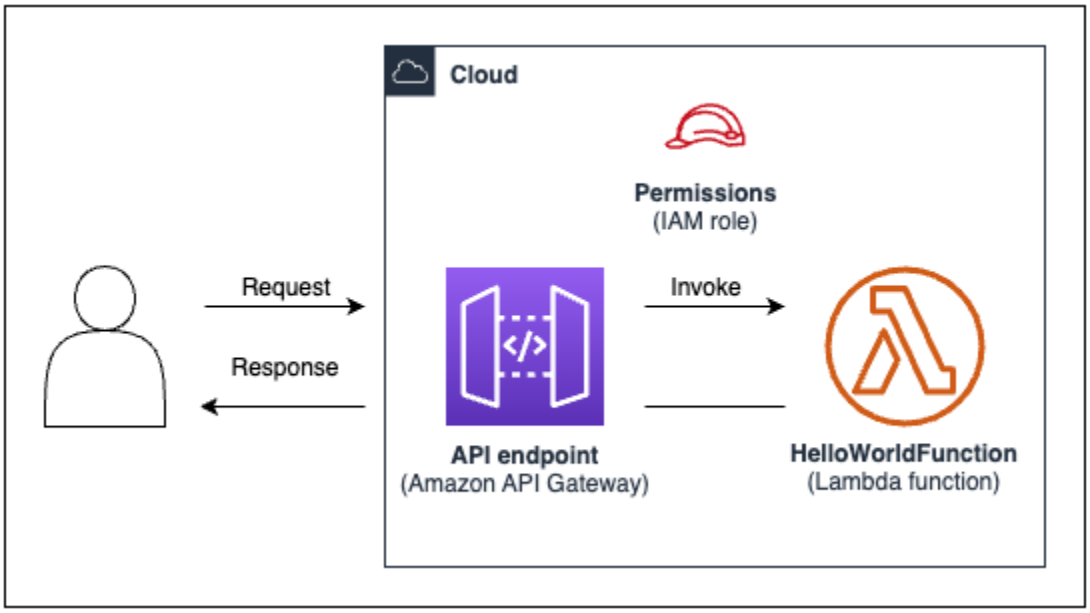
Argomenti

Crea un'applicazione Hello World senza server

In questo tutorial, si utilizza AWS Cloud Development Kit (AWS CDK) per creare una semplice Hello World applicazione serverless che implementa un backend API di base creando quanto segue:

- Amazon API Gateway REST API: fornisce un endpoint HTTP che viene utilizzato per richiamare la funzione tramite una HTTP GET richiesta.
- AWS Lambda function — Funzione che restituisce un `Hello World!` messaggio quando viene richiamata con l'endpoint. HTTP
- Integrazioni e autorizzazioni: dettagli di configurazione e autorizzazioni per consentire alle risorse di interagire tra loro ed eseguire azioni, come la scrittura di log su Amazon. CloudWatch

Il diagramma seguente mostra i componenti di questa applicazione:



Per questo tutorial, completerai quanto segue:

1. Crea un AWS CDK progetto.
2. Definisci una funzione Lambda e l'API REST di API Gateway utilizzando i costrutti L2 della Construct Library. AWS
3. Distribuisci la tua applicazione su. Cloud AWS
4. Interagisci con la tua applicazione in. Cloud AWS
5. Eliminare l'applicazione di esempio da Cloud AWS.

Prerequisiti

Prima di iniziare questo tutorial, completa quanto segue:

- Crea un Account AWS file e installa e configura AWS Command Line Interface (AWS CLI).
- Installa Node.js enpm.
- Installa CDK Toolkit a livello globale, utilizzando. `npm install -g aws-cdk`

Per ulteriori informazioni, consulta Iniziare con AWS CDK.

Consigliamo inoltre una conoscenza di base di quanto segue:

- Che cos'è il AWS CDK?per un'introduzione di base a AWS CDK.
- AWS CDK concettiper una panoramica dei concetti fondamentali di AWS CDK.

Fase 1: Creare un progetto CDK

In questo passaggio, si crea un nuovo progetto CDK utilizzando il AWS CDK CLI `cdk init` comando.

Per creare un progetto CDK

1. Da una directory iniziale a tua scelta, crea e naviga fino a una directory di progetto denominata `cdk-hello-world` sul tuo computer:

```
$ mkdir cdk-hello-world && cd cdk-hello-world
```

2. Usa il `cdk init` comando per creare un nuovo progetto nel tuo linguaggio di programmazione preferito:

   TypeScript

   ```
   $ cdk init --language typescript
   ```

   Installa AWS CDK le librerie:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   JavaScript

   ```
   $ cdk init --language javascript
   ```

   Installa AWS CDK le librerie:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   Python

   ```
   $ cdk init --language python
   ```

   Attiva l'ambiente virtuale:

   ```
   $ source .venv/bin/activate
   ```

   Installa AWS CDK le librerie e le dipendenze del progetto:

   ```
   (.venv)$ python3 -m pip install -r requirements.txt
   ```

   Java

   ```
   $ cdk init --language java
   ```

   Installa AWS CDK le librerie e le dipendenze del progetto:

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

Installa AWS CDK le librerie e le dipendenze del progetto:

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

Installa le dipendenze del progetto:

```
$ go mod tidy
```

Il CDK CLI crea un progetto con la seguente struttura:

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
```

Il CDK crea CLI automaticamente un'app CDK che contiene un singolo stack. L'istanza dell'app CDK viene creata dalla classe. App Quanto segue è una parte del file dell'applicazione CDK:

TypeScript

Si trova in`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

Situato in`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

Situato in`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

Situato in`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#

Situato in`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

Situato in`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
}
```

Fase 2: Crea la tua funzione Lambda

All'interno del progetto CDK, create una `lambda` directory che includa un nuovo `hello.js` file. Di seguito è riportato un esempio:

TypeScript

Dalla radice del progetto, esegui quanto segue:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Quanto segue dovrebbe ora essere aggiunto al tuo progetto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

JavaScript

Dalla radice del progetto, esegui quanto segue:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Quanto segue dovrebbe ora essere aggiunto al tuo progetto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

Dalla radice del progetto, esegui quanto segue:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Quanto segue dovrebbe ora essere aggiunto al tuo progetto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

Dalla radice del progetto, esegui quanto segue:

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

Quanto segue dovrebbe ora essere aggiunto al tuo progetto CDK:

```
cdk-hello-world
```

```
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

Dalla radice del progetto, esegui quanto segue:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Quanto segue dovrebbe ora essere aggiunto al tuo progetto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

Dalla radice del progetto, esegui quanto segue:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Quanto segue dovrebbe ora essere aggiunto al tuo progetto CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

> **Note**
>
> Per semplificare questo tutorial, utilizziamo una funzione JavaScript Lambda per tutti i linguaggi di programmazione CDK.

Definisci la tua funzione Lambda aggiungendo quanto segue al file appena creato:

```
exports.handler = async (event) => {
    return {
        statusCode: 200,
        headers: { "Content-Type": "text/plain" },
        body: JSON.stringify({ message: "Hello, World!" }),
    };
};
```

Fase 3: Definisci i tuoi costrutti

In questo passaggio, definirai le tue risorse Lambda e API Gateway utilizzando costrutti AWS CDK L2.

Apri il file di progetto che definisce lo stack CDK. Modificherete questo file per definire i vostri costrutti. Di seguito è riportato un esempio del file stack iniziale:

TypeScript

Situato in`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

Situato in`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

Situato in`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

Situato in`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#

Situato in`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

Situato in`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
}
```

In questo file, AWS CDK sta facendo quanto segue:

- L'istanza dello stack CDK viene istanziata dalla classe. `Stack`
- La classe `Constructs` base viene importata e fornita come ambito o elemento principale dell'istanza stack.

Definisci la risorsa della tua funzione Lambda

Per definire la risorsa della funzione Lambda, importate e utilizzate il costrutto `aws-lambda` L2 dalla Construct Library. AWS

Modificate il file dello stack come segue:

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**
>
> Importiamo il `aws_lambda` modulo `_lambda` perché `lambda` è un identificatore incorporato. 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
}

// ...
```

Qui si crea una risorsa per la funzione Lambda e si definiscono le seguenti proprietà:

- `runtime`— L'ambiente in cui viene eseguita la funzione. Qui, usiamo la Node.js versione20.x.
- `code`— Il percorso del codice della funzione sul computer locale.
- `handler`— Il nome del file specifico che contiene il codice della funzione.

Definisci la tua REST API risorsa API Gateway

Per definire la REST API risorsa API Gateway, importate e utilizzate il costrutto `aws-apigateway` L2 dalla AWS Construct Library.

Modificate il file dello stack come segue:

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
```

```
}

// ...
```

Qui puoi creare una REST API risorsa API Gateway, insieme a quanto segue:

- Un'integrazione tra la REST API e la tua funzione Lambda, che consente all'API di richiamare la tua funzione. Ciò include la creazione di una risorsa di autorizzazione Lambda.
- Una nuova risorsa o percorso denominato `hello` che viene aggiunto alla radice dell'endpoint dell'API. Questo crea un nuovo endpoint che si aggiunge `/hello` alla tua base. URL
- Un metodo GET per la `hello` risorsa. Quando una richiesta GET viene inviata all'`/hello`endpoint, viene richiamata la funzione Lambda e viene restituita la relativa risposta.

Fase 4: Preparare l'applicazione per la distribuzione

In questa fase preparate l'applicazione per la distribuzione compilando, se necessario, ed eseguendo la convalida di base con il AWS CDK CLI `cdk synth` comando.

Se necessario, crea la tua applicazione:

TypeScript

Dalla radice del progetto, esegui quanto segue:

```
$ npm run build
```

JavaScript

La costruzione non è necessaria.

Python

L'edificio non è richiesto.

Java

Dalla radice del progetto, esegui quanto segue:

```
$ mvn package
```

C#

Dalla radice del progetto, esegui quanto segue:

```
$ dotnet build src
```

Go

Dalla radice del progetto, esegui quanto segue:

```
$ go build
```

Esegui `cdk synth` per sintetizzare un AWS CloudFormation modello dal tuo codice CDK. Utilizzando i costrutti L2, molti dei dettagli di configurazione richiesti da AWS CloudFormation per facilitare l'interazione tra la funzione Lambda REST API vengono forniti da. AWS CDK

Dalla radice del progetto, esegui quanto segue:

```
$ cdk synth
```

> **Note**
>
> Se ricevi un errore come il seguente, verifica di essere nella `cdk-hello-world` directory e riprova:
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

In caso di successo, AWS CDK CLI restituirà il AWS CloudFormation modello in YAML formato al prompt dei comandi. Nella cartella viene inoltre salvato un modello JSON formattato`cdk.out`.

Di seguito è riportato un esempio di output del AWS CloudFormation modello:

AWS CloudFormation modello

```
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.
```

Utilizzando i costrutti L2, si definiscono alcune proprietà per configurare le risorse e si utilizzano metodi di supporto per integrarle insieme. AWS CDK configura la maggior parte delle AWS CloudFormation risorse e delle proprietà necessarie per il provisioning dell'applicazione.

Fase 5: distribuzione dell'applicazione

In questo passaggio, si utilizza il AWS CDK CLI `cdk deploy` comando per distribuire l'applicazione. AWS CDK Funziona con il AWS CloudFormation servizio per fornire le tue risorse.

> **Important**
>
> È necessario eseguire un avvio unico dell' AWS ambiente prima della distribuzione. Per istruzioni, consulta Bootstrap your environment.

Dalla radice del progetto, esegui quanto segue. Conferma le modifiche se richiesto:

```
$ cdk deploy

#  Synthesis time: 2.44s

...

Do you wish to deploy these changes (y/n)? y
```

Al termine della distribuzione, AWS CDK CLI verrà visualizzato l'URL dell'endpoint. Copia questo URL per il passaggio successivo. Di seguito è riportato un esempio:

```
...
#  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
...
```

Fase 6: Interagisci con l'applicazione

In questo passaggio, avviate una richiesta GET all'endpoint API e ricevete la risposta della funzione Lambda.

Individua l'URL dell'endpoint indicato nel passaggio precedente e aggiungi il percorso. `/hello` Quindi, utilizzando il browser o il prompt dei comandi, invia una richiesta GET al tuo endpoint. Di seguito è riportato un esempio:

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello
{"message":"Hello World!"}%
```

Congratulazioni, hai creato, distribuito e interagito con successo la tua applicazione utilizzando! AWS CDK

Passaggio 7: Eliminare l'applicazione

In questo passaggio, si utilizza AWS CDK CLI per eliminare l'applicazione da Cloud AWS.

Per eliminare l'applicazione, eseguicdk `destroy`. Quando richiesto, conferma la richiesta di eliminazione dell'applicazione:

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
 #  CdkHelloWorldStack: destroyed
```

Risoluzione dei problemi

Errore: {«message»: «Errore interno del server»}%

Quando si richiama la funzione Lambda distribuita, si riceve questo errore. Questo errore può verificarsi per diversi motivi.

Per risolvere ulteriormente i problemi

Usa AWS CLI per richiamare la tua funzione Lambda.

1. Modifica il tuo file stack per acquisire il valore di output del nome della funzione Lambda distribuita. Di seguito è riportato un esempio:

```
...

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. Implementa nuovamente l'applicazione. AWS CDK CLIProdurrà il valore del nome della funzione Lambda distribuita:

```
$ cdk deploy

#  Synthesis time: 0.29s
...
 #  CdkHelloWorldStack

#  Deployment time: 20.36s

Outputs:
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. Usa AWS CLI per richiamare la tua funzione Lambda in e inviare Cloud AWS la risposta a un file di testo:

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-identifier output.txt
```

4. Controlla `output.txt` per vedere i risultati.

Possibile causa: la risorsa API Gateway non è definita correttamente nel file stack.

Se `output.txt` mostra una risposta corretta della funzione Lambda, il problema potrebbe riguardare il modo in cui hai definito l'API REST dell'API Gateway. AWS CLI richiama la tua Lambda direttamente, non tramite l'endpoint. Controlla il codice per assicurarti che corrisponda a questo tutorial. Quindi, esegui nuovamente l'implementazione.

Possibile causa: la risorsa Lambda è definita in modo errato nel file stack.

Se `output.txt` restituisce un errore, il problema potrebbe riguardare il modo in cui hai definito la funzione Lambda. Controlla il codice per assicurarti che corrisponda a questo tutorial. Quindi esegui nuovamente l'implementazione.

Crea un'app con più stack

Puoi creare un' AWS Cloud Development Kit (AWS CDK) applicazione contenente più stack. Quando distribuisci l' AWS CDK app, ogni stack diventa un modello a sé stante. AWS CloudFormation Puoi anche sintetizzare e distribuire ogni stack singolarmente utilizzando il comando. AWS CDK CLI `cdk deploy`

Questo tutorial tratta quanto segue:

- Come estendere la `Stack` classe per accettare nuove proprietà o argomenti.
- Come utilizzare le proprietà per determinare quali risorse contiene lo stack e la loro configurazione.
- Come creare istanze di più stack da questa classe.

L'esempio in questo argomento utilizza una proprietà booleana denominata (`encryptBucket`Python:). `encrypt_bucket` Indica se un bucket Amazon S3 deve essere crittografato. In tal caso, lo stack abilita la crittografia utilizzando una chiave gestita da AWS Key Management Service ().AWS KMS L'app crea due istanze di questo stack, una con crittografia e una senza.

Argomenti

Prima di iniziare

Innanzitutto, installa Node.js e gli strumenti da riga di AWS CDK comando, se non l'hai già fatto. Per informazioni dettagliate, vedi Iniziare con AWS CDK.

Quindi, crea un AWS CDK progetto inserendo i seguenti comandi nella riga di 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
```

Puoi importare il progetto Maven risultante nel tuo IDE Java.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

È possibile aprire il file `src/Pipeline.sln` in Visual Studio.

Aggiungi un parametro opzionale

L'`props`argomento del `Stack` costruttore soddisfa l'interfaccia. `StackProps` In questo esempio, vogliamo che lo stack accetti una proprietà aggiuntiva che ci dica se crittografare il bucket Amazon S3. Dovremmo creare un'interfaccia o una classe che includa la proprietà. Ciò consente al compilatore di assicurarsi che la proprietà abbia un valore booleano e ne abilita il completamento automatico nell'IDE.

Quindi aprite il file sorgente indicato nel vostro IDE o editor e aggiungete la nuova interfaccia, classe o argomento. Il codice dovrebbe avere questo aspetto dopo le modifiche. Le righe che abbiamo aggiunto sono mostrate in grassetto.

TypeScript

File: `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

Archiviazione: `lib/multistack-stack.js`

JavaScript non ha una funzionalità di interfaccia; non è necessario aggiungere alcun codice.

```
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

File: `multistack/multistack_stack.py`

Python non ha una funzionalità di interfaccia, quindi estenderemo il nostro stack per accettare la nuova proprietà aggiungendo un argomento chiave.

```
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

File: `src/main/java/com/myorg/MultistackStack.java`

Estendere un tipo di oggetti di scena in Java è più complicato di quanto vogliamo davvero approfondire. Invece, scrivi il costruttore dello stack per accettare un parametro booleano opzionale. Poiché `props` è un argomento facoltativo, scriveremo un costruttore aggiuntivo che ti permetta di saltarlo. L'impostazione predefinita sarà. `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#

File: `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
        }
    }
}
```

La nuova proprietà è facoltativa. Se `encryptBucket` (Python:`encrypt_bucket`) non è presente, il suo valore è `undefined` o l'equivalente locale. Per impostazione predefinita, il bucket non sarà crittografato.

Definisci la classe stack

Ora definiamo la nostra classe stack, usando la nostra nuova proprietà. Rendi il codice simile al seguente. Il codice che devi aggiungere o modificare è mostrato in grassetto.

TypeScript

File: `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

Archiviazione: `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

Archiviazione: `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

Archiviazione: `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#

Archiviazione: `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
                });
            }
        }
    }
}
```

Crea due istanze di stack

Ora aggiungeremo il codice per creare un'istanza di due stack separati. Come in precedenza, le righe di codice mostrate in grassetto sono quelle che devi aggiungere. Eliminare la `MultistackStack` definizione esistente.

TypeScript

File: `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

Archiviazione: `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

Archiviazione: `./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

Archiviazione: `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#

File: 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();
        }
    }
}
```

Questo codice utilizza la nuova proprietà `encryptBucket` (Python:`encrypt_bucket`) sulla `MultistackStack` classe per istanziare quanto segue:

- Uno stack con un bucket Amazon S3 crittografato nella regione. `us-east-1` AWS
- Uno stack con un bucket Amazon S3 non crittografato nella regione. `us-west-1` AWS

Sintetizza e distribuisci lo stack

Ora puoi distribuire gli stack dall'app. Innanzitutto, sintetizza un AWS CloudFormation modello per `MyEastCdkStack` —the stack in. `us-east-1` Questo è lo stack con il bucket S3 crittografato.

```
$ cdk synth MyEastCdkStack
```

Per distribuire questo stack sul tuo AWS account, esegui uno dei seguenti comandi. Il primo comando utilizza il AWS profilo predefinito per ottenere le credenziali per distribuire lo stack. Il secondo utilizza un profilo specificato dall'utente. Per *`PROFILE_NAME`*, sostituisci il nome di un AWS CLI profilo che contiene le credenziali appropriate per la distribuzione nella regione. `us-east-1` AWS

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

Eliminazione

Per evitare addebiti per le risorse che hai distribuito, distruggi lo stack utilizzando il seguente comando.

```
cdk destroy MyEastCdkStack
```

L'operazione di distruzione fallisce se c'è qualcosa memorizzato nel bucket dello stack. Non dovrebbe esserci se hai solo seguito le istruzioni in questo argomento. Ma se hai messo qualcosa nel bucket, devi eliminarne il contenuto prima di distruggere lo stack. (Non eliminare il bucket stesso.) Usa AWS Management Console o AWS CLI per eliminare il contenuto del bucket.

Esempi

Questo argomento contiene i seguenti esempi:

- Crea un'applicazione Hello World senza serverCrea un'applicazione serverless utilizzando Lambda, API Gateway e Amazon S3.
- Creazione di un servizio AWS Fargate utilizzando AWS CDKCrea un servizio Amazon ECS Fargate a partire da un'immagine. DockerHub

Creazione di un servizio AWS Fargate utilizzando AWS CDK

Questo esempio illustra come creare un servizio AWS Fargate in esecuzione su un cluster Amazon Elastic Container Service (Amazon ECS) gestito da un Application Load Balancer connesso a Internet da un'immagine su Amazon ECR.

Amazon ECS è un servizio di gestione dei container rapido e altamente scalabile che consente di eseguire, arrestare e gestire container Docker in un cluster. Puoi ospitare il tuo cluster su un'infrastruttura serverless gestita da Amazon ECS avviando i tuoi servizi o attività utilizzando il tipo di lancio Fargate. Per un maggiore controllo, puoi ospitare le tue attività su un cluster di istanze Amazon Elastic Compute Cloud (Amazon EC2) che gestisci utilizzando il tipo di avvio di Amazon EC2.

Questo tutorial mostra come avviare alcuni servizi utilizzando il tipo di lancio Fargate. Se hai utilizzato il AWS Management Console per creare un servizio Fargate, sai che ci sono molti passaggi da seguire per eseguire tale operazione. AWS contiene diversi tutorial e argomenti di documentazione che illustrano la creazione di un servizio Fargate, tra cui:

- Come distribuire contenitori Docker - AWS
- Configurazione con Amazon ECS
- Guida introduttiva ad Amazon ECS con Fargate

Questo esempio crea un servizio Fargate simile nel AWS CDK codice.

Il costrutto Amazon ECS utilizzato in questo tutorial ti aiuta a utilizzare AWS i servizi offrendo i seguenti vantaggi:

- Configura automaticamente un sistema di bilanciamento del carico.

- Apre automaticamente un gruppo di sicurezza per i sistemi di bilanciamento del carico. Ciò consente ai sistemi di bilanciamento del carico di comunicare con le istanze senza che l'utente crei esplicitamente un gruppo di sicurezza.
- Ordina automaticamente la dipendenza tra il servizio e il sistema di bilanciamento del carico collegato a un gruppo target, applicando l'ordine corretto di creazione del AWS CDK listener prima della creazione di un'istanza.
- Configura automaticamente i dati degli utenti su gruppi di ridimensionamento automatico. In questo modo viene creata la configurazione corretta per associare un cluster alle AMI.
- Convalida anticipatamente le combinazioni di parametri. In questo modo i AWS CloudFormation problemi vengono scoperti prima, risparmiando così tempo per l'implementazione. Ad esempio, a seconda dell'attività, è facile configurare erroneamente le impostazioni della memoria. In precedenza, non si verificava alcun errore finché non si distribuiva l'app. Ma ora AWS CDK possono rilevare una configurazione errata ed emettere un errore durante la sintesi dell'app.
- Aggiunge automaticamente le autorizzazioni per Amazon Elastic Container Registry (Amazon ECR) se utilizzi un'immagine da Amazon ECR.
- Ridimensiona automaticamente. AWS CDK Fornisce un metodo che consente di scalare automaticamente le istanze quando si utilizza un cluster Amazon EC2. Ciò avviene automaticamente quando si utilizza un'istanza in un cluster Fargate.

  Inoltre, AWS CDK impedisce l'eliminazione di un'istanza quando il ridimensionamento automatico tenta di terminare un'istanza, ma un'attività è in esecuzione o è pianificata su quell'istanza.

  In precedenza, era necessario creare una funzione Lambda per avere questa funzionalità.
- Fornisce supporto per gli asset, in modo da poter distribuire un sorgente dalla macchina ad Amazon ECS in un unico passaggio. In precedenza, per utilizzare una fonte di applicazione era necessario eseguire diversi passaggi manuali, come il caricamento su Amazon ECR e la creazione di un'immagine Docker.

Vedi ECS per i dettagli.

> **Important**
>
> `ApplicationLoadBalancedFargateService`I costrutti che useremo includono numerosi AWS componenti, alcuni dei quali hanno costi non banali se lasciati disponibili nel tuo AWS

account, anche se non li usi. Assicurati di pulire (cdk destroy) dopo aver completato questo esempio.

Creazione della directory e inizializzazione di AWS CDK

Iniziamo creando una directory per contenere il AWS CDK codice e quindi creando un' AWS CDK app in quella directory.

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
```

Ora puoi importare il progetto Maven nel tuo IDE.

C#

```
mkdir MyEcsConstruct
```

```
cd MyEcsConstruct
cdk init --language csharp
```

Ora puoi aprirlo `src/MyEcsConstruct.sln` in Visual Studio.

Esegui l'app e conferma che crei uno stack vuoto.

```
cdk synth
```

Creare un servizio Fargate

Esistono due modi diversi per eseguire le attività relative ai container con Amazon ECS:

- Utilizza il tipo di `Fargate` avvio, in cui Amazon ECS gestisce per te le macchine fisiche su cui sono in esecuzione i container.
- Usa il tipo di `EC2` avvio, dove ti occupi della gestione, ad esempio specificando il ridimensionamento automatico.

Per questo esempio, creeremo un servizio Fargate in esecuzione su un cluster ECS gestito da un Application Load Balancer connesso a Internet.

Aggiungi le seguenti importazioni del modulo AWS Construct Library al file indicato.

TypeScript

File: `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

Archiviazione: `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

Archiviazione: `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

Archiviazione: `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#

Archiviazione: `src/MyEcsConstruct/MyEcsConstructStack.cs`

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

Sostituisci il commento alla fine del costruttore con il codice seguente.

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
                }
            );
```

Salvalo e assicurati che funzioni e crei uno stack.

```
cdk synth
```

La pila è composta da centinaia di righe, quindi non la mostreremo qui. Lo stack deve contenere un'istanza predefinita, una sottorete privata e una sottorete pubblica per le tre zone di disponibilità e un gruppo di sicurezza.

Distribuisci lo stack.

```
cdk deploy
```

AWS CloudFormation mostra informazioni sulle dozzine di passaggi necessari per distribuire l'app.

È così facile creare un servizio Amazon ECS basato su Fargate per eseguire un'immagine Docker.

Eliminazione

Per evitare AWS addebiti imprevisti, distruggi il tuo AWS CDK stack dopo aver terminato questo esercizio.

```
cdk destroy
```

AWS CDK esempi

Per altri esempi di AWS CDK stack e app nel tuo linguaggio di programmazione supportato preferito, consulta il repository AWS CDK Examples su. GitHub

AWS CDK attrezzi

Questa sezione contiene informazioni sugli AWS CDK strumenti elencati di seguito.

Argomenti

AWS CDK Toolkit (**cdk**comando)

Il AWS CDK Toolkit, il `cdk` comando CLI, è lo strumento principale per interagire con l'app. AWS CDK Esegue la tua app, interroga il modello di applicazione che hai definito e produce e distribuisce i modelli generati da. AWS CloudFormation AWS CDK Fornisce inoltre altre funzionalità utili per creare e lavorare con progetti. AWS CDK Questo argomento contiene informazioni sui casi d'uso comuni del CDK Toolkit.

Il AWS CDK Toolkit viene installato con Node Package Manager. Nella maggior parte dei casi, consigliamo di installarlo a livello globale.

```
npm install -g aws-cdk             # install latest version
npm install -g aws-cdk@X.YY.Z      # install specific version
```

> **Tip**
>
> Se lavori regolarmente con più versioni di AWS CDK, valuta la possibilità di installare una versione corrispondente del AWS CDK Toolkit nei singoli progetti CDK. Per fare ciò, omettete `-g` dal comando. `npm install` Quindi usa `npx aws-cdk` per invocarlo. Questo esegue la versione locale se ne esiste una, tornando a una versione globale in caso contrario.

Comandi del Toolkit

Tutti i comandi di CDK Toolkit iniziano con`cdk`, seguito da un sottocomando (`list`,, `synthesize`, `deploy`, ecc.). Alcuni sottocomandi hanno una versione più breve (`ls`,`synth`, ecc.)

che è equivalente. Le opzioni e gli argomenti seguono il sottocomando in qualsiasi ordine. I comandi disponibili sono riepilogati qui.

Comando	Funzione
`cdk list` (`ls`)	Elenca gli stack presenti nell'app
`cdk synthesize` (`synth`)	Sintetizza e stampa il CloudFormation modello per uno o più stack specificati
`cdk bootstrap`	Implementa lo stack di staging CDK Toolkit; vedi the section called “Bootstrapping (Processo di bootstrap)”
`cdk deploy`	Implementa uno o più stack specifici
`cdk destroy`	Distrugge uno o più stack specificati
`cdk diff`	Confronta lo stack specificato e le sue dipendenze con gli stack distribuiti o con un modello locale CloudFormation
`cdk import`	Utilizza le importazioni di CloudFormation risorse per inserire le risorse esistenti in uno stack gestito da CDK
`cdk metadata`	Visualizza i metadati relativi allo stack specifica to
`cdk init`	Crea un nuovo progetto CDK nella directory corrente a partire da un modello specificato
`cdk context`	Gestisce i valori di contesto memorizzati nella cache
`cdk docs` (`doc`)	Apre il CDK API Reference nel tuo browser
`cdk doctor`	Verifica la presenza di potenziali problemi nel progetto CDK

Per le opzioni disponibili per ogni comando, vederethe section called “Aiuto integrato”.

Specificare le opzioni e i relativi valori

Le opzioni della riga di comando iniziano con due trattini (). `--` Alcune opzioni utilizzate di frequente hanno sinonimi composti da una sola lettera che iniziano con un solo trattino (ad esempio, `--app` ha un sinonimo). `-a` L'ordine delle opzioni in un comando AWS CDK Toolkit non è importante.

Tutte le opzioni accettano un valore, che deve seguire il nome dell'opzione. Il valore può essere separato dal nome con uno spazio bianco o con un segno di uguale. = Le due opzioni seguenti sono equivalenti.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

Alcune opzioni sono bandiere (booleane). È possibile specificare `true` o `false` specificare il loro valore. Se non si fornisce un valore, il valore viene considerato uguale `atrue`. È inoltre possibile aggiungere al nome dell'opzione il prefisso «`no-`da `false` implicare».

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

Alcune opzioni, vale a dire`--context`,`--parameters`, e `--plugin` `--tags``--trust`, possono essere specificate più di una volta per specificare più valori. Queste sono note come scritte nella `[array]` guida di CDK Toolkit. Per esempio:

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

Aiuto integrato

Il AWS CDK Toolkit ha una guida integrata. È possibile visualizzare una guida generale sull'utilità e un elenco dei sottocomandi forniti eseguendo:

```
cdk --help
```

Per visualizzare l'aiuto relativo a un particolare sottocomando, ad esempio`deploy`, specificatelo prima del flag. `--help`

```
cdk deploy --help
```

Problema `cdk version` per visualizzare la versione del AWS CDK Toolkit. Fornisci queste informazioni quando richiedi assistenza.

Reportistica delle versioni

Per ottenere informazioni su come AWS CDK viene utilizzato, i costrutti utilizzati dalle AWS CDK applicazioni vengono raccolti e riportati utilizzando una risorsa identificata come`AWS::CDK::Metadata`. Questa risorsa viene aggiunta ai AWS CloudFormation modelli e può essere facilmente esaminata. Queste informazioni possono essere utilizzate anche AWS per identificare gli stack utilizzando un costrutto con problemi noti di sicurezza o affidabilità. Possono essere utilizzate anche per contattare gli utenti con informazioni importanti.

> **Note**
>
> Prima della versione 1.93.0, AWS CDK riportavano i nomi e le versioni dei moduli caricati durante la sintesi, anziché i costrutti utilizzati nello stack.

Per impostazione predefinita, AWS CDK riporta l'uso dei costrutti nei seguenti moduli NPM utilizzati nello stack:

- AWS CDK modulo principale
- AWS Moduli Construct Library
- AWS Modulo Solutions Constructs
- AWS Modulo Render Farm Deployment Kit

La `AWS::CDK::Metadata` risorsa ha un aspetto simile alla seguente.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
```

```
  Properties:
    Analytics:
 "v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/
XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz
+AqqLDt2Y3YMnLYWwAAAA="
```

La `Analytics` proprietà è un elenco con gzip, codificato in base64 e con prefisso, dei costrutti nello stack.

Per disattivare la segnalazione delle versioni, utilizzate uno dei seguenti metodi:

- Utilizzate il cdk comando con l'--no-version-reportingargomento per disattivare un singolo comando.

  ```
  cdk --no-version-reporting synth
  ```

  Ricorda che il AWS CDK Toolkit sintetizza nuovi modelli prima della distribuzione, quindi dovresti aggiungerli anche ai comandi. `--no-version-reporting` `cdk deploy`
- Impostato su false versionReporting in o. `./cdk.json` `~/.cdk.json` Questa opzione viene disattivata a meno che l'utente non effettui l'attivazione `--version-reporting` specificando un singolo comando.

  ```
  {
    "app": "...",
    "versionReporting": false
  }
  ```

Autenticazione con AWS

Esistono diversi modi in cui è possibile configurare l'accesso programmatico alle AWS risorse, a seconda dell'ambiente e dell' AWS accesso a disposizione.

Per scegliere il metodo di autenticazione e configurarlo per il CDK Toolkit, consulta Autenticazione e accesso nella Guida di riferimento agli AWS SDK e agli strumenti.

L'approccio consigliato per i nuovi utenti che si sviluppano localmente, ai quali non viene fornito un metodo di autenticazione dal datore di lavoro, è l'impostazione. AWS IAM Identity Center Questo metodo include l'installazione di AWS CLI per facilitare la configurazione e per accedere regolarmente al portale di AWS accesso. Se scegli questo metodo, l'ambiente dovrebbe contenere i

seguenti elementi dopo aver completato la procedura per l'autenticazione di IAM Identity Center nella Guida di riferimento agli AWS SDK e agli strumenti:

- Il AWS CLI, che viene utilizzato per avviare una sessione del portale di AWS accesso prima di eseguire l'applicazione.
- Un AWS`config`file condiviso con un `[default]` profilo con un set di valori di configurazione a cui è possibile fare riferimento da. AWS CDK Per trovare la posizione di questo file, consulta l'argomento relativo alla posizione dei file condivisi nella Guida di riferimento per SDK e strumenti AWS .
- Il `config` file condiviso imposta l'`region`impostazione. Questo imposta l'impostazione predefinita Regione AWS che CDK Toolkit AWS CDK e CDK Toolkit utilizzano per le AWS richieste.
- Il CDK Toolkit utilizza la configurazione del provider di token SSO del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il `sso_role_name` valore, che è un ruolo IAM connesso a un set di autorizzazioni IAM Identity Center, dovrebbe consentire l'accesso ai dati Servizi AWS utilizzati nell'applicazione.

  Il seguente `config` file di esempio mostra un profilo predefinito impostato con la configurazione del provider di token SSO. L'`sso_session`impostazione del profilo si riferisce alla `sso-session`sezione denominata. La `sso-session` sezione contiene le impostazioni per avviare una sessione del portale di AWS accesso.

```
[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
```

Avviare una sessione del portale di AWS accesso

Prima di accedere Servizi AWS, è necessaria una sessione attiva del portale di AWS accesso affinché CDK Toolkit utilizzi l'autenticazione IAM Identity Center per risolvere le credenziali. A seconda della durata della sessione configurata, l'accesso alla fine scadrà e CDK Toolkit riscontrerà

un errore di autenticazione. Esegui il seguente comando in AWS CLI per accedere al portale di accesso. AWS

```
aws sso login
```

Se la configurazione del provider di token SSO utilizza un profilo denominato anziché il profilo predefinito, il comando è`aws sso login --profile` *NAME*. Specificate inoltre questo profilo quando emettete cdk comandi utilizzando l'`--profile`opzione o la variabile di `AWS_PROFILE` ambiente.

Per verificare se hai già una sessione attiva, esegui il AWS CLI comando seguente.

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

La risposta a questo comando dovrebbe restituire l'account IAM Identity Center e il set di autorizzazioni configurati nel file `config` condiviso.

> **Note**
>
> Se hai già una sessione attiva del portale di AWS accesso ed eseguia`ws sso login`, non ti verrà richiesto di fornire credenziali.
> La procedura di accesso potrebbe richiedere all'utente di consentire l' AWS CLI accesso ai dati. Poiché AWS CLI è basato sull'SDK per Python, i messaggi di autorizzazione possono contenere variazioni del `botocore` nome.

Specificare la regione e altre configurazioni

Il CDK Toolkit deve conoscere la AWS regione in cui state distribuendo e come effettuare l'autenticazione. AWS Ciò è necessario per le operazioni di distribuzione e per recuperare i valori di contesto durante la sintesi. Insieme, l'account e la regione costituiscono l'ambiente.

La regione può essere specificata utilizzando variabili di ambiente o nei file di configurazione. Si tratta delle stesse variabili e degli stessi file utilizzati da altri AWS strumenti come AWS CLI i vari AWS SDK. Il CDK Toolkit cerca queste informazioni nell'ordine seguente.

- La variabile di `AWS_DEFAULT_REGION` ambiente.
- Un profilo denominato definito nel AWS `config` file standard e specificato utilizzando l'`--profile`opzione sui `cdk` comandi.

- La `[default]` sezione del AWS `config` file standard.

Oltre a specificare AWS l'autenticazione e una regione nella `[default]` sezione, puoi anche aggiungere una o più `[profile NAME]` sezioni, dove *NOME* è il nome del profilo. Per ulteriori informazioni sui profili denominati, consulta File di configurazione e credenziali condivisi nella Guida di riferimento agli AWS SDK e agli strumenti.

Il AWS `config` file standard si trova in `~/.aws/config` (macOS/Linux) o `%USERPROFILE%\.aws\config` (Windows). Per dettagli e posizioni alternative, consulta Posizione dei file di configurazione e credenziali condivisi nella Guida di riferimento agli SDK e agli strumenti AWS

L'ambiente specificato nell' AWS CDK app utilizzando la `env` proprietà dello stack viene utilizzato durante la sintesi. Viene utilizzato per generare un AWS CloudFormation modello specifico per l'ambiente e, durante la distribuzione, sostituisce l'account o la regione specificati con uno dei metodi precedenti. Per ulteriori informazioni, consulta the section called “Ambienti”.

> **Note**
>
> AWS CDK Utilizza le credenziali degli stessi file di origine di altri AWS strumenti e SDK, tra cui. AWS Command Line Interface Tuttavia, AWS CDK potrebbero comportarsi in modo leggermente diverso da questi strumenti. Usa la parte AWS SDK for JavaScript inferiore del cofano. Per i dettagli completi sulla configurazione delle credenziali per AWS SDK for JavaScript, vedere Impostazione delle credenziali.

Facoltativamente, puoi utilizzare l'opzione `--role-arn` (or`-r`) per specificare l'ARN di un ruolo IAM da utilizzare per la distribuzione. Questo ruolo deve essere assunto dall' AWS account utilizzato.

Specificare il comando app

Molte funzionalità del CDK Toolkit richiedono la sintesi di uno o più AWS CloudFormation modelli, il che a sua volta richiede l'esecuzione dell'applicazione. AWS CDK Supporta programmi scritti in una varietà di lingue. Pertanto, utilizza un'opzione di configurazione per specificare l'esatto comando necessario per eseguire l'app. Questa opzione può essere specificata in due modi.

Innanzitutto, e più comunemente, può essere specificata utilizzando la `app` chiave all'interno del file`cdk.json`. Si trova nella directory principale del AWS CDK progetto. Il CDK Toolkit fornisce un comando appropriato per la creazione di un nuovo progetto con. `cdk init` Ecco, ad esempio, il `cdk.json` brano tratto da un nuovo TypeScript progetto.

```
{
  "app": "npx ts-node bin/hello-cdk.ts"
}
```

Il CDK Toolkit cerca `cdk.json` nella directory di lavoro corrente quando tenta di eseguire l'app. Per questo motivo, potreste tenere aperta una shell nella directory principale del progetto per emettere i comandi di CDK Toolkit.

CDK Toolkit cerca anche la chiave dell'app `~/.cdk.json` (cioè nella home directory) se non riesce a trovarla. `./cdk.json` Aggiungere qui il comando app può essere utile se di solito lavori con codice CDK nella stessa lingua.

Se ti trovi in un'altra directory o desideri eseguire l'app utilizzando un comando diverso da quello in`cdk.json`, usa l'opzione `--app` (or`-a`) per specificarlo.

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

Durante la distribuzione, puoi anche specificare una directory contenente assembly cloud sintetizzati, ad esempio`cdk.out`, come valore di. --app Gli stack specificati vengono distribuiti da questa directory; l'app non viene sintetizzata.

Specificare gli stack

Molti comandi di CDK Toolkit (ad esempio`cdk deploy`) funzionano sugli stack definiti nell'app. Se l'app contiene solo uno stack, CDK Toolkit presume che intendiate quello se non specificate esplicitamente uno stack.

Altrimenti, devi specificare lo stack o gli stack con cui vuoi lavorare. È possibile farlo specificando singolarmente gli stack desiderati per ID sulla riga di comando. Ricorda che l'ID è il valore specificato dal secondo argomento quando crei un'istanza dello stack.

```
cdk synth PipelineStack LambdaStack
```

Potete anche usare i caratteri jolly per specificare gli ID che corrispondono a uno schema.

- ?corrisponde a qualsiasi carattere singolo
- *corrisponde a qualsiasi numero di caratteri (*da solo corrisponde a tutte le pile)

- **corrisponde a tutto ciò che è in una gerarchia

Puoi anche usare l'--allopzione per specificare tutti gli stack.

Se la tua app utilizza CDK Pipelines, CDK Toolkit interpreta i tuoi stack e le tue fasi come una gerarchia. Inoltre, l'--allopzione e la jolly corrispondono solo agli stack di primo livello. * Per abbinare tutte le pile, usa. ** Utilizzalo anche ** per indicare tutte le pile in una particolare gerarchia.

Quando usate i caratteri jolly, racchiudete il pattern tra virgolette o uscite dai caratteri jolly con. \ Se non lo fate, la vostra shell potrebbe cercare di espandere il pattern ai nomi dei file nella directory corrente. Nella migliore delle ipotesi, questo non farà quello che ti aspetti; nel peggiore dei casi, potresti distribuire stack che non avevi intenzione di fare. Questo non è strettamente necessario su Windows perché `cmd.exe` non espande i caratteri jolly, ma è comunque una buona pratica.

```
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**
>
> L'ordine in cui si specificano gli stack non è necessariamente l'ordine in cui verranno elaborati. Il AWS CDK Toolkit tiene conto delle dipendenze tra gli stack quando decide l'ordine in cui elaborarli. Ad esempio, supponiamo che uno stack utilizzi un valore prodotto da un altro (come l'ARN di una risorsa definita nel secondo stack). In questo caso, il secondo stack viene sintetizzato prima del primo a causa di questa dipendenza. È possibile aggiungere dipendenze tra gli stack manualmente utilizzando il metodo dello stack. `addDependency()`

Avvia il tuo ambiente AWS

L'implementazione degli stack con il CDK richiede il provisioning di risorse speciali dedicate AWS CDK . Il `cdk bootstrap` comando crea le risorse necessarie per te. È necessario eseguire il bootstrap solo se si distribuisce uno stack che richiede queste risorse dedicate. Per informazioni dettagliate, vedi the section called “Bootstrapping (Processo di bootstrap)”.

```
cdk bootstrap
```

Se emesso senza argomenti, come illustrato di seguito, il `cdk bootstrap` comando sintetizza l'app corrente e avvia gli ambienti in cui verranno distribuiti gli stack. Se l'app contiene stack indipendenti dall'ambiente, che non specificano esplicitamente un ambiente, l'account e la regione predefiniti vengono avviati oppure l'ambiente specificato viene utilizzato. `--profile`

All'esterno di un'app, è necessario specificare in modo esplicito l'ambiente da avviare. Puoi farlo anche per avviare un ambiente non specificato nella tua app o nel tuo profilo locale. AWS Le credenziali devono essere configurate (ad esempio in`~/.aws/credentials`) per l'account e la regione specificati. È possibile specificare un profilo che contenga le credenziali richieste.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.
cdk bootstrap 1111111111/us-east-1
cdk bootstrap --profile test 1111111111/us-east-1
```

Important

Ogni ambiente (combinazione account/regione) in cui viene distribuito tale stack deve essere avviato separatamente.

È possibile che vengano AWS addebitati costi per ciò che memorizza nelle risorse avviate. AWS CDK Inoltre, se si utilizza`-bootstrap-customer-key`, verrà creata una chiave AWS KMS, che comporta anche costi per ambiente.

Note

Le versioni precedenti del modello bootstrap creavano una chiave KMS per impostazione predefinita. Per evitare addebiti, riavvia il sistema utilizzando. `--no-bootstrap-customer-key`

Note

CDK Toolkit v2 non supporta il modello di bootstrap originale, soprannominato modello legacy, utilizzato di default con CDK v1.

> **Important**
>
> Il moderno modello di bootstrap concede in modo efficace le autorizzazioni implicite da a qualsiasi account presente nell'elenco. `--cloudformation-execution-policies` AWS `--trust` Per impostazione predefinita, questo estende le autorizzazioni di lettura e scrittura a qualsiasi risorsa nell'account avviato. Assicurati di configurare lo stack di bootstrap con politiche e account affidabili con cui ti senti a tuo agio.

Creare una nuova app

Per creare una nuova app, crea una cartella corrispondente, quindi, all'interno della directory, emetti`cdk init`.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

Le lingue supportate (*LANGUAGE*) sono:

Codice	Lingua
`typescript`	TypeScript
`javascript`	JavaScript
`python`	Python
`java`	Java
`csharp`	C#

TEMPLATE è un modello opzionale. Se il modello desiderato è app, l'impostazione predefinita, puoi ometterlo. I modelli disponibili sono:

Modello	Descrizione
app(impostazione predefinita)	Crea un' AWS CDK app vuota.

Modello	Descrizione
`sample-app`	Crea un' AWS CDK app con uno stack contenente una coda Amazon SQS e un argomento Amazon SNS.

I modelli utilizzano il nome della cartella del progetto per generare nomi per file e classi all'interno della nuova app.

Pile di elenchi

Per visualizzare un elenco degli ID degli stack presenti nell' AWS CDK applicazione, inserisci uno dei seguenti comandi equivalenti:

```
cdk list
cdk ls
```

Se l'applicazione contiene stack CDK Pipelines, CDK Toolkit visualizza i nomi degli stack come percorsi in base alla loro posizione nella gerarchia delle pipeline. (Ad esempio, e.) `PipelineStack` `PipelineStack/Prod` `PipelineStack/Prod/MyService`

Se l'app contiene molti stack, puoi specificare gli ID degli stack completi o parziali degli stack da elencare. Per ulteriori informazioni, consulta the section called “Specificare gli stack”.

Aggiungi il `--long` flag per visualizzare ulteriori informazioni sugli stack, inclusi i nomi degli stack e i relativi ambienti (AWS account e regione).

Sintetizzazione degli stack

Il `cdk synthesize` comando (quasi sempre abbreviato`synth`) sintetizza uno stack definito nell'app in un modello. CloudFormation

```
cdk synth         # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"     # all stacks in app
```

> **Note**
>
> Il CDK Toolkit esegue effettivamente l'app e sintetizza nuovi modelli prima della maggior parte delle operazioni (ad esempio durante la distribuzione o il confronto degli stack). Questi modelli sono memorizzati per impostazione predefinita nella directory. `cdk.out` Il `cdk synth` comando stampa semplicemente i modelli generati per uno o più stack specificati.

Vedi tutte `cdk synth --help` le opzioni disponibili. Alcune delle opzioni utilizzate più di frequente sono trattate nella sezione seguente.

Specificare i valori di contesto

Utilizzate l'`-c`opzione `--context` o per passare i valori di contesto di runtime all'app 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
```

Quando si distribuiscono più stack, i valori di contesto specificati vengono normalmente passati a tutti. Se lo desideri, puoi specificare valori diversi per ogni stack anteponendo il nome dello stack al valore di contesto.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

Specificare il formato di visualizzazione

Per impostazione predefinita, il modello sintetizzato viene visualizzato in formato YAML. Aggiungi invece il `--json` flag per visualizzarlo in formato JSON.

```
cdk synth --json MyStack
```

Specificare la directory di output

Aggiungete l'opzione `--output` (`-o`) per scrivere i modelli sintetizzati in una directory diversa da. `cdk.out`

```
cdk synth --output=~/templates
```

Distribuzione degli stack

Il `cdk deploy` sottocomando distribuisce uno o più stack specificati nell'account. AWS

```
cdk deploy        # if app contains only one stack
cdk deploy MyStack
cdk deploy Stack1 Stack2
cdk deploy "*"    # all stacks in app
```

> **Note**
>
> Il CDK Toolkit esegue l'app e sintetizza nuovi modelli prima di distribuire qualsiasi cosa. AWS CloudFormation Pertanto, la maggior parte delle opzioni della riga di comando che è possibile utilizzare `cdk synth` (ad esempio,`--context`) possono essere utilizzate anche con. `cdk deploy`

Vedi tutte `cdk deploy --help` le opzioni disponibili. Alcune delle opzioni più utili sono illustrate nella sezione seguente.

Saltare la sintesi

Il cdk deploy comando normalmente sintetizza gli stack dell'app prima della distribuzione per assicurarsi che la distribuzione rifletta la versione più recente dell'app. Se sai di non aver modificato il codice dall'ultima modificacdk synth, puoi eliminare la fase di sintesi ridondante durante la distribuzione. A tale scopo, specificate la `cdk.out` directory del progetto nell'opzione. --app

```
cdk deploy --app cdk.out StackOne StackTwo
```

Disabilitazione del rollback

AWS CloudFormation ha la capacità di annullare le modifiche in modo che le distribuzioni siano atomiche. Ciò significa che hanno successo o falliscono nel loro complesso. AWS CDK Eredita questa funzionalità perché sintetizza e distribuisce modelli. AWS CloudFormation

Il rollback assicura che le risorse siano sempre in uno stato coerente, il che è fondamentale per gli stack di produzione. Tuttavia, mentre state ancora sviluppando l'infrastruttura, alcuni guasti sono inevitabili e il ripristino delle implementazioni fallite può rallentare le vostre attività.

Per questo motivo, CDK Toolkit consente di disabilitare il rollback aggiungendoli al comando. `--no-rollback` `cdk deploy` Con questo flag, le distribuzioni non riuscite non vengono ripristinate. Le risorse distribuite prima della risorsa guasta rimangono invece valide e la distribuzione successiva inizia con la risorsa guasta. Dedicherai molto meno tempo ad aspettare le implementazioni e molto più tempo a sviluppare la tua infrastruttura.

Sostituzione a caldo

Usa il `--hotswap` flag con `cdk deploy` per cercare di aggiornare direttamente AWS le tue risorse invece di generare un set di AWS CloudFormation modifiche e distribuirlo. La distribuzione passa alla AWS CloudFormation distribuzione se l'hot swap non è possibile.

Attualmente l'hot swapping supporta funzioni Lambda, macchine a stati Step Functions e immagini di container Amazon ECS. Il `--hotswap` flag disabilita anche il rollback (cioè implica). `--no-rollback`

> **Important**
>
> L'hot-swapping non è consigliato per le implementazioni di produzione.

modalità Watch

La modalità watch (cdk deploy --watcho in breve) di CDK Toolkit monitora continuamente i file sorgente e le risorse dell'app CDK cdk watch per rilevare eventuali modifiche. Esegue immediatamente una distribuzione degli stack specificati quando viene rilevata una modifica.

Per impostazione predefinita, queste distribuzioni utilizzano il `--hotswap` flag, che accelera l'implementazione delle modifiche alle funzioni Lambda. Inoltre, se AWS CloudFormation hai modificato la configurazione dell'infrastruttura, ritorna alla distribuzione. Per eseguire `cdk watch` sempre AWS CloudFormation distribuzioni complete, aggiungi il `--no-hotswap` flag a. `cdk watch`

Tutte le modifiche apportate mentre `cdk watch` è già in esecuzione una distribuzione vengono combinate in un'unica distribuzione, che inizia non appena viene completata la distribuzione in corso.

La modalità di visualizzazione utilizza la `"watch"` chiave del progetto `cdk.json` per determinare quali file monitorare. Per impostazione predefinita, questi file sono i file e le risorse dell'applicazione, ma possono essere modificati modificando le `"exclude"` voci `"include"` and nella `"watch"` chiave. Il `cdk.json` file seguente mostra un esempio di queste voci.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

`cdk watch`esegue il `"build"` comando da `cdk.json` per creare l'app prima della sintesi. Se la distribuzione richiede comandi per creare o impacchettare il codice Lambda (o qualsiasi altra cosa non presente nell'app CDK), aggiungilo qui.

I caratteri jolly in stile Git, entrambi `*` e`**`, possono essere usati nei tasti and. `"watch"` `"build"` Ogni percorso viene interpretato in relazione alla directory principale di. `cdk.json` Il valore predefinito di `include` è`**/*`, ovvero tutti i file e le directory nella directory principale del progetto. `exclude`è facoltativo.

> **Important**
>
> La modalità Watch non è consigliata per le implementazioni di produzione.

Specificazione dei parametri AWS CloudFormation

Il AWS CDK Toolkit supporta la specificazione dei AWS CloudFormation parametri al momento della distribuzione. È possibile fornirli sulla riga di comando dopo il `--parameters` flag.

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

Per definire più parametri, utilizzate più `--parameters` flag.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
 downloadBucketName=DownBucket
```

Se state distribuendo più stack, potete specificare un valore diverso di ogni parametro per ogni stack. A tale scopo, aggiungete al nome del parametro il nome dello stack e i due punti. Altrimenti, lo stesso valore viene passato a tutti gli stack.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --
parameters YourStack:uploadBucketName=UpBucket
```

Per impostazione predefinita, AWS CDK conserva i valori dei parametri delle distribuzioni precedenti e li utilizza nelle distribuzioni successive se non vengono specificati in modo esplicito. Utilizzate il `--no-previous-parameters` flag per richiedere che tutti i parametri siano specificati.

Specificare il file di output

Se lo stack dichiara degli AWS CloudFormation output, questi vengono normalmente visualizzati sullo schermo al termine della distribuzione. Per scriverli in un file in formato JSON, usa il flag. `--outputs-file`

```
cdk deploy --outputs-file outputs.json MyStack
```

Modifiche relative alla sicurezza

Per proteggervi da modifiche involontarie che influiscono sul vostro livello di sicurezza, il AWS CDK Toolkit richiede di approvare le modifiche relative alla sicurezza prima di implementarle. Puoi specificare il livello di modifica che richiede l'approvazione:

```
cdk deploy --require-approval LEVEL
```

`LEVEL` può essere uno dei seguenti:

Termine	Significato
`never`	L'approvazione non è mai richiesta
`any-change`	Richiede l'approvazione di qualsiasi IAM o security-group-related modifica
`broadening` (impostazione predefinita)	Richiede l'approvazione quando vengono aggiunte le istruzioni IAM o le regole del

Termine	Significato
	traffico; le rimozioni non richiedono l'approva zione

L'impostazione può essere configurata anche nel `cdk.json` file.

```
{
  "app": "...",
  "requireApproval": "never"
}
```

Confronto degli stack

Il `cdk diff` comando confronta la versione corrente di uno stack (e le relative dipendenze) definita nell'app con le versioni già distribuite o con un AWS CloudFormation modello salvato e visualizza un elenco di modifiche.

```
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
```

Per confrontare gli stack dell'app con la distribuzione esistente:

```
cdk diff MyStack
```

Per confrontare gli stack della tua app con un modello salvato CloudFormation :

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

Importazione di risorse esistenti in uno stack

Puoi usare il `cdk import` comando per affidare la gestione delle CloudFormation risorse a un determinato AWS CDK stack. Questo è utile se si sta migrando o si stanno AWS CDK spostando risorse tra gli stack o se si sta modificando il relativo ID logico. `cdk import` Utilizza CloudFormation le importazioni di risorse. Consulta l'elenco delle risorse che possono essere importate qui.

Per importare una risorsa esistente in uno AWS CDK stack, procedi nel seguente modo:

- Assicurati che la risorsa non sia attualmente gestita da nessun altro CloudFormation stack. In caso affermativo, imposta innanzitutto la politica di rimozione sullo stack `RemovalPolicy.RETAIN` in cui si trova attualmente la risorsa ed esegui una distribuzione. Quindi, rimuovi la risorsa dallo stack ed esegui un'altra distribuzione. Questo processo assicurerà che la risorsa non sia più gestita da CloudFormation ma non la elimini.
- Esegui un comando `cdk diff` per assicurarti che non vi siano modifiche in sospeso allo AWS CDK stack in cui desideri importare le risorse. Le uniche modifiche consentite in un'operazione di «importazione» sono l'aggiunta di nuove risorse che si desidera importare.
- Aggiungi costrutti per le risorse che desideri importare nel tuo stack. Ad esempio, se desideri importare un bucket Amazon S3, aggiungi qualcosa come. `new s3.Bucket(this, 'ImportedS3Bucket', {});` Non apportate modifiche a nessun'altra risorsa.

  È inoltre necessario assicurarsi di modellare esattamente lo stato attuale della risorsa nella definizione. Per l'esempio del bucket, assicurati di includere AWS KMS chiavi, politiche del ciclo di vita e qualsiasi altra cosa pertinente al bucket. In caso contrario, le successive operazioni di aggiornamento potrebbero non funzionare come previsto.

Puoi scegliere se includere o meno il nome fisico del bucket. Di solito consigliamo di non includere i nomi delle risorse nelle definizioni AWS CDK delle risorse in modo da semplificare la distribuzione delle risorse più volte.

- Esegui `cdk import` *`STACKNAME`*.
- Se i nomi delle risorse non sono presenti nel modello, la CLI richiederà di inserire i nomi effettivi delle risorse che stai importando. Dopodiché, inizia l'importazione.
- Quando viene `cdk import` segnalato un esito positivo, la risorsa viene ora gestita da AWS CDK e CloudFormation. Qualsiasi modifica successiva apportata alle proprietà della risorsa nell' AWS CDK app, la configurazione del costrutto verrà applicata alla distribuzione successiva.
- Per confermare che la definizione della risorsa nell' AWS CDK app corrisponda allo stato corrente della risorsa, è possibile avviare un'operazione di rilevamento della CloudFormation deriva.

Questa funzionalità attualmente non supporta l'importazione di risorse in pile annidate.

Configurazione () **`cdk.json`**

I valori predefiniti per molti flag della riga di comando di CDK Toolkit possono essere memorizzati nel `cdk.json` file di un progetto o nel `.cdk.json` file nella directory utente. Di seguito è riportato un riferimento alfabetico alle impostazioni di configurazione supportate.

Chiave	Note	Opzione CDK Toolkit
`app`	Il comando che esegue l'applicazione CDK.	--app
`assetMetadata`	Se`false`, CDK non aggiunge metadati alle risorse che utilizzano risorse.	--no-asset-metadata
`bootstrapKmsKeyId`	Sostituisce l'ID della AWS KMS chiave utilizzata per crittografare il bucket di distribuzione di Amazon S3.	--bootstrap-kms-key-id
`build`	Il comando che compila o crea l'applicazione CDK prima della	--build

Chiave	Note	Opzione CDK Toolkit
	sintesi. Non consentito in. `~/.cdk.json`	
`browser`	Il comando per l'avvio di un browser Web per il `cdk docs` sottocomando.	--browser
`context`	Per informazioni, consulta the section called “Context”. I valori di contesto in un file di configurazione non verranno cancellati da. cdk context --clear (Il CDK Toolkit inserisce i valori di contesto memorizzati nella cache). `cdk.context.json`	--context
`debug`	Se `true`, CDK Toolkit emette informazioni più dettagliate utili per il debug.	--debug
`language`	Il linguaggio da usare per inizializzare nuovi progetti.	--language
`lookups`	Se `false`, non sono consentite ricerche contestuali. La sintesi avrà esito negativo se è necessario eseguire ricerche di contesto.	--no-lookups
`notices`	If `false`, sopprime la visualizzazione dei messaggi relativi a vulnerabilità di sicurezza, regressioni e versioni non supportate.	--no-notices

Chiave	Note	Opzione CDK Toolkit
`output`	Il nome della directory in cui verrà emesso l'assembly cloud sintetizzato (impostazione predefinita). `"cdk.out"`	--output
`outputsFile`	Il file in cui verranno scritti AWS CloudFormation gli output degli stack distribuiti (in formato JSON).	--outputs-file
`pathMetadata`	Se`false`, i metadati del percorso CDK non vengono aggiunti ai modelli sintetizzati.	--no-path-metadata
`plugin`	Array JSON che specifica i nomi dei pacchetti o i percorsi locali dei pacchetti che estendono il CDK	--plugin
`profile`	Nome del AWS profilo predefinito utilizzato per specificare la regione e le credenziali dell'account.	--profile
`progress`	Se impostato su`"events"`, CDK Toolkit visualizza tutti AWS CloudFormation gli eventi durante la distribuz ione, anziché una barra di avanzamento.	--progress

Chiave	Note	Opzione CDK Toolkit
`requireApproval`	Livello di approvazione predefinito per le modifiche alla sicurezza. Per informazioni, consultare the section called “Modifiche relative alla sicurezza”.	--require-approval
`rollback`	Se `false`, le distribuzioni non riuscite non vengono ripristinate.	--no-rollback
`staging`	Se `false`, le risorse non vengono copiate nella directory di output (utilizzate per il debug locale dei file sorgente con). AWS SAM	--no-staging
`tags`	Oggetto JSON contenente tag (coppie chiave-valore) per lo stack.	--tags
`toolkitBucketName`	Il nome del bucket Amazon S3 utilizzato per distribuire risorse come funzioni Lambda e immagini dei container (vedi. the section called “Avvia il tuo ambiente AWS”	--toolkit-bucket-name
`toolkitStackName`	Il nome dello stack di bootstrap (vedi. the section called “Avvia il tuo ambiente AWS”	--toolkit-stack-name
`versionReporting`	Se `false`, disattiva la segnalazione delle versioni.	--no-version-reporting

Chiave	Note	Opzione CDK Toolkit
`watch`	Oggetti JSON contenenti `"include"` `"exclude"` chiavi che indicano quali file devono (o non devono) attivare una ricostruzione del progetto quando vengono modificati. Per informazioni, consulta the section called “modalità Watch”.	--watch

cdk migrateriferimento al comando

Riferimento per il AWS Cloud Development Kit (AWS CDK) comando Command Line Interface (CLI). `cdk migrate` Per ulteriori informazioni sull'utilizzo`cdk migrate`, vedereEsegui la migrazione delle risorse e AWS CloudFormation dei modelli esistenti su AWS CDK.

Il `cdk migrate` comando migra le AWS risorse distribuite, gli AWS CloudFormation stack e i modelli locali AWS CloudFormation in. AWS CDK

Argomenti

- Utilizzo
- Opzioni

Utilizzo

```
$ cdk migrate <options>
```

Opzioni

Opzioni richieste

`--stack-name` *`STRING`*

Il nome dello AWS CloudFormation stack che verrà creato all'interno dell'app CDK dopo la migrazione.

Campo obbligatorio: sì

Opzioni condizionali

`--from-path` *`PATH`*

Il percorso del AWS CloudFormation modello da migrare. Fornisci questa opzione per specificare un modello locale.

Obbligatorio: condizionale. Obbligatorio in caso di migrazione da un AWS CloudFormation modello locale.

`--from-scan` *`STRING`*

Quando si migrano le risorse distribuite da un AWS ambiente, utilizzate questa opzione per specificare se avviare una nuova scansione o se AWS CDK CLI utilizzare l'ultima scansione riuscita.

Obbligatorio: condizionale. Richiesto per la migrazione dalle risorse distribuite. AWS

Valori accettati:, `most-recent` `new`

`--from-stack`

Fornisci questa opzione per migrare da uno stack distribuito AWS CloudFormation . Utilizzare `--stack-name` per specificare il nome dello stack distribuito. AWS CloudFormation

Obbligatorio: condizionale. Obbligatorio in caso di migrazione da uno stack distribuito. AWS CloudFormation

Opzioni opzionali

`--account` *`STRING`*

L'account da cui recuperare il modello dello AWS CloudFormation stack.

Required: No

Predefinito: AWS CDK CLI ottiene le informazioni sull'account da fonti predefinite.

`--compress`

Fornite questa opzione per comprimere il progetto CDK generato in un file. ZIP

Required: No

`--filter` *ARRAY*

Da utilizzare per la migrazione delle risorse distribuite da un account e. AWS Regione AWS Questa opzione specifica un filtro per determinare quali risorse distribuite migrare.

Questa opzione accetta una matrice di coppie chiave-valore, dove key rappresenta il tipo di filtro e value rappresenta il valore da filtrare.

Sono accettate le seguenti chiavi:

- `resource-identifier`— Un identificatore per la risorsa. Il valore può essere l'ID logico o fisico della risorsa. Ad esempio, `resource-identifier="ClusterName"`.
- `resource-type-prefix`— Il prefisso del tipo di AWS CloudFormation risorsa. Ad esempio, specifica di `resource-type-prefix="AWS::DynamoDB::"` filtrare tutte le risorse Amazon DynamoDB.
- `tag-key`— La chiave di un tag di risorsa. Ad esempio, `tag-key="myTagKey"`.
- `tag-value`— Il valore di un tag di risorsa. Ad esempio, `tag-value="myTagValue"`.

Fornisci più coppie chiave-valore per la logica `AND` condizionale. L'esempio seguente filtra per qualsiasi risorsa DynamoDB `myTagKey` etichettata come chiave tag:. `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

Fornisci l'`--filter`opzione più volte in un unico comando per la logica `OR` condizionale. L'esempio seguente filtra per qualsiasi risorsa che sia una risorsa DynamoDB o contrassegnata come chiave tag`myTagKey`:. `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

Required: No

`--language` *STRING*

Il linguaggio di programmazione da utilizzare per il progetto CDK creato durante la migrazione.

Required: No

Valori accettati:`typescript`,`python`,, `javacsharp`,`go`.

Default: `typescript`

`--output-path` *PATH*

Il percorso di output per il progetto CDK migrato.

Required: No

Impostazione predefinita: per impostazione predefinita, AWS CDK CLI utilizzerà la directory di lavoro corrente.

`--region` *STRING*

Regione AWS Da cui recuperare il modello dello AWS CloudFormation stack.

Required: No

Predefinito: AWS CDK CLI ottiene Regione AWS informazioni da fonti predefinite.

AWS Toolkit for Visual Studio Code

AWS Toolkit for Visual Studio Code è un plug-in open source per Visual Studio Code che semplifica la creazione, il debug e la distribuzione di applicazioni. AWS Il toolkit offre un'esperienza integrata per lo sviluppo di applicazioni. AWS CDK Include la funzionalità AWS CDK Explorer per elencare i AWS CDK progetti e sfogliare i vari componenti dell'applicazione CDK. Installa il AWS Toolkit e scopri di più sull'uso di Explorer. AWS CDK

AWS SAM integrazione

the AWS CDK e the AWS Serverless Application Model (AWS SAM) possono funzionare insieme per consentirvi di creare e testare localmente applicazioni serverless definite nel CDK. Per informazioni complete, consulta AWS Cloud Development Kit (AWS CDK)la Guida per gli AWS SAM sviluppatori. Per installare la CLI SAM, vedere Installazione della AWS SAM CLI.

Costrutti di test

Con AWS CDK, la tua infrastruttura può essere testabile come qualsiasi altro codice che scrivi. L'approccio standard al test AWS CDK delle app utilizza il modulo assertions e i framework AWS CDK di test più diffusi come Jest for e/o Pytest for TypeScript JavaScript Python.

Esistono due categorie di test che puoi scrivere per le app. AWS CDK

- Le asserzioni dettagliate verificano aspetti specifici del AWS CloudFormation modello generato, ad esempio «questa risorsa ha questa proprietà con questo valore». Questi test possono rilevare regressioni. Sono utili anche quando si sviluppano nuove funzionalità utilizzando lo sviluppo basato sui test. (Puoi prima scrivere un test, poi farlo passare scrivendo un'implementazione corretta.) Le asserzioni dettagliate sono i test più utilizzati.
- I test snapshot testano il modello sintetizzato rispetto a un AWS CloudFormation modello di base precedentemente memorizzato. I test snapshot consentono di eseguire il refactoring liberamente, poiché si può essere certi che il codice refattorizzato funzioni esattamente allo stesso modo dell'originale. Se le modifiche sono state intenzionali, puoi accettare una nuova base per i test futuri. Tuttavia, gli aggiornamenti CDK possono anche causare la modifica dei modelli sintetizzati, quindi non potete affidarvi solo alle istantanee per assicurarvi che l'implementazione sia corretta.

> **Note**
>
> Le versioni complete delle TypeScript app Python e Java utilizzate come esempi in questo argomento sono disponibili su. GitHub

Nozioni di base

Per illustrare come scrivere questi test, creeremo uno stack che contiene una macchina a AWS Step Functions stati e una funzione. AWS Lambda La funzione Lambda è sottoscritta a un argomento di Amazon SNS e inoltra semplicemente il messaggio alla macchina a stati.

Innanzitutto, crea un progetto di applicazione CDK vuoto utilizzando CDK Toolkit e installando le librerie di cui avremo bisogno. I costrutti che useremo si trovano tutti nel pacchetto CDK principale, che è una dipendenza predefinita nei progetti creati con CDK Toolkit. Tuttavia, è necessario installare il framework di test.

TypeScript

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

Crea una directory per i tuoi test.

```
mkdir test
```

Modifica il progetto `package.json` per dire a NPM come eseguire Jest e per dire a Jest quali tipi di file raccogliere. Le modifiche necessarie sono le seguenti.

- Aggiungere una nuova `test` chiave alla `scripts` sezione
- Aggiungi Jest e i suoi tipi alla sezione `devDependencies`
- Aggiungi una nuova chiave `jest` di primo livello con una dichiarazione `moduleFileExtensions`

Queste modifiche sono illustrate nello schema seguente. Posiziona il nuovo testo dove indicato in`package.json`. I segnaposto «...» indicano le parti esistenti del file che non devono essere modificate.

```
{
  ...
  "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
```

Crea una cartella per i tuoi test.

```
mkdir test
```

Modifica il progetto `package.json` per dire a NPM come eseguire Jest e per dire a Jest quali tipi di file raccogliere. Le modifiche necessarie sono le seguenti.

- Aggiungere una nuova `test` chiave alla `scripts` sezione
- Aggiungi Jest alla sezione `devDependencies`
- Aggiungi una nuova chiave `jest` di primo livello con una dichiarazione `moduleFileExtensions`

Queste modifiche sono illustrate nello schema seguente. Posiziona il nuovo testo dove indicato in`package.json`. I segnaposto «...» indicano parti esistenti del file che non devono essere modificate.

```
{
  ...
  "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
```

Apri il progetto nel tuo IDE Java preferito. (In Eclipse, usa File > Importa > Progetti Maven esistenti.)

C#

```
mkdir state-machine && cd-state-machine
cdk init --language=csharp
```

Apri `src\StateMachine.sln` in Visual Studio.

Fai clic con il pulsante destro del mouse sulla soluzione in Solution Explorer e scegli Aggiungi > Nuovo progetto. Cerca MSTest C# e aggiungi un progetto di test MSTest per C#. (Il nome predefinito va bene.) `TestProject1`

Fate clic con il pulsante destro del mouse **`TestProject1`** e scegliete Aggiungi > Riferimento al `StateMachine` progetto, quindi aggiungete il progetto come riferimento.

Lo stack di esempio

Ecco lo stack che verrà testato in questo argomento. Come descritto in precedenza, contiene una funzione Lambda e una macchina a stati Step Functions e accetta uno o più argomenti di Amazon SNS. La funzione Lambda è sottoscritta agli argomenti di Amazon SNS e li inoltra alla macchina a stati.

Non devi fare nulla di speciale per rendere l'app testabile. In effetti, questo stack CDK non è diverso in alcun modo importante dagli altri stack di esempio in questa Guida.

TypeScript

Inserite il seguente codice in: `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

Inserisci il seguente codice in`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

Inserisci il seguente codice in`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);
            }

        }
    }
}
```

Modificheremo il punto di ingresso principale dell'app in modo da non creare effettivamente un'istanza del nostro stack. Non vogliamo distribuirlo accidentalmente. I nostri test creeranno un'app e un'istanza dello stack per i test. Questa è una tattica utile se combinata con lo sviluppo basato sui test: assicurati che lo stack superi tutti i test prima di abilitare l'implementazione.

TypeScript

In `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

In `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

In `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();
        }
    }
}
```

La funzione Lambda

Il nostro stack di esempio include una funzione Lambda che avvia la nostra macchina a stati. Dobbiamo fornire il codice sorgente per questa funzione in modo che il CDK possa raggrupparla e distribuirla come parte della creazione della risorsa della funzione Lambda.

- Crea la cartella `start-state-machine` nella directory principale dell'app.
- In questa cartella, crea almeno un file. Ad esempio, è possibile salvare il codice seguente in`start-state-machines/index.js`.

```
exports.handler = async function (event, context) {
 return 'hello world';
};
```

  Tuttavia, qualsiasi file funzionerà, poiché in realtà non distribuiremo lo stack.

Esecuzione di test.

A titolo di riferimento, ecco i comandi che usi per eseguire i test nella tua AWS CDK app. Questi sono gli stessi comandi che useresti per eseguire i test in qualsiasi progetto utilizzando lo stesso framework di test. Per le lingue che richiedono una fase di compilazione, includila per assicurarti che i test siano stati compilati.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

Crea la tua soluzione (F6) per scoprire i test, quindi esegui i test (Test > Esegui tutti i test). Per scegliere quali test eseguire, apri Test Explorer (Test > Test Explorer).

O:

```
dotnet test src
```

Asserzioni granulari

Il primo passo per testare uno stack con asserzioni granulari è sintetizzare lo stack, perché stiamo scrivendo asserzioni sulla base del modello generato. AWS CloudFormation

`StateMachineStackStack`Richiediamo che venga passato l'argomento Amazon SNS per essere inoltrato alla macchina a stati. Quindi, nel nostro test, creeremo uno stack separato per contenere l'argomento.

In genere, quando si scrive un'app CDK, è possibile creare sottoclassi `Stack` e istanziare l'argomento Amazon SNS nel costruttore dello stack. Nel nostro test, creiamo `Stack` direttamente un'istanza, quindi passiamo questo stack come ambito e lo colleghiamo allo stack. `Topic` Questo è funzionalmente equivalente e meno prolisso. Aiuta anche a far sì che gli stack utilizzati solo nei test «abbiano un aspetto diverso» dagli stack che intendi distribuire.

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
        }
    }
}
```

Ora possiamo affermare che la funzione Lambda e l'abbonamento Amazon SNS sono stati creati.

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);
```

Il nostro test funzionale Lambda afferma che due proprietà particolari della risorsa funzionale hanno valori specifici. Per impostazione predefinita, il `hasResourceProperties` metodo esegue una corrispondenza parziale sulle proprietà della risorsa come indicato nel modello sintetizzato CloudFormation . Questo test richiede che le proprietà fornite esistano e abbiano i valori specificati, ma la risorsa può avere anche altre proprietà, che non sono state testate.

La nostra affermazione di Amazon SNS afferma che il modello sintetizzato contiene un abbonamento, ma nulla sull'abbonamento stesso. Abbiamo incluso questa affermazione principalmente per illustrare come affermare il numero di risorse. La `Template` classe offre metodi più specifici per scrivere asserzioni rispetto alle `Mapping` sezioni `ResourcesOutputs`, e del modello. CloudFormation

Matchers

Il comportamento di abbinamento parziale predefinito di `hasResourceProperties` può essere modificato utilizzando i matcher della `Match`classe.

I matcher vanno da lenient (`Match.anyValue`) a strict (). `Match.objectEquals` Possono essere annidati per applicare diversi metodi di corrispondenza a diverse parti delle proprietà della risorsa. Utilizzando `Match.objectEquals` e `Match.anyValue` insieme, ad esempio, possiamo testare in modo più completo il ruolo IAM della macchina a stati, senza richiedere valori specifici per le proprietà che potrebbero cambiare.

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" }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }));
```

Molte CloudFormation risorse includono oggetti JSON serializzati rappresentati come stringhe. Il `Match.serializedJson()` matcher può essere utilizzato per abbinare le proprietà all'interno di questo JSON.

Ad esempio, le macchine a stati Step Functions vengono definite utilizzando una stringa in Amazon States Language basato su JSON. Lo useremo `Match.serializedJson()` per assicurarci che il nostro stato iniziale sia l'unico passaggio. Ancora una volta, useremo abbinatori annidati per applicare diversi tipi di corrispondenza a diverse parti dell'oggetto.

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

Catturare

Spesso è utile testare le proprietà per assicurarsi che seguano formati specifici o abbiano lo stesso valore di un'altra proprietà, senza bisogno di conoscerne i valori esatti in anticipo. Il `assertions` modulo offre questa funzionalità nella sua `Capture`classe.

Specificando un'`Capture`istanza al posto di un valore in`hasResourceProperties`, tale valore viene mantenuto nell'`Capture`oggetto. Il valore effettivo acquisito può essere recuperato utilizzando i `as` metodi dell'oggetto, tra cui, e `asNumber()` `asString()`. `asObject`, e sottoposto a test. Da utilizzare `Capture` con un matcher per specificare la posizione esatta del valore da acquisire all'interno delle proprietà della risorsa, incluse le proprietà JSON serializzate.

L'esempio seguente verifica che lo stato iniziale della nostra macchina a stati abbia un nome che inizia con. `Start` Verifica inoltre che questo stato sia presente nell'elenco degli stati della macchina.

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()));
```

Test istantanei

Nei test delle istantanee, si confronta l'intero modello sintetizzato con un CloudFormation modello di base precedentemente memorizzato (spesso chiamato «master»). A differenza delle asserzioni granulari, il test delle istantanee non è utile per rilevare le regressioni. Questo perché il test delle istantanee si applica all'intero modello e altre cose, oltre alle modifiche al codice, possono causare piccole (o) differenze nei risultati di sintesi. not-so-small Queste modifiche potrebbero non influire nemmeno sulla distribuzione, ma causeranno comunque il fallimento di un test di snapshot.

Ad esempio, è possibile aggiornare un costrutto CDK per incorporare una nuova best practice, che può causare modifiche alle risorse sintetizzate o al modo in cui sono organizzate. In alternativa, è possibile aggiornare il CDK Toolkit a una versione che riporti metadati aggiuntivi. Le modifiche ai valori di contesto possono influire anche sul modello sintetizzato.

Tuttavia, i test istantanei possono essere di grande aiuto nel refactoring, purché si mantengano costanti tutti gli altri fattori che potrebbero influenzare il modello sintetizzato. Saprai immediatamente se una modifica apportata ha modificato involontariamente il modello. Se la modifica è intenzionale, è sufficiente accettare il nuovo modello come riferimento.

Ad esempio, se abbiamo questo `DeadLetterQueue` costrutto:

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()
            });
        }
    }
}
```

Possiamo testarlo in questo modo:

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());
        }
    }
}
```

Suggerimenti per i test

Ricorda che i tuoi test dureranno tanto quanto il codice che testano e verranno letti e modificati con la stessa frequenza. Pertanto, vale la pena dedicare un momento a considerare il modo migliore per scriverli.

Non copiate e incollate righe di configurazione o asserzioni comuni. Invece, rifattorizza questa logica in dispositivi o funzioni di supporto. Usa nomi validi che riflettano ciò che ogni test effettivamente verifica.

Non cercare di fare troppe cose in un solo test. Preferibilmente, un test dovrebbe testare uno e un solo comportamento. Se si interrompe accidentalmente questo comportamento, dovrebbe fallire esattamente un test e il nome del test dovrebbe indicare cosa non è riuscito. Tuttavia, questo è più un ideale da perseguire; a volte inevitabilmente (o inavvertitamente) si scrivono test che testano più di un comportamento. I test istantanei, per i motivi che abbiamo già descritto, sono particolarmente inclini a questo problema, quindi usateli con parsimonia.

Sicurezza per AWS Cloud Development Kit (AWS CDK)

La sicurezza cloud di Amazon Web Services (AWS) è la priorità più alta. In qualità di AWS cliente, puoi trarre vantaggio da un data center e da un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza. La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud.

Security of the Cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce tutti i servizi offerti nel AWS Cloud e della fornitura di servizi che è possibile utilizzare in modo sicuro. La nostra responsabilità in AWS materia di sicurezza è la massima priorità e l'efficacia della nostra sicurezza viene regolarmente testata e verificata da revisori di terze parti nell'ambito dei Programmi di AWS conformità.

Sicurezza nel cloud: la responsabilità dell'utente è determinata dal AWS servizio utilizzato e da altri fattori, tra cui la sensibilità dei dati, i requisiti dell'organizzazione e le leggi e i regolamenti applicabili.

AWS CDK Segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Argomenti

Gestione delle identità e degli accessi per AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) è un programma Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) a utilizzare le risorse. AWS IAM è uno Servizio AWS strumento che puoi utilizzare senza costi aggiuntivi.

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro che AWS svolgi.

Utente del servizio: se lo utilizzi Servizi AWS per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più AWS funzionalità per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore.

Amministratore del servizio: se sei responsabile delle AWS risorse della tua azienda, probabilmente hai pieno accesso alle AWS risorse. È tuo compito determinare a quali risorse Servizi AWS e a quali risorse devono accedere gli utenti del servizio. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM.

Amministratore IAM: un amministratore IAM potrebbe essere interessato a ottenere dei dettagli su come scrivere policy per gestire l'accesso a Servizi AWS.

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente.

Per accedere a AWS livello di codice, AWS fornisce i AWS CDK kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le richieste utilizzando le credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni

sul metodo consigliato per firmare personalmente le richieste, consulta la sezione Processo di firma con Signature Version 4 nella Riferimenti generali di AWS.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA) in AWS nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzarle per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente di IAM.

Identità federata

Come procedura consigliata, richiedi agli utenti umani, compresi gli utenti che richiedono l'accesso come amministratore, di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente dell'elenco utenti aziendale, di un provider di identità Web AWS Directory Service, della directory Identity Center o di qualsiasi utente che accede utilizzando le Servizi AWS credenziali fornite tramite un'origine di identità. Quando le identità federate accedono Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Puoi creare utenti e gruppi in IAM Identity Center oppure puoi connetterti e sincronizzarti con un set di utenti e gruppi nella tua fonte di identità per utilizzarli su tutte le tue applicazioni. Account AWS Per ulteriori informazioni sul Centro identità IAM, consulta Cos'è Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, per casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente di IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, è possibile avere un gruppo denominato Amministratori IAM e concedere a tale gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Quando creare un utente IAM (invece di un ruolo) nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Puoi assumere temporaneamente un ruolo IAM in AWS Management Console cambiando ruolo. Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente di IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Creazione di un ruolo per un provider di identità di terza parte nella Guida per l'utente di IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo

in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.
- Accesso a più servizi: alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è comune che tale servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
  - Ruolo di servizio: un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire azioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di IAM.
  - Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.
- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 e che AWS CLI effettuano richieste API. AWS Ciò è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un AWS ruolo a un'istanza EC2 e renderlo disponibile per tutte le sue applicazioni, crei un profilo di istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2 nella Guida per l'utente di IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta Quando creare un ruolo IAM (invece di un utente) nella Guida per l'utente di IAM.

Convalida della conformità per AWS Cloud Development Kit (AWS CDK)

AWS CDK Segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

La sicurezza e la conformità dei AWS servizi vengono valutate da revisori di terze parti nell'ambito di diversi programmi di AWS conformità. Questi includono SOC, PCI, FedRAMP, HIPAA e altri. AWS fornisce un elenco frequentemente aggiornato di servizi nell'ambito di specifici programmi di conformità nella pagina AWS Services in Scope by Compliance Program AWS .

I report di audit di terze parti possono essere scaricati utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in. AWS Artifact

Per ulteriori informazioni sui programmi di AWS conformità, vedere Programmi di AWS conformità.

La responsabilità di conformità quando si utilizza il AWS servizio AWS CDK per accedere a un servizio è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'organizzazione e dalle leggi e dai regolamenti applicabili. Se l'utilizzo di un AWS servizio è soggetto alla conformità a standard come HIPAA, PCI o FedRAMP, fornisce risorse per aiutare a: AWS

- Guide introduttive su sicurezza e conformità: guide all'implementazione che illustrano considerazioni sull'architettura e forniscono passaggi per implementare ambienti di base incentrati sulla sicurezza e sulla conformità. AWS
- AWS Risorse per la conformità: una raccolta di cartelle di lavoro e guide che potrebbero riguardare il settore e la località in cui operi.
- AWS Config Questo servizio valuta il livello di conformità delle configurazioni delle risorse con pratiche interne, linee guida e regolamenti industriali.
- AWS Security Hub— Una panoramica completa dello stato di sicurezza AWS che consente di verificare la conformità agli standard e alle best practice del settore della sicurezza.

Resilienza per AWS Cloud Development Kit (AWS CDK)

L'infrastruttura globale di Amazon Web Services (AWS) è costruita attorno a AWS regioni e zone di disponibilità.

AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti.

Con le zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture tradizionali a data center singolo o multiplo.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, consulta Global Infrastructure.AWS

AWS CDK Segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Sicurezza dell'infrastruttura per AWS Cloud Development Kit (AWS CDK)

AWS CDK Segue il modello di responsabilità condivisa attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la pagina della documentazione sulla sicurezza del AWS servizio e AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità.

Risoluzione dei AWS CDK problemi più comuni

In questo argomento viene descritto come risolvere i seguenti problemi con. AWS CDK

Dopo l'aggiornamento AWS CDK, il AWS CDK Toolkit (CLI) segnala una mancata corrispondenza con la Construct Library AWS

La versione del AWS CDK Toolkit (che fornisce il `cdk` comando) deve essere almeno uguale alla versione del modulo principale AWS di Construct Library,. `aws-cdk-lib` Il Toolkit è pensato per essere retrocompatibile. L'ultima versione 2.x del toolkit può essere utilizzata con qualsiasi versione 1.x o 2.x della libreria. Per questo motivo, ti consigliamo di installare questo componente a livello globale e di mantenerlo aggiornato.

```
npm update -g aws-cdk
```

Se hai bisogno di lavorare con più versioni del AWS CDK Toolkit, installa una versione specifica del toolkit localmente nella cartella del progetto.

Se state utilizzando TypeScript or JavaScript, la directory del progetto contiene già una copia locale con versione del CDK Toolkit.

Se state usando un'altra lingua, usate `npm` per installare il AWS CDK Toolkit, omettendo il `-g` flag e specificando la versione desiderata. Per esempio:

```
npm install aws-cdk@2.0
```

Per eseguire un AWS CDK Toolkit installato localmente, utilizzate il comando `npx aws-cdk` anziché only. `cdk` Per esempio:

```
npx aws-cdk deploy MyStack
```

`npx aws-cdk`esegue la versione locale del AWS CDK Toolkit, se ne esiste una. Torna alla versione globale quando un progetto non ha un'installazione locale. Potrebbe essere utile impostare un alias di shell per assicurarsi che `cdk` venga sempre invocato in questo modo.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

(torna alla lista)

Quando distribuisco il mio AWS CDK stack, ricevo un errore **`NoSuchBucket`**

Il tuo AWS ambiente non è stato avviato e quindi non dispone di un bucket Amazon S3 per contenere risorse durante la distribuzione. Puoi creare il bucket di staging e altre risorse necessarie con il seguente comando:

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

Per evitare di generare AWS addebiti imprevisti, AWS CDK non avvia automaticamente nessun ambiente. È necessario avviare in modo esplicito ogni ambiente in cui verrà distribuito.

Per impostazione predefinita, le risorse di bootstrap vengono create nella regione o nelle regioni utilizzate dagli stack dell'applicazione corrente. AWS CDK In alternativa, vengono create nella

regione specificata nel AWS profilo locale (impostata da`aws configure`), utilizzando l'account di quel profilo. È possibile specificare un account e una regione diversi nella riga di comando come segue. (È necessario specificare l'account e la regione se non ci si trova nella directory di un'app.)

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

Per ulteriori informazioni, consulta the section called “Bootstrapping (Processo di bootstrap)”.

(torna all'elenco)

Quando distribuisco il mio AWS CDK stack, ricevo un messaggio **`forbidden: null`**

Stai implementando uno stack che richiede risorse di bootstrap, ma stai utilizzando un ruolo o un account IAM che non dispone dell'autorizzazione per scriverlo. (Lo staging bucket viene utilizzato quando si distribuiscono stack che contengono risorse o che sintetizzano un modello più grande di 50.000.) AWS CloudFormation Utilizzate un account o un ruolo autorizzato a eseguire l'azione sul bucket indicato nel messaggio di `s3:*` errore.

(torna all'elenco)

Quando sintetizzo uno AWS CDK stack, ricevo il messaggio **`--app is required either in command-line, in cdk.json or in ~/.cdk.json`**

Questo messaggio di solito significa che non ti trovi nella directory principale del tuo AWS CDK progetto quando emetti. `cdk synth` Il file `cdk.json` in questa directory, creato dal `cdk init` comando, contiene la riga di comando necessaria per eseguire (e quindi sintetizzare) l'app AWS CDK . Per un' TypeScript app, ad esempio, l'impostazione predefinita è `cdk.json` simile a questa:

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

Ti consigliamo di impartire `cdk` i comandi solo nella directory principale del progetto, in modo che il AWS CDK toolkit possa `cdk.json` trovarli ed eseguire correttamente l'app.

Se per qualche motivo ciò non è pratico, il AWS CDK Toolkit cerca la riga di comando dell'app in altre due posizioni:

- `cdk.json`Nella tua home directory

- Sul `cdk synth` comando stesso usando l'`-a`opzione

Ad esempio, è possibile sintetizzare uno stack da un' TypeScript app nel modo seguente.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

(torna all'elenco)

Quando sintetizzo uno AWS CDK stack, ricevo un errore perché il AWS CloudFormation modello contiene troppe risorse

AWS CDK Genera e distribuisce modelli. AWS CloudFormation AWS CloudFormation ha un limite rigido al numero di risorse che uno stack può contenere. Con AWS CDK, puoi raggiungere questo limite più rapidamente di quanto ti aspetti.

> **Note**
>
> Il limite di AWS CloudFormation risorse è 500 al momento della stesura di questo documento. Vedi le AWS CloudFormation quote per l'attuale limite di risorse.

I costrutti di livello superiore e basati sugli intenti della AWS Construct Library forniscono automaticamente tutte le risorse ausiliarie necessarie per la registrazione, la gestione delle chiavi, l'autorizzazione e altri scopi. Ad esempio, la concessione dell'accesso a una risorsa a un'altra genera tutti gli oggetti IAM necessari per la comunicazione tra i servizi pertinenti.

In base alla nostra esperienza, l'uso nel mondo reale di costrutti basati sugli intenti comporta da 1 a 5 AWS CloudFormation risorse per costrutto, anche se questo dato può variare. Per le applicazioni serverless, sono tipicamente 5-8 risorse per endpoint API. AWS

I pattern, che rappresentano un livello di astrazione più elevato, consentono di definire ancora più AWS risorse con ancora meno codice. Il AWS CDK codice inthe section called “ECS”, ad esempio, genera più di 50 AWS CloudFormation risorse definendo solo tre costrutti!

Il superamento del limite di AWS CloudFormation risorse è un errore durante la AWS CloudFormation sintesi. AWS CDK Emette un avviso se lo stack supera l'80% del limite. Puoi utilizzare un limite diverso impostando la `maxResources` proprietà sullo stack o disabilitare la convalida impostandola su 0. `maxResources`

> **Tip**
>
> È possibile ottenere un conteggio esatto delle risorse nell'output sintetizzato utilizzando il seguente script di utilità. (Poiché ogni AWS CDK sviluppatore ha bisogno di Node.js, lo script è scritto in JavaScript esso.)

```
// 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");
```

Man mano che il numero di risorse dello stack si avvicina al limite, prendi in considerazione la possibilità di riprogettare per ridurre il numero di risorse contenute nello stack: ad esempio, combinando alcune funzioni Lambda o suddividendo lo stack in più stack. Il CDK supporta i riferimenti tra gli stack, quindi puoi separare le funzionalità dell'app in diversi stack nel modo che ritieni più opportuno.

> **Note**
>
> AWS CloudFormation gli esperti spesso suggeriscono l'uso di stack annidati come soluzione al limite delle risorse. AWS CDK Supporta questo approccio tramite il `NestedStack`costrutto.

(torna all'elenco)

Ho specificato tre (o più) zone di disponibilità per il mio gruppo Auto Scaling o VPC, ma sono state implementate solo in due

Per ottenere il numero di zone di disponibilità richieste, specifica l'account e la regione nella proprietà dello stack. `env` Se non li specificate entrambi, per impostazione predefinita lo AWS CDK stack sintetizza lo stack come indipendente dall'ambiente. È quindi possibile distribuire lo stack in una regione specifica utilizzando. AWS CloudFormation Poiché alcune regioni hanno solo due zone di disponibilità, un modello indipendente dall'ambiente non ne utilizza più di due.

> **Note**
>
> In passato, le regioni venivano lanciate occasionalmente con una sola zona di disponibilità. Gli AWS CDK stack indipendenti dall'ambiente non possono essere distribuiti in tali regioni. Al momento della stesura di questo documento, tuttavia, tutte le AWS regioni hanno almeno due AZ.

Puoi modificare questo comportamento sovrascrivendo la proprietà `availability_zones` (`availablilityZones`Python:) dello stack per specificare esplicitamente le zone che desideri utilizzare.

Per ulteriori informazioni su come specificare l'account e la regione di uno stack al momento della sintesi, pur mantenendo la flessibilità necessaria per la distribuzione in qualsiasi regione, consulta. the section called “Ambienti”

(torna all'elenco)

Il mio bucket S3, la tabella DynamoDB o un'altra risorsa non vengono eliminati quando emetto **`cdk destroy`**

Per impostazione predefinita, le risorse che possono contenere dati utente hanno una proprietà `removalPolicy` (Python:`removal_policy`) di `RETAIN` e la risorsa non viene eliminata quando lo stack viene distrutto. Invece, la risorsa è rimasta orfana dallo stack. È quindi necessario eliminare la risorsa manualmente dopo la distruzione dello stack. Finché non lo fai, la ridistribuzione dello stack non riesce. Questo perché il nome della nuova risorsa creata durante la distribuzione è in conflitto con il nome della risorsa orfana.

Se imposti la politica di rimozione di una risorsa su`DESTROY`, quella risorsa verrà eliminata quando lo stack viene distrutto.

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 non può eliminare un bucket Amazon S3 non vuoto. Se imposti la politica di rimozione di un bucket Amazon S3 su e questo contiene datiDESTROY, il tentativo di distruggere lo stack avrà esito negativo perché il bucket non può essere eliminato. Puoi fare in modo che gli oggetti nel bucket vengano AWS CDK eliminati prima di tentare di distruggerlo impostando la proprietà del bucket su. `autoDeleteObjects` `true`

(torna all'elenco)

Chiavi OpenPGP per e jsii AWS CDK

Questo argomento contiene le chiavi OpenPGP attuali e storiche per e jsii. AWS CDK

Chiavi attuali

Queste chiavi devono essere utilizzate per convalidare le versioni correnti di AWS CDK e jsii.

AWS CDK Chiave OpenPGP

ID chiave:	0x42B9CF2286CD987A
Type:	RSA
Dimensioni:	4096/4096
Creato:	2022-07-05
Scade:	04-07-2020
ID utente:	Kit di sviluppo cloud AWS < aws-cdk@amazon.com >
Impronta digitale chiave:	69B5 25DB A295 1D11 FA65 413B 42B9 CF22 86CD 987A

Seleziona l'icona «Copia» per copiare la seguente chiave 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-----
```

chiave jsii OpenPGP

ID chiave:	0x056C4E15DAE3D8D9
Type:	RSA
Dimensioni:	4096/4096
Creato:	2022-07-05
Scade:	04-07-2020
ID utente:	Team AWS JSII < aws-jsii@amazon.com >
Impronta digitale chiave:	E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

Seleziona l'icona «Copia» per copiare la seguente chiave 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-----
```

Chiavi storiche

Queste chiavi possono essere utilizzate per convalidare le versioni di AWS CDK e jsii prima del 05/07/2022.

> **Important**
>
> Le nuove chiavi vengono create prima della scadenza di quelle precedenti. Di conseguenza, in un dato momento, può essere valida più di una chiave. Le chiavi vengono utilizzate per firmare gli artefatti a partire dal giorno della loro creazione, quindi utilizzate la chiave emessa più di recente nel caso in cui la validità delle chiavi si sovrappone.

AWS CDK Chiave OpenPGP (2022-04-07)

> **Note**
>
> Questa chiave non è stata utilizzata per firmare artefatti dopo il 2022-07-05. AWS CDK

ID chiave:	0x015584281F44A3C3
Type:	RSA
Dimensioni:	4096/4096
Creato:	2022-04-07
Scade:	06/04/2020
ID utente:	Kit di sviluppo cloud AWS < aws-cdk@amazon.com >
Impronta digitale chiave:	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

Seleziona l'icona «Copia» per copiare la seguente chiave 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-----
```

chiave jsii OpenPGP (2022-04-07)

> **Note**
>
> Questa chiave non è stata utilizzata per firmare gli artefatti jsii dopo il 2022-07-05.

ID chiave:	0x985F5BC974B79356
Type:	RSA
Dimensioni:	4096/4096
Creato:	2022-04-07
Scade:	06/04/2020
ID utente:	Team AWS JSII < aws-jsii@amazon.com >
Impronta digitale chiave:	35A7 1785 8FA6 282D 5AC CD95 985F 5BC9 74B7 9356

Seleziona l'icona «Copia» per copiare la seguente chiave 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 Chiave OpenPGP (2018-06-19)

ID chiave:	0x0566A784E17F3870
Type:	RSA
Dimensioni:	4096/4096
Creato:	2018-06-19
Scade:	18/06/2022
ID utente:	Team CDK di AWS < aws-cdk@amazon.com >

Impronta digitale chiave:	E88B E3B6 F0B1 E350 9E36 4F96 0566 A784 E17F 3870

Seleziona l'icona «Copia» per copiare la seguente chiave 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-----
```

chiave jsii OpenPGP (2018-08-06)

ID chiave:	0x1C7ACE4CB2A1B93A
Type:	RSA

Dimensioni:	4096/4096
Creato:	2018-08-06
Scade:	05/08/2022
ID utente:	Team AWS JSII < aws-jsii@amazon.com >
Impronta digitale chiave:	85EF 6522 4CE2 1E8C 72 DB 28EC 1C7A CE4C B2A1 B93A

Seleziona l'icona «Copia» per copiare la seguente chiave 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 Cronologia della Guida per sviluppatori

Vedi Releases per informazioni sulle AWS CDK release. AWS CDK Viene aggiornato all'incirca una volta alla settimana. Le versioni di manutenzione possono essere rilasciate tra le versioni settimanali per risolvere problemi critici. Ogni versione include un AWS CDK Toolkit (CDK CLI) AWS , Construct Library e API Reference corrispondenti. Gli aggiornamenti di questa Guida in genere non si sincronizzano con le versioni. AWS CDK

> **Note**
>
> La tabella seguente rappresenta le tappe fondamentali della documentazione. Correggiamo gli errori e miglioriamo i contenuti su base continuativa.

Modifica	Descrizione	Data
Aggiungi la documentazione per la funzionalità CDK Migrate	Usa il AWS CDK CLI `cdk migrate` comando per migrare le AWS risorse distribuite, gli AWS CloudFormation stack distribuiti e i modelli locali verso. AWS CloudFormation AWS CDKPer ulteriori informazioni, consulta Migrate to. AWS CDK	2 febbraio 2024
Aggiornamenti delle best practice di IAM	Guida aggiornata per l'allineamento alle best practice IAM. Per ulteriori informazioni, consulta Best practice per la sicurezza in IAM.	23 marzo 2023
Documento `cdk.json`	Aggiungere la documentazione dei valori di `cdk.json` configurazione.	20 aprile 2022

Gestione delle dipendenze	Aggiungi un argomento sulla gestione delle dipendenze con. AWS CDK	7 aprile 2022
Rimuovi le doppie parentesi dagli esempi in Java	Sostituisci questo anti-pattern con Java 9 dappertutto. `Map.of`	9 marzo 2022
AWS CDK versione v2	È stata rilasciata la versione 2 della AWS CDK Developer Guide. Cronologia dei documenti per CDK v1.	4 dicembre 2021

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.