Guía para desarrolladores

AWS Cloud Development Kit (AWS CDK) v2

Version 2

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Cloud Development Kit (AWS CDK) v2: Guía para desarrolladores

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

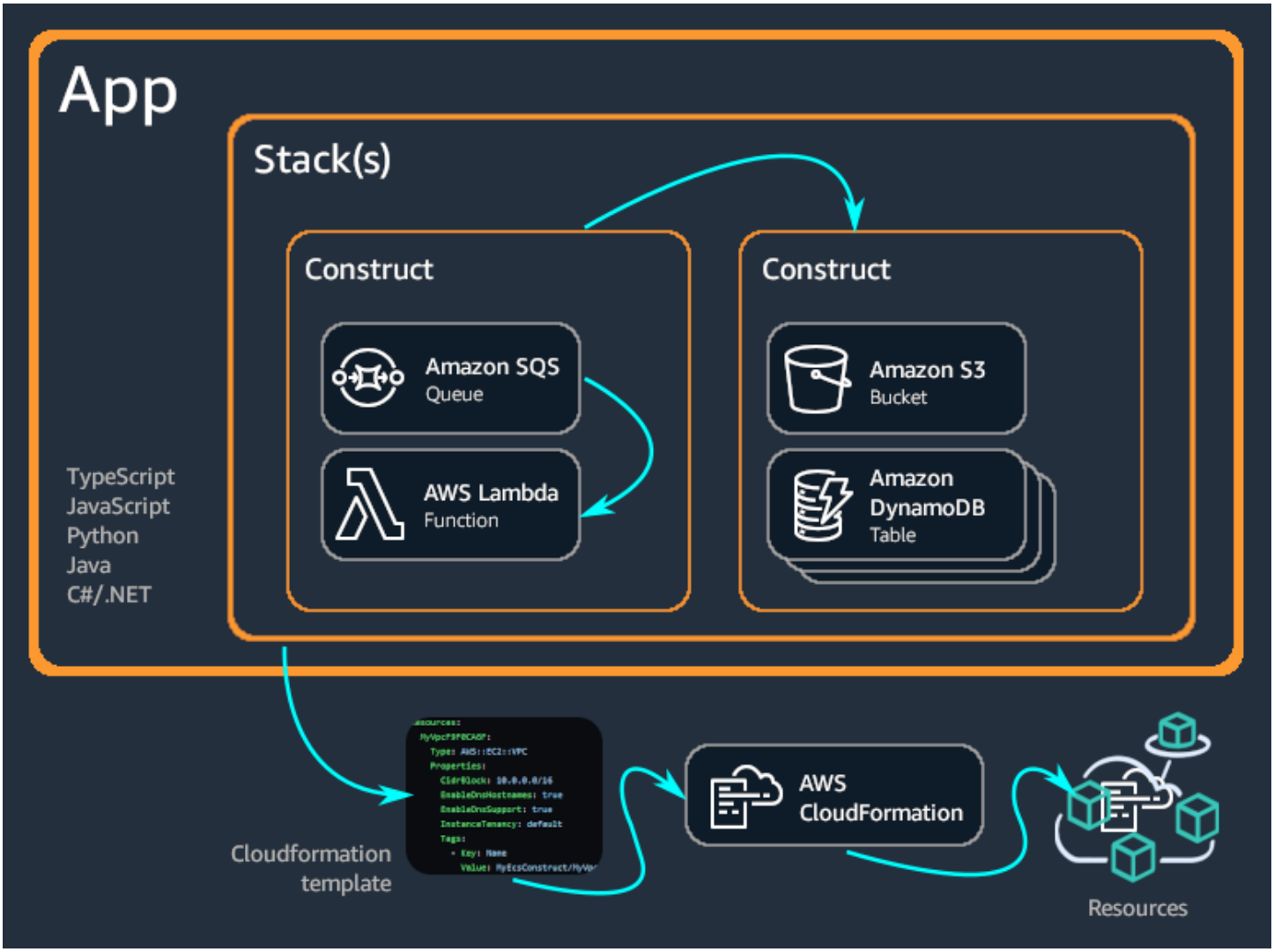
¿Qué es el AWS CDK?

AWS Cloud Development Kit (AWS CDK) Es un marco de desarrollo de software de código abierto para definir la infraestructura de la nube en el código y aprovisionarla mediante ella. AWS CloudFormation

AWS CDK Consta de dos partes principales:

- AWS CDK Biblioteca de construcción: una colección de piezas de código modulares y reutilizables preescritas, denominadas construcciones, que puede usar, modificar e integrar para desarrollar su infraestructura rápidamente. El objetivo de la biblioteca AWS CDK Construct es reducir la complejidad necesaria para definir e integrar los AWS servicios juntos a la hora de crear aplicaciones. AWS
- AWS CDK Kit de herramientas: una herramienta de línea de comandos para interactuar con las aplicaciones de CDK. Use el AWS CDK kit de herramientas para crear, administrar e implementar sus proyectos. AWS CDK

Los AWS CDK soportesTypeScript,JavaScript,Python, JavaC#/.Net, yGo. Puede usar cualquiera de estos lenguajes de programación compatibles para definir componentes de nube reutilizables conocidos como construcciones. Los puede agrupar en pilas y aplicaciones. Luego, despliega sus aplicaciones de CDK para AWS CloudFormation aprovisionar o actualizar sus recursos.



Temas

- Ventajas de la AWS CDK
- Ejemplo de AWS CDK
- AWS CDK features
- Siguientes pasos
- Más información

Ventajas de la AWS CDK

Úselo AWS CDK para desarrollar aplicaciones confiables, escalables y rentables en la nube con el considerable poder expresivo de un lenguaje de programación. Este enfoque ofrece muchos beneficios, entre los que se incluyen:

Desarrolle y administre su infraestructura como código (IaC)

Practique la infraestructura como código para crear, implementar y mantener la infraestructura de forma programática, descriptiva y declarativa. Con IaC, se trata la infraestructura de la misma manera que los desarrolladores tratan el código. Esto da como resultado un enfoque escalable y estructurado para administrar la infraestructura. Para obtener más información sobre la IaC, consulte La infraestructura como código en la Introducción a un DevOps documento AWS técnico.

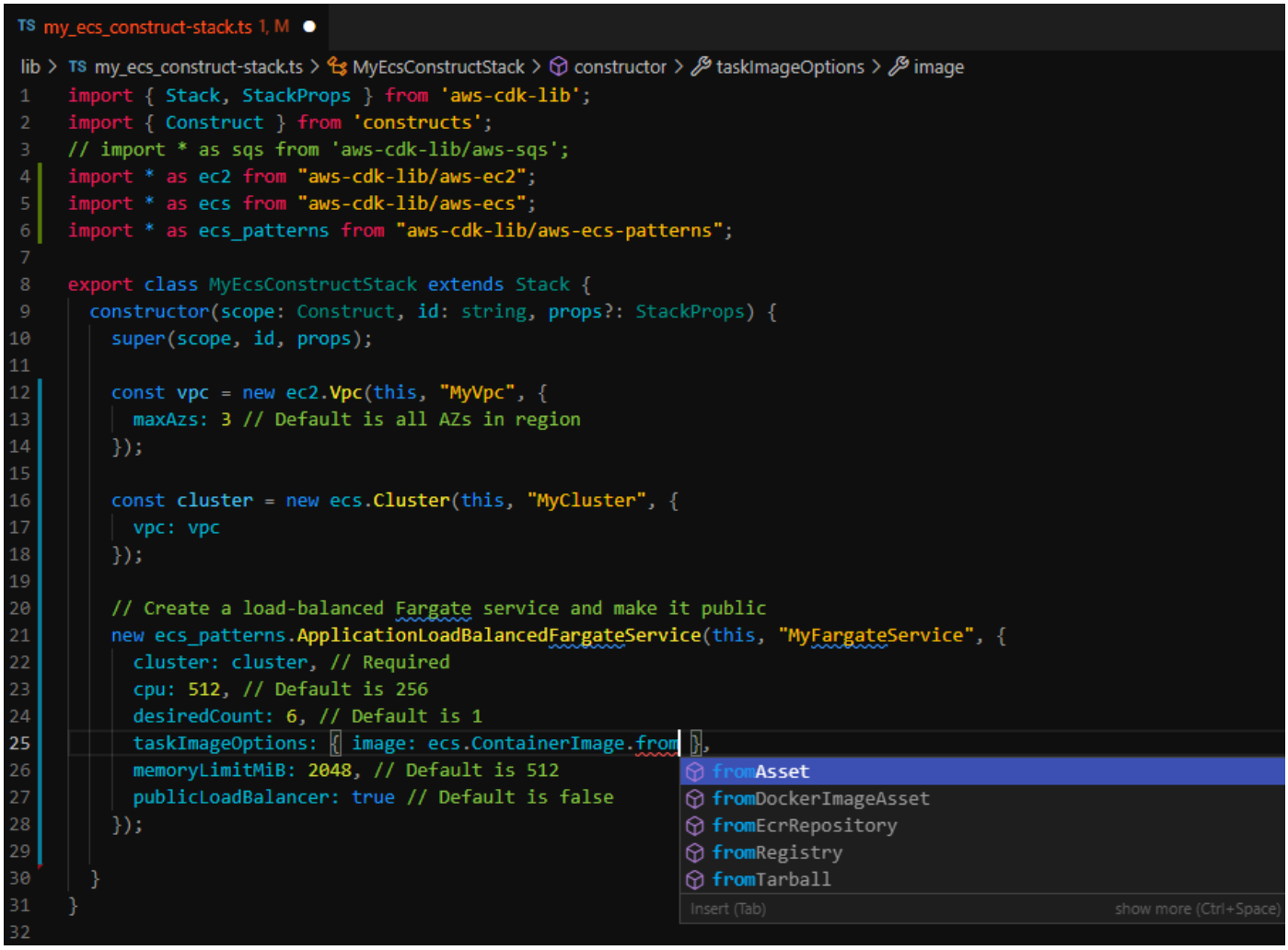
Con él AWS CDK, puede colocar la infraestructura, el código de la aplicación y la configuración en un solo lugar, lo que garantiza que cuenta con un sistema completo que se pueda implementar en la nube en cada etapa. Emplee las mejores prácticas de ingeniería de software, como revisiones de código, pruebas unitarias y control de código fuente, para que su infraestructura sea más sólida.

Defina su infraestructura de nube mediante lenguajes de programación de uso general

Con él AWS CDK, puede utilizar cualquiera de los siguientes lenguajes de programación para definir su infraestructura de nube:TypeScript,JavaScript, Python JavaC#/.Net, y. Go Elija el lenguaje que prefiera y utilice elementos de programación como parámetros, condicionales, bucles, composición y herencia para definir el resultado deseado de su infraestructura.

Utilice el mismo lenguaje de programación para definir la infraestructura y la lógica de la aplicación.

Disfrute de las ventajas de desarrollar una infraestructura en el IDE (entorno de desarrollo integrado) que prefiera, como el resaltado de la sintaxis y la finalización inteligente del código.



Implemente la infraestructura mediante AWS CloudFormation

AWS CDK se integra con AWS CloudFormation para implementar y aprovisionar su infraestructura AWS. AWS CloudFormation es un sistema gestionado Servicio de AWS que ofrece un amplio soporte de configuraciones de recursos y propiedades para el aprovisionamiento de servicios en AWS. Con él AWS CloudFormation, puede realizar despliegues de infraestructura de forma predecible y repetida, con la posibilidad de revertirlos en caso de error. Si ya lo conoce AWS CloudFormation, no tiene que aprender un nuevo servicio de administración de iAC para empezar con él. AWS CDK

Comience a desarrollar su aplicación rápidamente con construcciones

Desarrolle más rápido utilizando y compartiendo componentes reutilizables denominados componentes constructos. Utilice construcciones de bajo nivel para definir los AWS CloudFormation recursos individuales y sus propiedades. Utilice estructuras de alto nivel para definir rápidamente los componentes más grandes de su aplicación, con valores predeterminados

razonables y seguros para sus AWS recursos, lo que permite definir más infraestructura con menos código.

Cree sus propias estructuras personalizadas para sus casos de uso específicos y compártalas en toda la organización o incluso con el público.

Ejemplo de AWS CDK

A continuación se muestra un ejemplo del uso de la biblioteca AWS CDK Constructs para crear un servicio Amazon Elastic Container Service (Amazon ECS) con un tipo de lanzamiento. AWS Fargate (Fargate) Para obtener más información sobre este ejemplo, consultethe section called “ECS”.

TypeScript

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

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

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

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

JavaScript

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

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

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

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

module.exports = { MyEcsConstructStack }
```

Python

```
class MyEcsConstructStack(Stack):

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

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

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

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

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

Java

```
public class MyEcsConstructStack extends Stack {

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

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

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

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

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

C#

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

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

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

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

Go

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

 var sprops awscdk.StackProps

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

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

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

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

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

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

 return stack

}
```

Esta clase produce una AWS CloudFormation plantilla de más de 500 líneas. La implementación de la AWS CDK aplicación produce más de 50 recursos de los siguientes tipos.

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

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

AWS CDK features

El AWS CDKGitHub repositorio

Para ver el AWS CDK GitHub repositorio oficial, consulte aws-cdk. Aquí puede enviar sus ediciones, ver nuestra licencia, hacer un seguimiento de las versiones y mucho más.

Como AWS CDK es de código abierto, el equipo lo alienta a contribuir para que sea una herramienta aún mejor. Para obtener más información, consulte Contribuir a la AWS Cloud Development Kit (AWS CDK).

La referencia AWS CDK de la API

La biblioteca AWS CDK Construct proporciona API para definir su aplicación de CDK y añadir construcciones de CDK a la aplicación. Para obtener más información, consulte la Referencia de la API de AWS CDK.

El modelo de programación Construct

El modelo de programación de construcciones (CPM) amplía los conceptos subyacentes AWS CDK a dominios adicionales. Otras herramientas que utilizan el CPM incluyen:

- CDK para Terraform (CDKtf)
- CDK para Kubernetes (CDK8s)
- Projen, para crear configuraciones de proyectos

The Construct Hub

The Construct Hub es un registro en línea donde puede encontrar, publicar y compartir AWS CDK bibliotecas de código abierto.

Siguientes pasos

Para empezar a usar AWS CDK, consulteCómo empezar con el AWS CDK.

Más información

Para seguir aprendiendo sobre el AWS CDK, consulte lo siguiente:

- AWS CDK conceptos— Conceptos y términos importantes para el AWS CDK.
- AWS CDK Taller: taller práctico para aprender y usar el AWS CDK.
- AWS CDK Patterns: colección de código abierto de patrones de arquitectura AWS sin servidor, creada para expertos. AWS CDK AWS
- AWS CDK ejemplos de código: GitHub repositorio de proyectos de ejemplo AWS CDK .
- cdk.dev: centro impulsado por la comunidad AWS CDK, que incluye un espacio de trabajo comunitario. Slack
- Impresionante CDK: GitHub repositorio que contiene una lista seleccionada de proyectos, guías, blogs y otros recursos de AWS CDK código abierto.
- AWS Construcciones de soluciones: patrones de infraestructura de configuración aprobados como código (IaC) que se pueden ensamblar fácilmente en aplicaciones listas para la producción.
- AWS Blog de herramientas para desarrolladores: publicaciones de blog filtradas para. AWS CDK
- AWS CDK activado Stack Overflow: preguntas etiquetadas con aws-cdk activado. Stack Overflow
- AWS CDK tutorial para AWS Cloud9 — Tutorial sobre cómo usarlo AWS CDK con el AWS Cloud9 entorno de desarrollo.

Para obtener más información sobre temas relacionados con el AWS CDK, consulte lo siguiente:

- AWS CloudFormation conceptos: dado que AWS CDK está diseñado para funcionar con AWS CloudFormation ellos, le recomendamos que aprenda y comprenda AWS CloudFormation los conceptos clave.
- AWS Glosario: definiciones de los términos clave utilizados en todas partes. AWS

Para obtener más información sobre las herramientas relacionadas con las AWS CDK que se pueden utilizar para simplificar el desarrollo y la implementación de aplicaciones sin servidor, consulte lo siguiente:

- AWS Serverless Application Model— Una herramienta de código abierto para desarrolladores que simplifica y mejora la experiencia de creación y ejecución de aplicaciones sin servidor. AWS
- AWSChalice— Un marco para escribir aplicaciones sin servidor. Python

AWS CDK conceptos

Conozca los conceptos básicos detrás de AWS Cloud Development Kit (AWS CDK).

AWS CDK y IaC

AWS CDK Se trata de un marco de código abierto que puede utilizar para gestionar su AWS infraestructura mediante código. Este enfoque se conoce como infraestructura como código (IaC). Al administrar y aprovisionar su infraestructura como código, trata su infraestructura de la misma manera que los desarrolladores tratan el código. Esto proporciona muchos beneficios, como el control de versiones y la escalabilidad. Para obtener más información sobre la IaC, consulte ¿Qué es la infraestructura como código?

AWS CDK y AWS CloudFormation

AWS CDK Está estrechamente integrado con AWS CloudFormation. AWS CloudFormation es un servicio totalmente gestionado que puede utilizar para gestionar y aprovisionar su infraestructura AWS. Con AWS CloudFormationél, puede definir su infraestructura en plantillas y desplegarlas en ellas AWS CloudFormation. A continuación, el AWS CloudFormation servicio aprovisiona la infraestructura de acuerdo con la configuración definida en las plantillas.

AWS CloudFormation las plantillas son declarativas, lo que significa que declaran el estado o resultado deseado de su infraestructura. Con JSON o YAML, declaras tu AWS infraestructura definiendo AWS los recursos y las propiedades. Los recursos representan los numerosos servicios AWS y las propiedades representan la configuración deseada de dichos servicios. Al implementar la plantilla en AWS CloudFormation, los recursos y sus propiedades configuradas se aprovisionan tal y como se describe en la plantilla.

Con ella AWS CDK, puede gestionar su infraestructura de forma imperativa mediante lenguajes de programación de uso general. En lugar de simplemente definir el estado deseado de forma declarativa, puede definir la lógica o la secuencia necesaria para alcanzar el estado deseado. Por ejemplo, puede usar `if` sentencias o bucles condicionales que determinen cómo alcanzar el estado final deseado para su infraestructura.

La infraestructura creada con el AWS CDK se traduce finalmente o se sintetiza en AWS CloudFormation plantillas y se implementa mediante el AWS CloudFormation servicio. Por lo

tanto, si bien AWS CDK ofrece un enfoque diferente para crear su infraestructura, usted seguirá beneficiándose de ello AWS CloudFormation, como un amplio soporte para la configuración de los AWS recursos y unos procesos de implementación sólidos.

Para obtener más información AWS CloudFormation, consulte ¿Qué es AWS CloudFormation? en la Guía AWS CloudFormation del usuario.

AWS CDK y abstracciones

Con AWS CloudFormation, debe definir cada detalle de cómo se configuran sus recursos. Esto proporciona la ventaja de tener un control total sobre su infraestructura. Sin embargo, esto requiere que aprenda, comprenda y cree plantillas sólidas que contengan detalles de la configuración de los recursos y las relaciones entre los recursos, como los permisos y las interacciones basadas en eventos.

Con él AWS CDK, puede tener el mismo control sobre las configuraciones de sus recursos. Sin embargo, AWS CDK también ofrece potentes abstracciones, que pueden acelerar y simplificar el proceso de desarrollo de la infraestructura. Por ejemplo, AWS CDK incluye construcciones que proporcionan configuraciones predeterminadas razonables y métodos auxiliares que generan código repetitivo para usted. AWS CDK También ofrece herramientas, como la interfaz de línea de AWS CDK comandos (AWS CDK CLI), que realizan acciones de administración de la infraestructura por usted.

Obtenga más información sobre AWS CDK los conceptos básicos

Interactuando con el AWS CDK

Cuando lo utilice con AWS CDK, interactuará principalmente con la biblioteca de AWS construcciones y el AWS CDK CLI.

Desarrollando con el AWS CDK

Se AWS CDK puede escribir en cualquier lenguaje de programación compatible. Se comienza con un proyecto de CDK, que contiene una estructura de carpetas y archivos, incluidos los activos. Dentro del proyecto, se crea una aplicación de CDK. Dentro de la aplicación, se define una pila, que representa directamente una CloudFormation pila. Dentro de la pila, se definen AWS los recursos y las propiedades mediante componentes fijos.

Desplegar con AWS CDK

Las aplicaciones de CDK se implementan en un AWS entorno. Antes de la implementación, debe realizar un arranque único para preparar el entorno.

Más información

Para obtener más información sobre los conceptos AWS CDK básicos, consulte los temas de esta sección.

Lenguajes de programación admitidos

AWS Cloud Development Kit (AWS CDK) Cuenta con un soporte de primera clase para los siguientes lenguajes de programación de uso general:

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

En teoría, también se pueden utilizar otros JVM .NET CLR lenguajes, pero no ofrecemos soporte oficial en este momento.

> **Note**
>
> Actualmente, esta guía no incluye instrucciones ni ejemplos de código para, Go aparte dethe section called “En Go”.

AWS CDK Está desarrollado en un idioma,TypeScript. Para admitir los otros idiomas, AWS CDK utiliza una herramienta llamada JSIIa generar enlaces lingüísticos.

Intentamos ofrecer las convenciones habituales de cada idioma para que el desarrollo sea AWS CDK lo más natural e intuitivo posible. Por ejemplo, distribuimos los módulos de AWS Construct Library

utilizando el repositorio estándar de su idioma preferido y usted los instala utilizando el administrador de paquetes estándar del idioma. Los métodos y las propiedades también se nombran según los patrones de nomenclatura recomendados en su idioma.

A continuación se muestran algunos ejemplos de código:

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

```
                        }});
```

Go

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

> **Note**
>
> Estos fragmentos de código están destinados únicamente a fines ilustrativos. Están incompletos y no se ejecutarán como están.

La biblioteca AWS Construct se distribuye utilizando las herramientas de administración de paquetes estándar de cada idiomaNPM, incluidasPyPi,Maven, yNuGet. También ofrecemos una versión de la referencia de la AWS CDK API para cada idioma.

Para ayudarte a usar la AWS CDK en tu idioma preferido, esta guía incluye los siguientes temas sobre los idiomas compatibles:

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

TypeScriptfue el primer idioma admitido por AWS CDK, y gran parte del código de AWS CDK ejemplo está escrito en élTypeScript. Esta guía incluye un tema específico para mostrar cómo adaptar el TypeScript AWS CDK código para su uso con los demás lenguajes compatibles. Para obtener más información, consulte Comparación AWS CDKTypeScript con otros lenguajes.

AWS CDK proyectos

Un AWS Cloud Development Kit (AWS CDK) proyecto representa los archivos y carpetas que contienen el código CDK. El contenido variará en función del lenguaje de programación.

Puede crear su AWS CDK proyecto manualmente o con el AWS CDK comando Command Line Interface (AWS CDK CLI)`cdk init`. En este tema, haremos referencia a la estructura del proyecto y a las convenciones de nomenclatura de los archivos y carpetas creados por la CLI de AWS CDK. Puede personalizar y organizar sus proyectos de CDK para adaptarlos a sus necesidades.

> **Note**
>
> La estructura del proyecto creada por el AWS CDK CLI puede variar según las versiones a lo largo del tiempo.

Temas

Archivos y carpetas universales

.git

Si lo ha `git` instalado, inicializa AWS CDK CLI automáticamente un Git repositorio para su proyecto. El `.git` directorio contiene información sobre el repositorio.

.gitignore

Archivo de texto utilizado por Git para especificar los archivos y carpetas que se deben ignorar.

README.md

Archivo de texto que proporciona orientación básica e información importante para gestionar el AWS CDK proyecto. Modifique este archivo según sea necesario para documentar la información importante relacionada con su proyecto de CDK.

cdk.json

Archivo de configuración para. AWS CDK Este archivo proporciona instrucciones AWS CDK CLI sobre cómo ejecutar su aplicación.

Archivos y carpetas para idiomas específicos

Los siguientes archivos y carpetas son exclusivos de cada lenguaje de programación compatible.

TypeScript

El siguiente es un ejemplo de proyecto creado en el `my-cdk-ts-project` directorio mediante el `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

Archivo que especifica qué archivos y carpetas se deben ignorar al publicar un paquete en el registro. npm Este archivo es similar a los npm paquetes `.gitignore`, pero es específico de ellos.

bin/ .ts my-cdk-ts-project

El archivo de la aplicación define la aplicación CDK. Los proyectos de CDK pueden contener uno o más archivos de aplicación. Los archivos de aplicación se almacenan en la `bin` carpeta.

El siguiente es un ejemplo de un archivo de aplicación básico que define una aplicación 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

Archivo de configuración para. Jest Jestes un marco JavaScript de pruebas popular.

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

El archivo de pila define tu pila de CDK. Dentro de su pila, usted define AWS los recursos y las propiedades mediante construcciones.

El siguiente es un ejemplo de un archivo de pila básico que define una pila de 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

Carpeta común en Node.js los proyectos que contiene dependencias para su proyecto.

package-lock.json

Archivo de metadatos que funciona con el `package.json` archivo para administrar las versiones de las dependencias.

paquete.json

Archivo de metadatos que se utiliza habitualmente en Node.js los proyectos. Este archivo contiene información sobre su proyecto de CDK, como el nombre del proyecto, las definiciones de los scripts, las dependencias y otra información de importación a nivel del proyecto.

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

Se crea una carpeta de pruebas para organizar las pruebas de su proyecto de CDK. También se crea un archivo de prueba de muestra.

Puede escribir pruebas TypeScript y utilizarlas Jest para compilar el TypeScript código antes de ejecutarlas.

tsconfig.json

Archivo de configuración utilizado en TypeScript proyectos que especifica las opciones del compilador y la configuración del proyecto.

JavaScript

A continuación se muestra un ejemplo de proyecto creado en el `my-cdk-js-project` directorio mediante el `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

Archivo que especifica qué archivos y carpetas se deben ignorar al publicar un paquete en el registro. npm Este archivo es similar a los npm paquetes `.gitignore`, pero es específico de ellos.

bin/ .js my-cdk-js-project

El archivo de la aplicación define la aplicación CDK. Los proyectos de CDK pueden contener uno o más archivos de aplicación. Los archivos de aplicación se almacenan en la `bin` carpeta.

El siguiente es un ejemplo de un archivo de aplicación básico que define una aplicación 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

Archivo de configuración para. Jest Jestes un marco JavaScript de pruebas popular.

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

El archivo de pila define la pila de CDK. Dentro de su pila, usted define AWS los recursos y las propiedades mediante construcciones.

El siguiente es un ejemplo de un archivo de pila básico que define una pila de 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

Carpeta común en Node.js los proyectos que contiene dependencias para su proyecto.

package-lock.json

Archivo de metadatos que funciona con el `package.json` archivo para administrar las versiones de las dependencias.

paquete.json

Archivo de metadatos que se utiliza habitualmente en Node.js los proyectos. Este archivo contiene información sobre su proyecto de CDK, como el nombre del proyecto, las definiciones de los scripts, las dependencias y otra información de importación a nivel del proyecto.

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

Se crea una carpeta de pruebas para organizar las pruebas de su proyecto de CDK. También se crea un archivo de prueba de muestra.

Puede escribir pruebas JavaScript y utilizarlas Jest para compilar el JavaScript código antes de ejecutarlas.

Python

El siguiente es un ejemplo de proyecto creado en el `my-cdk-py-project` directorio mediante el `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

El CDK crea CLI automáticamente un entorno virtual para su proyecto. El `.venv` directorio hace referencia a este entorno virtual.

app.py

El archivo de la aplicación define la aplicación CDK. Los proyectos de CDK pueden contener uno o más archivos de aplicación.

El siguiente es un ejemplo de un archivo de aplicación básico que define una aplicación de 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

Directorio que contiene tus archivos de pila. El CDK CLI crea lo siguiente aquí:

- `__init__.py` — Un fichero de definición de Python paquete vacío.
- `my_cdk_py_project`— Archivo que define su pila de CDK. A continuación, defina AWS los recursos y las propiedades de la pila mediante construcciones.

El siguiente es un ejemplo de un archivo apilado:

```
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

Archivo similar `arequirements.txt`, pero utilizado para gestionar las dependencias específicamente con fines de desarrollo y no de producción.

requirements.txt

Archivo común utilizado en Python los proyectos para especificar y gestionar las dependencias del proyecto.

source.bat

El archivo Batch para Windows eso se usa para configurar el entorno Python virtual.

exámenes

Directorio que contiene las pruebas para su proyecto de CDK.

El siguiente es un ejemplo de una prueba unitaria:

```
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

El siguiente es un ejemplo de proyecto creado en el `my-cdk-java-project` directorio mediante el `cdk init --language java` comando:

```
my-cdk-java-project
### .git
### .gitignore
```

```
### README.md
### cdk.json
### pom.xml
### src
    ### main
    ### test
```

pom.xml

Archivo que contiene información de configuración y metadatos sobre su proyecto de CDK. Este archivo forma parte deMaven.

src/main

Directorio que contiene los archivos de la aplicación y de la pila.

El siguiente es un ejemplo de archivo de aplicación:

```
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();
 }
}
```

El siguiente es un ejemplo de archivo de pila:

```
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

Directorio que contiene sus archivos de prueba. A continuación, se muestra un ejemplo:

```
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#

El siguiente es un ejemplo de proyecto creado en el `my-cdk-csharp-project` directorio mediante el `cdk init --language csharp` comando:

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

src/ MyCdkCsharpProject

Directorio que contiene los archivos de la aplicación y de la pila.

El siguiente es un ejemplo de archivo de aplicación:

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

El siguiente es un ejemplo de archivo de pila:

```
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
  }
 }
}
```

Este directorio también contiene lo siguiente:

- `GlobalSuppressions.cs`— Archivo utilizado para suprimir advertencias o errores específicos del compilador en todo el proyecto.
- `.csproj`— Archivo basado en XML que se utiliza para definir los ajustes, las dependencias y las configuraciones de compilación del proyecto.

MyCdkCsharpProjectsrc/ .sln

Microsoft Visual Studio Solution Filese utiliza para organizar y gestionar proyectos relacionados.

Go

El siguiente es un ejemplo de proyecto creado en el `my-cdk-go-project` directorio mediante el `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

Archivo que contiene información del módulo y que se utiliza para gestionar las dependencias y el control de versiones del proyecto. Go

my-cdk-go-project.go

Archivo que define la aplicación y las pilas de CDK.

A continuación, se muestra un ejemplo:

```
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

Archivo que define un ejemplo de prueba.

A continuación, se muestra un ejemplo:

```
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 aplicaciones

La AWS Cloud Development Kit (AWS CDK) aplicación o aplicación es una colección de una o más pilas de CDK. Las pilas son un conjunto de una o más estructuras que definen AWS los recursos y las propiedades. Por lo tanto, la agrupación general de las pilas y construcciones se conoce como aplicación CDK.

Temas

Definir aplicaciones

Para crear una aplicación, defina una instancia de aplicación en el archivo de aplicación de su proyecto. Para ello, importe y utilice el componente fijo Appde la biblioteca AWS de componentes. La App construcción no requiere ningún argumento de inicialización. Es la única construcción que se puede utilizar como raíz.

`Stack`Las clases `App` y de la biblioteca de AWS construcciones son construcciones únicas. En comparación con otras construcciones, no configuran AWS los recursos por sí mismas. En cambio, se utilizan para proporcionar contexto a las otras construcciones. Todos los constructos que representan AWS recursos deben definirse, directa o indirectamente, dentro del ámbito de un `Stack` constructo. `Stack`las construcciones se definen dentro del ámbito de una App construcción.

Luego, las aplicaciones se sintetizan para crear AWS CloudFormation plantillas para tus pilas. A continuación, se muestra un ejemplo:

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)
```

Las pilas de una sola aplicación pueden hacer referencia fácilmente a los recursos y propiedades de las demás. AWS CDK Deduce las dependencias entre las pilas para poder desplegarlas en el orden correcto. Puedes implementar una o todas las pilas de una aplicación con un solo comando. `cdk deploy`

El árbol de construcción

Las construcciones se definen dentro de otras construcciones mediante el `scope` argumento que se pasa a cada construcción, con la `App` clase como raíz. De esta forma, una AWS CDK aplicación define una jerarquía de construcciones conocida como árbol de construcciones.

La raíz de este árbol es tu aplicación, que es una instancia de la `App` clase. Dentro de la aplicación, se crean instancias de una o más pilas. Dentro de las pilas, se crean instancias de componentes

fijos, que a su vez pueden crear instancias de recursos u otros componentes, y así sucesivamente en el resto del árbol.

Los constructos siempre se definen de forma explícita dentro del ámbito de otro constructo, lo que crea relaciones entre los constructos. Casi siempre, debes pasar `this` (en Python`self`) como ámbito, lo que indica que la nueva construcción es un elemento secundario de la construcción actual. El patrón previsto consiste en derivar la construcción y, a continuación `Construct`, crear una instancia de las construcciones que utiliza en su constructor.

Pasar el ámbito de forma explícita permite que cada construcción se añada al árbol, y este comportamiento está contenido exclusivamente en la `Construct` clase base. Funciona de la misma manera en todos los idiomas admitidos por el AWS CDK y no requiere ninguna personalización adicional.

> **Important**
>
> Técnicamente, es posible pasar algún ámbito distinto `this` al de instanciar una construcción. Puedes añadir componentes fijos en cualquier parte del árbol o incluso en otra pila de la misma aplicación. Por ejemplo, puedes escribir una función de estilo mixto que añada construcciones a un ámbito pasado como argumento. La dificultad práctica en este caso es que no es fácil garantizar que los identificadores que elija para sus construcciones sean únicos dentro del ámbito de otra persona. Esta práctica también hace que el código sea más difícil de entender, mantener y reutilizar. Casi siempre es mejor encontrar una manera de expresar tu intención sin recurrir a abusar del `scope` argumento.

AWS CDK Utiliza los ID de todas las construcciones de la ruta desde la raíz del árbol hasta cada construcción secundaria para generar las ID únicas que necesita. AWS CloudFormation Este enfoque significa que los ID de construcción solo tienen que ser únicos dentro de su ámbito, y no en toda la pila, como en el caso de los nativos AWS CloudFormation. Sin embargo, si mueves una construcción a un ámbito diferente, el identificador único de la pila generado cambia y AWS CloudFormation no se considerará el mismo recurso.

El árbol de construcciones es independiente de las construcciones que definas en el código. AWS CDK Sin embargo, se puede acceder a él a través del `node` atributo de cualquier construcción, que es una referencia al nodo que representa esa construcción en el árbol. Cada nodo es una `Node`instancia, cuyos atributos proporcionan acceso a la raíz del árbol y a los ámbitos principales y secundarios del nodo.

1. `node.children`— Los hijos directos de la construcción.
2. `node.id`— El identificador del constructo dentro de su ámbito.
3. `node.path`— La ruta completa del constructo, incluidos los identificadores de todos sus padres.
4. `node.root`— La raíz del árbol de construcción (la aplicación).
5. `node.scope`— El ámbito (principal) de la construcción, o indefinido si el nodo es la raíz.
6. `node.scopes`— Todos los padres de la construcción, hasta la raíz.
7. `node.uniqueId`— El identificador alfanumérico único de esta construcción dentro del árbol (de forma predeterminada, se genera a partir de `node.path` un hash).

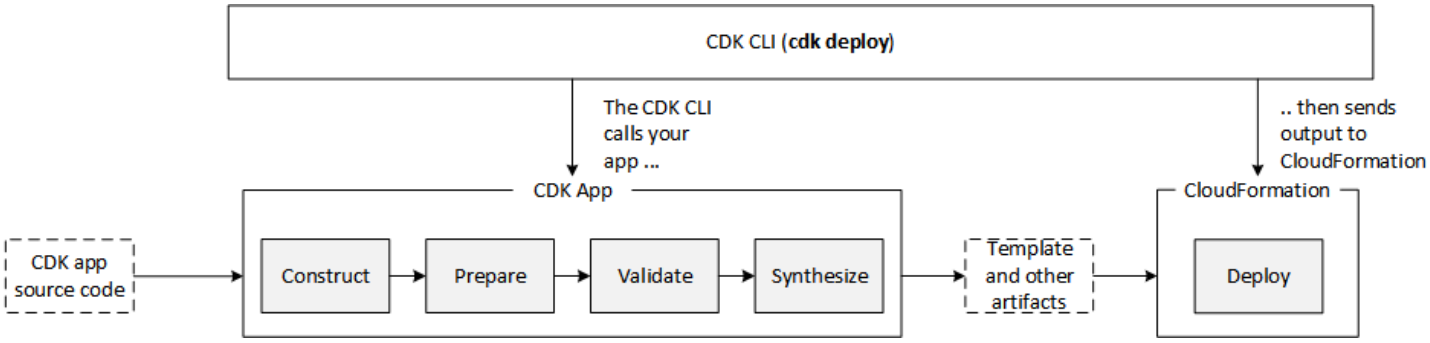
El árbol de construcciones define un orden implícito en el que las construcciones se sintetizan en recursos en la plantilla final. AWS CloudFormation Donde se debe crear un recurso antes que otro, AWS CloudFormation o la biblioteca de AWS construcción generalmente deduce la dependencia. A continuación, se aseguran de que los recursos se creen en el orden correcto.

También puede añadir una dependencia explícita entre dos nodos mediante`node.addDependency()`. Para obtener más información, consulta las dependencias en la referencia de la AWS CDK API.

AWS CDK Proporciona una forma sencilla de visitar todos los nodos del árbol de construcción y realizar una operación en cada uno de ellos. Para obtener más información, consulte the section called “Aspectos”.

El ciclo de vida de la aplicación

Al implementar la aplicación CDK, se llevan a cabo las siguientes fases. Esto se conoce como ciclo de vida de la aplicación:



Una AWS CDK aplicación pasa por las siguientes fases de su ciclo de vida.

- Construcción (o inicialización): el código crea una instancia de todas las construcciones definidas y, a continuación, las vincula entre sí. En esta etapa, se instancian todas las construcciones (app, stacks y sus construcciones secundarias) y se ejecuta la cadena de constructores. La mayor parte del código de la aplicación se ejecuta en esta etapa.
- Preparación: todas las construcciones que han implementado el `prepare` método participan en una ronda final de modificaciones para configurar su estado final. La fase de preparación se produce automáticamente. Como usuario, no ves ningún comentario sobre esta fase. Es poco frecuente que necesites usar el anzuelo de «preparar» y, por lo general, no se recomienda. Tenga mucho cuidado al mutar el árbol de construcción durante esta fase, ya que el orden de las operaciones podría afectar al comportamiento.
- Validación: todas las construcciones que han implementado el `validate` método pueden validarse a sí mismas para garantizar que se encuentren en un estado que permita su despliegue correcto. Se le notificará cualquier error de validación que se produzca durante esta fase. Por lo general, te recomendamos que realices la validación lo antes posible (normalmente tan pronto como recibas información) y que descartes las excepciones lo antes posible. Realizar la validación de forma temprana mejora la fiabilidad, ya que los seguimientos de las pilas son más precisos y garantiza que el código pueda seguir ejecutándose de forma segura.
- Síntesis: esta es la etapa final de la ejecución de la AWS CDK aplicación. Se activa con una llamada a`app.synth()`, recorre el árbol de construcción e invoca el `synthesize` método en todas las construcciones. Las construcciones que se implementan `synthesize` pueden participar en la síntesis y emitir artefactos de implementación al ensamblaje de nube resultante. Estos artefactos incluyen AWS CloudFormation plantillas, paquetes de AWS Lambda aplicaciones, activos de archivos e Docker imágenes y otros artefactos de implementación. the section called “Conjuntos en la nube”describe el resultado de esta fase. En la mayoría de los casos, no necesitará implementar el `synthesize` método.
- Despliegue: en esta fase, AWS CDK CLI se toman los artefactos de despliegue en la nube generados por la fase de síntesis y se despliegan en un AWS entorno. Carga los activos a Amazon S3 y Amazon ECR, o a cualquier lugar al que tengan que ir. A continuación, inicia una AWS CloudFormation implementación para implementar la aplicación y crear los recursos.

Cuando comience la fase de AWS CloudFormation implementación, la AWS CDK aplicación ya habrá finalizado y se habrá cerrado. Esto tiene las siguientes implicaciones:

- La AWS CDK aplicación no puede responder a los eventos que se producen durante la implementación, como la creación de un recurso o la finalización de toda la implementación. Para ejecutar el código durante la fase de despliegue, debes inyectarlo en la AWS CloudFormation

plantilla como un recurso personalizado. Para obtener más información sobre cómo agregar un recurso personalizado a tu aplicación, consulta el AWS CloudFormation módulo o el ejemplo del recurso personalizado.

- Es posible que la AWS CDK aplicación tenga que funcionar con valores que no se pueden conocer en el momento en que se ejecuta. Por ejemplo, si la AWS CDK aplicación define un bucket de Amazon S3 con un nombre generado automáticamente y usted recupera el atributo `bucket.bucketName` (Python:`bucket_name`), ese valor no es el nombre del bucket implementado. En su lugar, se obtiene un `Token` valor. Para determinar si un valor concreto está disponible, llama a `cdk.isUnresolved(value)` (Python:`is_unresolved`). Para obtener más información, consulte the section called “Tokens”.

Conjuntos en la nube

La llamada a `app.synth()` es lo que indica que hay que AWS CDK sintetizar un ensamblaje en la nube a partir de una aplicación. Por lo general, no se interactúa directamente con los ensamblajes en la nube. Son archivos que incluyen todo lo necesario para implementar la aplicación en un entorno de nube. Por ejemplo, incluye una AWS CloudFormation plantilla para cada pila de la aplicación. También incluye una copia de cualquier recurso de archivo o imagen de Docker a los que hagas referencia en tu aplicación.

Consulte la especificación de ensamblaje en la nube para obtener detalles sobre el formato de los ensamblados en la nube.

Para interactuar con el ensamblaje de nubes que crea tu AWS CDK aplicación, normalmente usas el AWS CDK CLI. Sin embargo, cualquier herramienta que pueda leer el formato de ensamblaje en la nube se puede usar para implementar su aplicación.

Ejecutando tu aplicación

El CDK CLI necesita saber cómo ejecutar tu AWS CDK aplicación. Si creaste el proyecto a partir de una plantilla mediante el `cdk init` comando, el `cdk.json` archivo de tu aplicación incluye una `app` clave. Esta clave especifica el comando necesario para el idioma en el que está escrita la aplicación. Si tu idioma requiere compilación, la línea de comandos realiza este paso antes de ejecutar la aplicación, para que no te olvides de hacerlo.

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"
}
```

Si no creaste tu proyecto con el CDKCLI, o si deseas anular la línea de comandos incluida`cdk.json`, puedes usar --app esta opción al ejecutar el comando. `cdk`

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

La parte *`ejecutable`* del comando indica el comando que debe ejecutarse para ejecutar la aplicación CDK. Use comillas como se muestra, ya que estos comandos contienen espacios. El

comando cdk es un subcomando similar synth o deploy que le indica al CDK CLI lo que quieres hacer con tu aplicación. Siga esto con cualquier opción adicional necesaria para ese subcomando.

También AWS CDK CLI pueden interactuar directamente con un conjunto de nubes ya sintetizado. Para ello, pase el directorio en el que está almacenado el ensamblaje de la nube. --app En el siguiente ejemplo, se enumeran las pilas definidas en el ensamblaje de nube en el que se almacenan. `./my-cloud-assembly`

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

Pilas

Una AWS Cloud Development Kit (AWS CDK) pila es una colección de una o más construcciones que definen AWS los recursos. Cada pila de CDK representa una AWS CloudFormation pila de tu aplicación de CDK. En el momento de la implementación, las construcciones de una pila se aprovisionan como una sola unidad, denominada pila. AWS CloudFormation Para obtener más información sobre las AWS CloudFormation pilas, consulte Cómo trabajar con pilas en la Guía del usuario.AWS CloudFormation

Como las pilas de CDK se implementan a través de AWS CloudFormation pilas, AWS CloudFormation se aplican cuotas y limitaciones. Para obtener más información, consulte las cuotas.AWS CloudFormation

Temas

- Definir pilas
- Trabajo con pilas de

Definir pilas

Las pilas se definen en el contexto de una aplicación. Para definir una pila, utilice la `Stack` clase de la biblioteca AWS Construct. Las pilas se pueden definir de cualquiera de las siguientes maneras:

- Directamente dentro del ámbito de la aplicación.
- Indirectamente por cualquier construcción del árbol.

El siguiente ejemplo define una aplicación de CDK que contiene dos pilas:

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();
```

El siguiente ejemplo es un patrón común para definir una pila en un archivo independiente. Aquí, ampliamos o heredamos la `Stack` clase y definimos un constructor que acepta `scopeid`, y`props`. Luego, invocamos el constructor de la `Stack` clase base usando `super` los valores recibidos `scopeid`, y. `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
}
```

En el siguiente ejemplo, se declara una clase de pila denominada `MyFirstStack` que incluye un único bucket de 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
}
```

Sin embargo, este código solo ha declarado una pila. Para que la pila se sintetice realmente en una AWS CloudFormation plantilla y se despliegue, debe crearse una instancia de la misma. Además, como todas las construcciones de CDK, se debe instanciar en algún contexto. Ese `App` es ese contexto.

Si utilizas la plantilla de AWS CDK desarrollo estándar, tus pilas se instanciarán en el mismo archivo en el que instanciaste el objeto. `App`

TypeScript

El archivo que lleva el nombre de tu proyecto (por ejemplo,`hello-cdk.ts`) en la carpeta de tu proyecto. `bin`

JavaScript

El archivo que lleva el nombre de tu proyecto (por ejemplo,`hello-cdk.js`) en la `bin` carpeta de tu proyecto.

Python

El archivo `app.py` del directorio principal del proyecto.

Java

El archivo denominado*`ProjectName`*`App.java`, por ejemplo`HelloCdkApp.java`, está ubicado en lo profundo del `src/main` directorio.

C#

El archivo nombrado `Program.cs` debajo`src\`*`ProjectName`*, por ejemplo`src\HelloCdk\Program.cs`.

La API de pila

El objeto Stack proporciona una API completa, que incluye lo siguiente:

- `Stack.of(construct)`— Un método estático que devuelve la pila en la que se define una construcción. Esto resulta útil si necesitas interactuar con una pila desde una construcción reutilizable. La llamada falla si no se encuentra una pila en el alcance.
- `stack.stackName`(Python:`stack_name`) — Devuelve el nombre físico de la pila. Como se mencionó anteriormente, todas las AWS CDK pilas tienen un nombre físico que AWS CDK pueden resolver durante la síntesis.
- `stack.region`y `stack.account` — Devuelve la AWS región y la cuenta, respectivamente, en las que se desplegará esta pila. Estas propiedades devuelven una de las siguientes opciones:
  - La cuenta o región especificada de forma explícita cuando se definió la pila
  - Un token codificado en cadenas que se resuelve según los AWS CloudFormation pseudoparámetros de la cuenta y la región para indicar que esta pila es independiente del entorno

  Para obtener información sobre cómo se determinan los entornos de las pilas, consulte. the section called “Entornos”
- `stack.addDependency(stack)`(Python: `stack.add_dependency(stack)` — Se puede usar para definir explícitamente el orden de dependencia entre dos pilas. El cdk deploy comando respeta este orden cuando despliega varias pilas a la vez.
- `stack.tags`— Devuelve una TagManagerque puedes usar para añadir o eliminar etiquetas a nivel de pila. Este administrador de etiquetas etiqueta todos los recursos de la pila y también etiqueta la propia pila cuando se crea mediante ella. AWS CloudFormation

- `stack.partition`, `stack.urlSuffix` (Python:`url_suffix`), `stack.stackId` (Python:`stack_id`) y `stack.notificationArn` (Python:`notification_arn`): devuelven los tokens que se resuelven en los AWS CloudFormation pseudoparámetros respectivos, como`{ "Ref": "AWS::Partition" }`. Estos tokens están asociados al objeto de pila específico para que el AWS CDK marco pueda identificar las referencias entre pilas.
- `stack.availabilityZones`(Python:`availability_zones`): devuelve el conjunto de zonas de disponibilidad disponibles en el entorno en el que se implementa esta pila. En el caso de las pilas independientes del entorno, esto siempre devuelve una matriz con dos zonas de disponibilidad. En el caso de las pilas específicas del entorno, AWS CDK consulta el entorno y devuelve el conjunto exacto de zonas de disponibilidad disponibles en la región que especificó.
- `stack.parseArn(arn)`and `stack.formatArn(comps)` (Python:`parse_arn`,`format_arn`): se puede utilizar para trabajar con nombres de recursos de Amazon (ARN).
- `stack.toJsonString(obj)`(Python:`to_json_string`): se puede usar para formatear un objeto arbitrario como una cadena JSON que se puede incrustar en una AWS CloudFormation plantilla. El objeto puede incluir símbolos, atributos y referencias, que solo se resuelven durante el despliegue.
- `stack.templateOptions`(Python:`template_options`): se usa para especificar opciones de AWS CloudFormation plantilla, como transformación, descripción y metadatos, para la pila.

Trabajo con pilas de

Para enumerar todas las pilas de una aplicación de CDK, usa el cdk ls comando. El ejemplo anterior generaría lo siguiente:

```
stack1
stack2
```

Las pilas se implementan como parte de una AWS CloudFormation pila en un AWS entorno. El entorno cubre una Cuenta de AWS y específica. Región de AWS

Cuando ejecutas el cdk synth comando para una aplicación con varias pilas, el ensamblaje en la nube incluye una plantilla independiente para cada instancia de pila. Incluso si las dos pilas son instancias de la misma clase, las AWS CDK emite como dos plantillas individuales.

Puede sintetizar cada plantilla especificando el nombre de la pila en el comando. cdk synth En el siguiente ejemplo, se sintetiza la plantilla de stack1.

```
$ cdk synth stack1
```

Este enfoque es conceptualmente diferente de la forma en que se usan normalmente las AWS CloudFormation plantillas, donde una plantilla se puede implementar varias veces y parametrizar a través de parámetros.AWS CloudFormation Si bien AWS CloudFormation los parámetros se pueden definir en el AWS CDK, por lo general no se recomienda utilizarlos porque los AWS CloudFormation parámetros solo se resuelven durante el despliegue. Esto significa que no puede determinar su valor en el código.

Por ejemplo, para incluir condicionalmente un recurso en tu aplicación en función del valor de un parámetro, debes configurar una AWS CloudFormation condición y etiquetar el recurso con ella. AWS CDK Adopta un enfoque en el que las plantillas concretas se resuelven en el momento de la síntesis. Por lo tanto, puede utilizar una sentencia if para comprobar el valor y determinar si se debe definir un recurso o si se debe aplicar algún comportamiento.

> **Note**
>
> AWS CDK Proporciona la mayor resolución posible durante el tiempo de síntesis para permitir el uso idiomático y natural del lenguaje de programación.

Como cualquier otra construcción, las pilas se pueden componer juntas en grupos. El código siguiente muestra un ejemplo de un servicio que consta de tres pilas: un plano de control, un plano de datos y pilas de supervisión. La construcción del servicio se define dos veces: una para el entorno beta y otra para el entorno de producción.

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();
    }
}
```

En última instancia, esta AWS CDK aplicación consta de seis pilas, tres para cada entorno:

```
$ cdk ls

betacpDA8372D3
betadataE23DB2BA
betamon632BD457
prodcp187264CE
proddataF7378CE5
prodmon631A1083
```

Los nombres físicos de las AWS CloudFormation pilas se determinan automáticamente en AWS CDK función de la ruta de construcción de la pila en el árbol. De forma predeterminada, el nombre de una pila se deriva del ID de construcción del `Stack` objeto. Sin embargo, puede especificar un nombre explícito mediante el `stackName` accesorio (en Python,`stack_name`), de la siguiente manera.

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"
});
```

Pilas anidadas

La NestedStackconstrucción ofrece una forma de sortear el límite de AWS CloudFormation 500 recursos para las pilas. Una pila anidada cuenta como un único recurso de la pila que la contiene. Sin embargo, puede contener hasta 500 recursos, incluidas pilas anidadas adicionales.

El ámbito de una pila anidada debe ser una `Stack` construcción O. `NestedStack` La pila anidada no necesita declararse léxicamente dentro de su pila principal. Solo es necesario pasar la pila principal como primer parámetro (`scope`) al crear una instancia de la pila anidada. Aparte de esta restricción, la definición de construcciones en una pila anidada funciona exactamente igual que en una pila normal.

En el momento de la síntesis, la pila anidada se sintetiza en su propia AWS CloudFormation plantilla, que se carga en el depósito provisional durante el AWS CDK despliegue. Las pilas anidadas están vinculadas a su pila principal y no se tratan como artefactos de despliegue independientes. No aparecen en la lista por `cdk list` ni se pueden implementar por. `cdk deploy`

Las referencias entre las pilas principales y las pilas anidadas se traducen automáticamente en parámetros y resultados de la pila en las AWS CloudFormation plantillas generadas, como ocurre con cualquier referencia entre pilas.

> **Warning**
>
> En el caso de las pilas anidadas, los cambios en la postura de seguridad no se muestran antes del despliegue. Esta información solo se muestra en las pilas de nivel superior.

Construcciones

Los componentes fijos son los componentes básicos de las AWS Cloud Development Kit (AWS CDK) aplicaciones. Una construcción es un componente de la aplicación que representa uno o más AWS CloudFormation recursos y su configuración. Para crear la aplicación, pieza por pieza, se importan y configuran componentes fijos.

Las construcciones son clases que se importan a las aplicaciones de CDK. Las construcciones están disponibles en la biblioteca de construcciones. AWS También puede crear y distribuir sus propios componentes fijos o utilizar componentes creados por desarrolladores externos.

Las construcciones forman parte del modelo de programación de construcciones (CPM). Están disponibles para su uso con otras herramientas, como CDK para Terraform (CDKtf), CDK para (CDK8s) y. Kubernetes Projen

Temas

AWS Construye una biblioteca

La biblioteca AWS Construct contiene una colección de construcciones desarrolladas y mantenidas por AWS. Está organizada en varios módulos que contienen componentes fijos que representan todos los recursos disponibles en ella. AWS Para obtener información de referencia, consulte la referencia de la AWS CDK API.

Se llama `aws-cdk-lib` al paquete CDK principal y contiene la mayor parte de la biblioteca AWS Construct. También contiene clases base como `Stack` y`App`.

El nombre real del paquete CDK principal varía según el idioma.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Go

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

> **Note**
>
> Si has creado un proyecto de CDK concdk init, no necesitas instalarlo manualmente. `aws-cdk-lib`

La biblioteca AWS Construct también contiene el `constructs` paquete con la clase `Construct` base. Está incluido en su propio paquete porque lo utilizan otras herramientas basadas en construcciones AWS CDK, como el CDK para Terraform y el CDK para Kubernetes.

Numerosos terceros también han publicado construcciones compatibles con. AWS CDK Visite Construct Hub para explorar el ecosistema de socios de AWS CDK construcción.

Construye niveles

Los componentes fijos de la AWS biblioteca de componentes se clasifican en tres niveles. Cada nivel ofrece un nivel de abstracción cada vez mayor. Cuanto mayor sea la abstracción, más fácil será la configuración y se requerirá menos experiencia. Cuanto menor sea la abstracción, mayor será la personalización disponible, lo que requerirá más experiencia.

Construcciones de nivel 1 (L1)

Las construcciones L1, también conocidas como recursos CFN, son las construcciones de nivel más bajo y no ofrecen abstracción. Cada construcción de L1 se asigna directamente a un único recurso. AWS CloudFormation Con las construcciones de L1, se importa una construcción que representa un recurso específico. AWS CloudFormation A continuación, defina las propiedades del recurso en la instancia de construcción.

Las construcciones L1 son ideales para utilizarlas cuando se conoce AWS CloudFormation y se necesita un control total sobre la definición de las propiedades de los AWS recursos.

En la biblioteca de AWS construcciones, las construcciones de L1 se nombran empezando por`Cfn`, seguido de un identificador para el AWS CloudFormation recurso que representan. Por ejemplo, la `CfnBucket` construcción es una construcción L1 que representa un recurso. `AWS::S3::Bucket` AWS CloudFormation

Las construcciones de L1 se generan a partir de la AWS CloudFormation especificación del recurso. Si existe un recurso en AWS CloudFormation, estará disponible AWS CDK como una construcción L1. Los nuevos recursos o propiedades pueden tardar hasta una semana en estar disponibles en la biblioteca de AWS construcciones. Para obtener más información, consulte la referencia sobre los tipos de AWS recursos y propiedades en la Guía del AWS CloudFormation usuario.

Construcciones de nivel 2 (L2)

Las construcciones L2, también conocidas como construcciones seleccionadas, son desarrolladas cuidadosamente por el equipo de CDK y, por lo general, son el tipo de construcción

más utilizado. Las construcciones L2 se asignan directamente a recursos individuales, de forma similar a las construcciones L1. AWS CloudFormation En comparación con las construcciones L1, las construcciones L2 proporcionan una abstracción de alto nivel a través de una API intuitiva basada en la intención. Las construcciones de nivel 2 incluyen configuraciones de propiedades predeterminadas y sensatas, políticas de seguridad recomendadas y generan gran parte del código repetitivo y de la lógica necesaria para el usuario.

Las construcciones de nivel 2 también proporcionan métodos auxiliares para la mayoría de los recursos, que facilitan y agilizan la definición de propiedades, permisos, interacciones entre recursos basadas en eventos, etc.

La `s3.Bucket` clase es un ejemplo de una construcción de nivel 2 para un recurso de bucket de Amazon Simple Storage Service (Amazon S3).

La biblioteca AWS Construct contiene construcciones de nivel 2 designadas como estables y listas para su uso en producción. En el caso de las construcciones de nivel 2 en desarrollo, se denominan experimentales y se ofrecen en un módulo independiente.

Construcciones de nivel 3 (L3)

Los constructos L3, también conocidos como patrones, son el nivel más alto de abstracción. Cada construcción de L3 puede contener un conjunto de recursos que están configurados para trabajar juntos a fin de realizar una tarea o un servicio específicos dentro de la aplicación. Las construcciones L3 se utilizan para crear AWS arquitecturas completas para casos de uso particulares de la aplicación.

Para proporcionar diseños de sistemas completos, o partes sustanciales de un sistema más grande, las construcciones L3 ofrecen configuraciones de propiedades predeterminadas y fundamentadas. Se basan en un enfoque particular para resolver un problema y proporcionar una solución. Con las construcciones L3, puede crear y configurar varios recursos rápidamente, con la menor cantidad de entrada y código.

La `ecsPatterns.ApplicationLoadBalancedFargateService` clase es un ejemplo de una construcción de nivel 3 que representa un AWS Fargate servicio que se ejecuta en un clúster de Amazon Elastic Container Service (Amazon ECS) y dirigido por un balanceador de carga de aplicaciones.

Al igual que las construcciones L2, las construcciones L3 que están listas para su uso en producción se incluyen en la biblioteca Construct. AWS Las que están en desarrollo se ofrecen en módulos separados.

Definir construcciones

Composición

La composición es el patrón clave para definir abstracciones de alto nivel a través de constructos. Un constructo de alto nivel puede estar compuesto por cualquier número de constructos de nivel inferior. Desde una perspectiva ascendente, se utilizan componentes fijos para organizar los AWS recursos individuales que se desean implementar. Utiliza las abstracciones que sean convenientes para su propósito, con tantos niveles como necesite.

Con la composición, defines componentes reutilizables y los compartes como cualquier otro código. Por ejemplo, un equipo puede definir una estructura que implemente las mejores prácticas de la empresa para una tabla de Amazon DynamoDB, incluidas las copias de seguridad, la replicación global, el escalado automático y la supervisión. El equipo puede compartir la construcción internamente con otros equipos o públicamente.

Los equipos pueden usar construcciones como cualquier otro paquete de biblioteca. Cuando se actualiza la biblioteca, los desarrolladores tienen acceso a las mejoras y correcciones de errores de la nueva versión, de forma similar a cualquier otra biblioteca de códigos.

Inicialización

Los constructos se implementan en clases que amplían la clase base `Construct`. Para definir una construcción, se crea una instancia de la clase. Todos los constructos toman tres parámetros cuando se inicializan:

- alcance: el padre o el propietario de la construcción. Puede ser una pila u otra construcción. El alcance determina el lugar que ocupa el componente fijo en el árbol de componentes fijos. Por lo general, se debe pasar `this` (`self`dentroPython), que representa el objeto actual, por el ámbito.
- id: un identificador que debe ser único dentro del ámbito. El identificador sirve como espacio de nombres para todo lo que se define en la construcción. Se usa para generar identificadores únicos, como nombres de recursos e AWS CloudFormation ID lógicos.

  Los identificadores solo tienen que ser únicos dentro de un ámbito. Esto le permite crear instancias de las construcciones y reutilizarlas sin preocuparse por las construcciones e identificadores que puedan contener, y permite componer las construcciones en abstracciones de nivel superior. Además, los ámbitos permiten hacer referencia a grupos de construcciones de una sola vez. Algunos ejemplos son el etiquetado o la especificación de dónde se desplegarán las construcciones.

- props: conjunto de propiedades o argumentos de palabras clave, según el idioma, que definen la configuración inicial de la construcción. Las construcciones de nivel superior proporcionan más valores predeterminados y, si todos los elementos prop son opcionales, puede omitir el parámetro props por completo.

Configuración

La mayoría de las construcciones aceptan `props` como tercer argumento (o en Python, argumentos de palabras clave), una colección de nombre/valor que define la configuración de la construcción. El siguiente ejemplo define un depósito con el cifrado AWS Key Management Service (AWS KMS) y el alojamiento de sitios web estáticos habilitados. Como no especifica explícitamente una clave de cifrado, la `Bucket` construcción define una nueva `kms.Key` y la asocia al depósito.

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"),
})
```

Interactuar con los constructos

Los constructos son clases que amplían la clase Construct base. Tras crear una instancia de una construcción, el objeto de construcción expone un conjunto de métodos y propiedades que permiten interactuar con la construcción y transmitirla como referencia a otras partes del sistema.

El AWS CDK marco no impone ninguna restricción a las API de las construcciones. Los autores pueden definir cualquier API que deseen. Sin embargo, las AWS construcciones que se incluyen en la biblioteca AWS Construct, por ejemplos`s3.Bucket`, siguen pautas y patrones comunes. Esto proporciona una experiencia coherente en todos los AWS recursos.

La mayoría de AWS las construcciones tienen un conjunto de métodos de concesión que puedes usar para conceder permisos AWS Identity and Access Management (de IAM) sobre esa construcción a un director. En el siguiente ejemplo, se concede al grupo de IAM `data-science` permiso para leer desde el bucket `raw-data` de Amazon 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)
```

Otro patrón común es que AWS las construcciones establezcan uno de los atributos del recurso a partir de datos suministrados en otros lugares. Los atributos pueden incluir nombres de recursos de Amazon (ARN), nombres o URL.

El código siguiente define una AWS Lambda función y la asocia a una cola de Amazon Simple Queue Service (Amazon SQS) a través de la URL de la cola en una variable de entorno.

TypeScript

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

```
  environment: {
    QUEUE_URL: jobsQueue.queueUrl
  }
});
```

JavaScript

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

Python

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

Java

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

C#

```
var jobsQueue = new Queue(this, "jobs");
```

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

Go

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

Para obtener información sobre los patrones de API más comunes de la biblioteca AWS Construct, consulte. the section called “Recursos”

La construcción de la aplicación y la pila

`Stack`Las clases `App` y de la biblioteca AWS Construct son construcciones únicas. En comparación con otras construcciones, no configuran AWS los recursos por sí mismas. En cambio, se utilizan para proporcionar contexto a las otras construcciones. Todos los constructos que representan AWS recursos deben definirse, directa o indirectamente, dentro del ámbito de un `Stack` constructo. `Stack`los constructos se definen dentro del ámbito de un `App` constructo.

Para obtener más información sobre las aplicaciones de CDK, consulte. AWS CDK aplicaciones Para obtener más información sobre las pilas de CDK, consulte. Pilas

En el siguiente ejemplo, se define una aplicación con una sola pila. Dentro de la pila, se utiliza una construcción L2 para configurar un recurso de bucket de 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

Pila definida en el `HelloCdkStack.java` archivo:

```
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();
    }
}
```

Aplicación definida en el `HelloCdkApp.java` archivo:

```
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
}
```

Trabajando con construcciones

Trabajando con construcciones L1

Las construcciones de L1 asignan directamente a los recursos individuales. AWS CloudFormation Debe proporcionar la configuración requerida del recurso.

En este ejemplo, creamos un `bucket` objeto mediante la construcción `CfnBucket` L1:

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

```
});
```

Go

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

Las propiedades de construcción que no sean simples booleanos, cadenas, números o contenedores se gestionan de forma diferente en los lenguajes compatibles.

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

En Python, estas propiedades se representan mediante tipos definidos como clases internas de la construcción L1. Por ejemplo, la propiedad opcional `cors_configuration` de a `CfnBucket` requiere un contenedor de tipo. `CfnBucket.CorsConfigurationProperty` Aquí estamos definiendo `cors_configuration` en una `CfnBucket` instancia.

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

Java

En Java, estas propiedades se representan mediante tipos definidos como clases internas de la construcción L1. Por ejemplo, la propiedad opcional `corsConfiguration` de a `CfnBucket` requiere un contenedor de tipo. `CfnBucket.CorsConfigurationProperty` Aquí estamos definiendo `corsConfiguration` en una `CfnBucket` instancia.

```
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#

En C#, estas propiedades se representan mediante tipos definidos como clases internas de la construcción L1. Por ejemplo, la propiedad opcional `CorsConfiguration` de a `CfnBucket` requiere un contenedor de tipo. `CfnBucket.CorsConfigurationProperty` Aquí estamos definiendo `CorsConfiguration` en una `CfnBucket` instancia.

```
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

En Go, estos tipos se nombran con el nombre de la construcción L1, un guión bajo y el nombre de la propiedad. Por ejemplo, la propiedad opcional `CorsConfiguration` de a `CfnBucket` requiere un contenedor de tipo. `CfnBucket_CorsConfigurationProperty` Aquí estamos definiendo `CorsConfiguration` en una `CfnBucket` instancia.

```
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

No puedes usar tipos de propiedades L2 con construcciones L1, o viceversa. Cuando trabaje con construcciones de L1, utilice siempre los tipos definidos para la construcción de L1 que utilice. No utilice tipos de otras construcciones de L1 (algunas pueden tener el mismo nombre, pero no son del mismo tipo).

Actualmente, algunas de nuestras referencias a la API específicas del idioma contienen errores en las rutas de los tipos de propiedades de la capa 1 o no documentan estas clases en absoluto. Esperamos solucionar este problema pronto. Mientras tanto, recuerde que estos tipos son siempre clases internas de la construcción L1 con la que se utilizan.

Trabajando con construcciones L2

En el siguiente ejemplo, definimos un bucket de Amazon S3 mediante la creación de un objeto a partir de la construcción `Bucket`L2:

TypeScript

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

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

JavaScript

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

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

Python

```
import aws_cdk.aws_s3 as s3

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

Java

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

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

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

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

C#

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

// "this" is HelloCdkStack
new Bucket(this, "MyFirstBucket", new BucketProps
{
    Versioned = true
});
```

Go

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

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

`MyFirstBucket`no es el nombre del bucket que se AWS CloudFormation crea. Es un identificador lógico que se asigna a la nueva construcción en el contexto de la aplicación CDK. El valor PhysicalName se usará para nombrar el recurso. AWS CloudFormation

Trabajar con construcciones de terceros

Construct Hub es un recurso que le ayuda a descubrir otras construcciones de terceros y de AWS la comunidad de CDK de código abierto.

Escribiendo tus propias construcciones

Además de usar las construcciones existentes, también puedes escribir las tuyas propias y permitir que cualquiera las use en sus aplicaciones. Todas las construcciones son iguales en. AWS CDK

Las construcciones de la biblioteca de AWS construcciones se tratan de la misma manera que las construcciones de una biblioteca de terceros publicadas medianteNPM,, o. Maven PyPI Los constructos publicados en el repositorio de paquetes interno de su empresa también se tratan del mismo modo.

Para declarar una nueva construcción, cree una clase que amplíe la clase base de Construct en el `constructs` paquete y, a continuación, siga el patrón de los argumentos del inicializador.

El siguiente ejemplo muestra cómo declarar una construcción que representa un bucket de Amazon S3. El bucket S3 envía una notificación del Amazon Simple Notification Service (Amazon SNS) cada vez que alguien carga un archivo en él.

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**
>
> Nuestra `NotifyingBucket` construcción no hereda de, `Bucket` sino más bien de. `Construct` Utilizamos la composición, no la herencia, para agrupar un bucket de Amazon S3 y un tema de Amazon SNS. En general, se prefiere la composición a la herencia a la hora de desarrollar AWS CDK construcciones.

El `NotifyingBucket` constructor tiene una firma de construcción típica: `scope`id, y`props`. El último argumento,`props`, es opcional (obtiene el valor predeterminado`{}`) porque todos los accesorios son opcionales. (La `Construct` clase base no acepta ningún `props` argumento). Podrías definir una instancia de esta construcción en tu aplicación sin`props`, por ejemplo:

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)
```

O puedes usar `props` (en Java, un parámetro adicional) para especificar el prefijo de ruta por el que deseas filtrar, por ejemplo:

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/"),
})
```

Por lo general, también querrá exponer algunas propiedades o métodos en sus construcciones. No es muy útil tener un tema oculto detrás de la construcción, ya que los usuarios de la construcción no

pueden suscribirse a ella. Añadir una `topic` propiedad permite a los consumidores acceder al tema interno, como se muestra en el siguiente ejemplo:

TypeScript

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

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

JavaScript

```
class NotifyingBucket extends Construct {

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

module.exports = { NotifyingBucket };
```

Python

```
class NotifyingBucket(Construct):

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

Java

```
public class NotifyingBucket extends Construct {

    public Topic topic = null;

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

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

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

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

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

C#

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

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

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

Go

Para hacer esto en Go, necesitaremos un poco más de fontanería. Nuestra `NewNotifyingBucket` función original devolvió un`awss3.Bucket`. Tendremos que ampliarla `Bucket` para incluir un `topic` miembro mediante la creación de una `NotifyingBucket` estructura. Nuestra función devolverá entonces este 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
}
```

Ahora, los consumidores pueden suscribirse al tema, por ejemplo:

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))
```

Más información

El siguiente vídeo ofrece una visión general completa de las estructuras de CDK y explica cómo utilizarlas en sus aplicaciones de CDK.

Explicación de los constructos de CDK

Entornos

El objetivo Cuenta de AWS es un entorno en el Región de AWS que se implementan las pilas. Todas las pilas de tu aplicación de CDK están asociadas explícita o implícitamente a un entorno (). `env`

Temas

- Configuración de entornos
- Entornos de arranque

Configuración de entornos

En el caso de las pilas de producción, te recomendamos que especifiques de forma explícita el entorno de cada pila de tu aplicación mediante la propiedad. `env` En el siguiente ejemplo, se especifican entornos diferentes para sus dos pilas diferentes.

TypeScript

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

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

JavaScript

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

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

Python

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

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

Java

```
public class MyApp {

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

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

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

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

        app.synth();
    }
}
```

C#

```
Amazon.CDK.Environment makeEnv(string account, string region)
{
```

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

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

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

Al codificar de forma rígida la cuenta y la región de destino, como se muestra en el ejemplo anterior, la pila siempre se despliega en esa cuenta y región específicas. Para hacer que la pila se pueda implementar en un objetivo diferente, pero para determinar el objetivo en el momento de la síntesis, la pila puede usar dos variables de entorno proporcionadas por la AWS CDK CLI: `CDK_DEFAULT_ACCOUNT` y`CDK_DEFAULT_REGION`. Estas variables se establecen en función del AWS perfil especificado mediante la --profile opción, o del AWS perfil predeterminado si no se especifica ninguno.

El siguiente fragmento de código muestra cómo acceder a la cuenta y la región transferidas desde la AWS CDK CLI de su pila.

TypeScript

Acceda a las variables de entorno a través del `process` objeto de Node.

> **Note**
>
> Necesitas el `DefinitelyTyped` módulo para usarlo `process` TypeScript. `cdk init`instala este módulo por ti. Sin embargo, debe instalar este módulo manualmente si está trabajando con un proyecto creado antes de agregarlo o si no configuró su proyecto utilizando`cdk init`.
>
> ```
> npm install @types/node
> ```

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

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

JavaScript

Acceda a las variables de entorno a través del `process` objeto de Node.

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

Python

Utilice el `environ` diccionario del `os` módulo para acceder a las variables de entorno.

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

Java

Se utiliza `System.getenv()` para obtener el valor de una variable de entorno.

```
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#

Se utiliza `System.Environment.GetEnvironmentVariable()` para obtener el valor de una variable de entorno.

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

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

Especifique el Región de AWS uso de un código de región. Para ver una lista, consulte Puntos finales regionales.

Esto AWS CDK distingue entre no especificar la `env` propiedad en absoluto y especificarla mediante `CDK_DEFAULT_ACCOUNT` y. `CDK_DEFAULT_REGION` La primera implica que la pila debe sintetizar una plantilla independiente del entorno. Las construcciones que se definen en una pila de este tipo no pueden utilizar ninguna información sobre su entorno. Por ejemplo, no puedes escribir código `if (stack.region === 'us-east-1')` ni usar funciones de framework como vpc.FromLookup

(Python:`from_lookup`), que necesitan consultar tu cuenta. AWS Estas funciones no funcionan en absoluto hasta que especifiques un entorno explícito; para usarlas, debes especificarlo. `env`

Cuando pase a su entorno mediante `CDK_DEFAULT_ACCOUNT` y`CDK_DEFAULT_REGION`, la pila se implementará en la cuenta y la región determinadas por la AWS CDK CLI en el momento de la síntesis. Esto permite que el código que depende del entorno funcione, pero también significa que la plantilla sintetizada podría ser diferente en función de la máquina, el usuario o la sesión en la que se sintetiza. Este comportamiento suele ser aceptable o incluso deseable durante el desarrollo, pero probablemente sería un antipatrón para su uso en producción.

Puede configurarlo `env` como desee, utilizando cualquier expresión válida. Por ejemplo, puede escribir su pila para que admita dos variables de entorno adicionales que le permitan anular la cuenta y la región en el momento de la síntesis. Las llamaremos `CDK_DEPLOY_ACCOUNT` y `CDK_DEPLOY_REGION` aquí, pero puedes ponerles el nombre que quieras, ya que no están definidas por. AWS CDK En el entorno de la siguiente pila, se utilizan variables de entorno alternativas si están configuradas. Si no están configuradas, vuelven al entorno predeterminado proporcionado por 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 el entorno de tu pila declarado de esta manera, puedes escribir un script corto o un archivo por lotes como el siguiente para establecer las variables a partir de los argumentos de la línea de comandos y, a continuación, llamar`cdk deploy`. Los argumentos que no estén entre los dos primeros se pasan a las opciones o pilas de la línea de comandos `cdk deploy` y se pueden usar para especificarlos.

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
```

Guarde el script como y`cdk-deploy-to.sh`, a continuación, ejecútelo `chmod +x cdk-deploy-to.sh` para hacerlo ejecutable.

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 versión para Windows del script suele PowerShell proporcionar la misma funcionalidad que la versión para macOS/Linux. También contiene instrucciones que permiten que se ejecute como un archivo por lotes para que pueda invocarse fácilmente desde una línea de comandos. Debe guardarse como`cdk-deploy-to.bat`. El archivo se `cdk-deploy-to.ps1` creará cuando se invoque el archivo por lotes.

Luego, puede escribir scripts adicionales que se denominen «deploy-to» para implementarlos en entornos específicos (incluso en varios entornos por script):

macOS/Linux

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

Windows

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

Al realizar la implementación en varios entornos, considere si desea continuar con la implementación en otros entornos después de que una implementación falle. El siguiente ejemplo evita la implementación en el segundo entorno de producción si el primero no se realiza correctamente.

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 %*
```

Los desarrolladores podrían seguir utilizando el `cdk deploy` comando normal para realizar la implementación en sus propios AWS entornos de desarrollo.

Si no especificas un entorno al crear una instancia de una pila, se dice que la pila es independiente del entorno. AWS CloudFormation las plantillas sintetizadas a partir de una pila de este tipo intentarán utilizar la resolución en el momento de la implementación en atributos relacionados con el entorno`stack.account`, como`stack.region`, y (`stack.availabilityZones`Python:`availability_zones`).

Cuando se utilizan cdk deploy para implementar pilas independientes del entorno, utilizan el AWS CLI perfil especificado para determinar AWS CDK CLI dónde se van a implementar. Si no se especifica ningún perfil, se utiliza el perfil predeterminado. A AWS CDK CLI continuación, se presenta un protocolo similar AWS CLI al utilizado para determinar qué AWS credenciales utilizar al realizar operaciones en la AWS cuenta. Para obtener más información, consulte the section called “AWS CDK Kit de herramientas”.

En una pila independiente del entorno, cualquier construcción que utilice zonas de disponibilidad verá dos zonas de disponibilidad, lo que permitirá implementar la pila en cualquier región.

Entornos de arranque

Debe arrancar cada entorno en el que vaya a implementar las pilas de CDK. El arranque prepara el entorno para la implementación. Para obtener más información, consulte Bootstrapping (Proceso de arranque).

Bootstrapping (Proceso de arranque)

El arranque es el proceso de preparar un entorno para la implementación. El arranque es una acción que se realiza una sola vez y que se debe realizar en cada entorno en el que se desplieguen los recursos.

Temas

Entornos de arranque

> **Important**
>
> Es posible que se le cobren AWS gastos por los datos almacenados en los recursos que se están iniciando.

Bootstrapping aprovisiona recursos en su entorno, como un depósito de Amazon Simple Storage Service (Amazon S3) para almacenar archivos y funciones AWS Identity and Access Management (IAM) que otorgan los permisos necesarios para realizar las implementaciones. Estos recursos se aprovisionan en una AWS CloudFormation pila, denominada pila bootstrap. Suele tener un nombre. `CDKToolkit` Como cualquier AWS CloudFormation pila, aparecerá en la AWS CloudFormation consola de su entorno una vez que se haya implementado.

> **Note**
>
> CDK v2 utiliza una plantilla de bootstrap moderna. La plantilla antigua del CDK v1 no es compatible con la versión 2.

Los entornos son independientes. Si desea realizar la implementación en varios entornos, cada entorno debe iniciarse por separado.

Si intenta implementar una aplicación de CDK en un entorno que no se ha iniciado, recibirá un mensaje de error que le recordará que debe iniciar el entorno.

Puesta en marcha con CDK Pipelines

Si utilizas CDK Pipelines para realizar la implementación en el entorno de otra cuenta y recibes un mensaje como el siguiente:

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

Este mensaje de error significa que las funciones de IAM adecuadas no existen en el otro entorno. La causa más probable es que el entorno no se haya iniciado. Inicie el entorno e inténtelo de nuevo.

> **Note**
>
> Si el entorno está arrancado, no elimine ni vuelva a crear la pila de arranque del entorno. Al eliminar la pila de arranque, se eliminarán los AWS recursos que se aprovisionaron originalmente en el entorno para respaldar las implementaciones de CDK. Esto hará que la canalización deje de funcionar. En su lugar, intenta actualizar la pila de bootstrap a una nueva versión ejecutando de nuevo el CLI `cdk bootstrap` comando CDK.

¿Cómo arrancar

Al arrancar un entorno, se despliega una AWS CloudFormation plantilla en el entorno específico. Esta plantilla aprovisiona recursos en su cuenta para preparar el entorno para la implementación.

La plantilla de arranque acepta parámetros que personalizan algunos aspectos de los recursos de arranque. Para obtener más información, consulte the section called “Personalización del bootstrapping”.

Puede realizar el arranque de cualquiera de las siguientes maneras:

- Use el comando AWS CDK CLI.cdk bootstrap Este es el método más simple y funciona bien si solo tiene unos pocos entornos para arrancar.
- Implemente la plantilla proporcionada por el AWS CDK CLI uso de otra herramienta AWS CloudFormation de despliegue. Esto le permite usar AWS CloudFormation StackSets o AWS Control Tower y también la AWS CloudFormation consola o el AWS CLI. Puede realizar pequeñas modificaciones en la plantilla antes de la implementación. Este enfoque es más flexible y adecuado para despliegues a gran escala.

No es un error iniciar un entorno más de una vez. Si un entorno al que se está iniciando ya se ha iniciado, su pila de arranque se actualizará si es necesario. De lo contrario, no pasará nada.

Empezando con el AWS CDKCLI

Utilice el `cdk bootstrap` comando para arrancar uno o más entornos. AWS

El siguiente ejemplo arranca dos entornos:

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

Los siguientes ejemplos muestran varias formas de arrancar entornos. Como se muestra en el segundo ejemplo, el `aws://` prefijo es opcional cuando se especifica un entorno.

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

Cuando se ejecutacdk bootstrap, la CDK CLI siempre sintetiza la aplicación CDK en el directorio actual. Si no especificas al menos un entorno, la CDK CLI iniciará todos los entornos a los que se haga referencia en la aplicación.

En el caso de las pilas independientes del entorno, la CDK CLI intentará determinar un entorno a partir de las fuentes predeterminadas. Puede ser un entorno especificado mediante la --profile opción, a partir de variables de entorno o de fuentes predeterminadas. AWS CLI Si se encuentra, se arranca el entorno.

Por ejemplo, el siguiente comando sintetiza la AWS CDK aplicación actual mediante el `prod` AWS perfil y, a continuación, arranca sus entornos.

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

Arrancar desde la plantilla AWS CloudFormation

Puede arrancar un entorno obteniendo e implementando la plantilla de arranque. AWS CloudFormation

Para obtener una copia de esta plantilla en el archivo`bootstrap-template.yaml`, ejecute el siguiente comando:

macOS/Linux

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

Windows

En Windows, PowerShell debe usarse para conservar la codificación de la plantilla.

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

La plantilla también está disponible en el AWS CDK GitHub repositorio.

Implemente esta plantilla mediante la CLI de CDK o su mecanismo de implementación preferido para AWS CloudFormation las plantillas. Para realizar la implementación mediante la CLI de CDK, ejecutecdk bootstrap --template ***TEMPLATE_FILENAME***. También puede implementarlo AWS CLI mediante la ejecución del siguiente comando o implementarlo en una o más cuentas a la vez mediante AWS CloudFormation Stack Sets.

macOS/Linux

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

Windows

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

Personalización del bootstrapping

Hay dos formas de personalizar el arranque de los recursos de su entorno:

- Utilice los parámetros de la línea de comandos con el comando. `cdk bootstrap` Esto le permite modificar algunos aspectos de la plantilla.
- Modifique la plantilla de bootstrap predeterminada e impleméntela usted mismo. Esto le da un control más completo sobre los recursos de bootstrap.

Las siguientes opciones de línea de comandos, cuando se utilizan con CDK CLIcdk bootstrap, proporcionan los ajustes más utilizados en la plantilla de arranque:

- - -bootstrap-bucket-name anula el nombre del bucket de Amazon S3. Es posible que sea necesario realizar cambios en la aplicación CDK (consultethe section called “Apile sintetizadores”).
- --bootstrap-kms-key-idanula la AWS KMS clave utilizada para cifrar el depósito de S3.
- --cloudformation-execution-policiesespecifica los ARN de las políticas gestionadas que se deben adjuntar a la función de despliegue que se asume AWS CloudFormation durante el despliegue de las pilas. De forma predeterminada, las pilas se implementan con todos los permisos de administrador mediante la política. `AdministratorAccess`

  Los ARN de la política se deben pasar como un argumento de cadena único, con los ARN individuales separados por comas. Por ejemplo:

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

  > **Important**
  >
  > Para evitar errores en la implementación, asegúrese de que las políticas que especifique sean suficientes para cualquier implementación que vaya a realizar en el entorno que se está iniciando.

- --qualifieres una cadena que se añade a los nombres de todos los recursos de la pila de arranque. Un calificador permite evitar conflictos de nombres de recursos al aprovisionar varias pilas de bootstrap en el mismo entorno. El valor predeterminado es `hnb659fds` (este valor no tiene importancia).

  Para cambiar el calificador, también es necesario que la aplicación CDK pase el valor modificado al sintetizador de pilas. Para obtener más información, consulte the section called “Apile sintetizadores”.
- --tagsañade una o más AWS CloudFormation etiquetas a la pila de bootstrap.

- --trustenumera las AWS cuentas que pueden desplegarse en el entorno que se está iniciando.

  Utilice este indicador al iniciar un entorno en el que se implementará una canalización de CDK en otro entorno. Siempre se confía en la cuenta que realiza el arranque.

- --trust-for-lookupmuestra las AWS cuentas que pueden buscar información contextual del entorno que se está iniciando.

  Use este indicador para dar permiso a las cuentas para sintetizar las pilas que se van a implementar en el entorno, sin darles permiso para implementar esas pilas directamente.

- --termination-protectionimpide que se elimine la pila de arranque. Para obtener más información, consulte Cómo proteger una pila para que no se elimine en la Guía del AWS CloudFormation usuario.

> **Important**
>
> La plantilla bootstrap moderna concede de forma efectiva los permisos implícitos `--cloudformation-execution-policies` a cualquier AWS cuenta de la `--trust` lista. De forma predeterminada, esto amplía los permisos de lectura y escritura en cualquier recurso de la cuenta de arranque. Asegúrese de configurar la pila de arranque con políticas y cuentas de confianza con las que se sienta cómodo.

Personalización de la plantilla

Cuando necesite más personalización de la que CLI puede ofrecer el CDK, puede modificar la plantilla de bootstrap para adaptarla a sus necesidades. En primer lugar, se obtiene la plantilla mediante la --show-template opción. A continuación, se muestra un ejemplo:

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

Cualquier modificación que realice debe cumplir con el contrato de plantilla inicial. Para asegurarse de que alguien que cdk bootstrap utilice la plantilla predeterminada no sobrescriba accidentalmente sus personalizaciones más adelante, cambie el valor predeterminado del parámetro de la plantilla. `BootstrapVariant` La CLI de CDK solo permitirá sobrescribir la pila de arranque con plantillas que tengan la misma `BootstrapVariant` versión o una versión igual o superior a la de la plantilla que está implementada actualmente.

A continuación, puede implementar la plantilla modificada tal y como se describe en the section called “Arrancar desde la plantilla AWS CloudFormation” o utilizando. cdk bootstrap --template

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

Arrancar las diferencias entre plantillas

Como se mencionó anteriormente, la AWS CDK versión 1 admitía dos plantillas de arranque, antiguas y modernas. CDK v2 solo admite la plantilla moderna. Como referencia, estas son las diferencias de alto nivel entre estas dos plantillas.

Característica	Legacy (solo en la versión 1)	Moderno (v1 y v2)
Implementaciones entre cuentas	No permitido	Permitida
AWS CloudFormation Permisos	Se implementa con los permisos del usuario actual (determinados por el AWS perfil, las variables de entorno, etc.)	Se despliega con los permisos especificados cuando se aprovisionó la pila de arranque (por ejemplo, mediante el uso) `--trust`
Control de versiones	Solo hay disponible una versión de la pila de bootstrap	La pila de Bootstrap está versionada; se pueden agregar nuevos recursos en futuras versiones y las AWS CDK aplicaciones pueden requerir una versión mínima
Recursos *	Bucket de Amazon S3	Bucket de Amazon S3
		AWS KMS key
		Roles de IAM
		Repositorio Amazon ECR
		Parámetro SSM para el control de versiones

Característica	Legacy (solo en la versión 1)	Moderno (v1 y v2)
Denominación de recursos	Generado automáticamente	Determinista
Cifrado por cubos	Clave predeterminada	Clave administrada por clientes

* Agregaremos recursos adicionales a la plantilla de bootstrap según sea necesario.

Un entorno que se inició con la plantilla anterior debe actualizarse para utilizar la plantilla moderna de CDK v2 mediante el reinicio. Vuelva a implementar todas las AWS CDK aplicaciones del entorno al menos una vez antes de eliminar el bucket antiguo.

Apile sintetizadores

Tu AWS CDK aplicación necesita conocer los recursos de arranque disponibles para poder sintetizar correctamente una pila que se pueda implementar. El sintetizador de pilas es una AWS CDK clase que controla cómo se sintetiza la plantilla de la pila. Esto incluye la forma en que utiliza los recursos de arranque (por ejemplo, cómo se refiere a los activos almacenados en el depósito de arranque).

Los sintetizadores AWS CDK de pila integrados se llaman. `DefaultStackSynthesizer` Incluye capacidades para despliegues entre cuentas y despliegues de CDK Pipelines.

Puede pasar un sintetizador de pilas a una pila al instanciarla mediante la propiedad. `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
    })
});
```

Si no proporciona la `synthesizer` propiedad, se usa. `DefaultStackSynthesizer`

Personalización de la síntesis

En función de los cambios que haya realizado en la plantilla de bootstrap, es posible que también necesite personalizar la síntesis. Se `DefaultStackSynthesizer` puede personalizar con las propiedades que se describen a continuación.

Si ninguna de estas propiedades proporciona las personalizaciones que necesita, puede escribir el sintetizador como una clase que se implemente `IStackSynthesizer` (tal vez derivada de). `DefaultStackSynthesizer`

Cambiar el calificador

El calificador se añade al nombre de los recursos de arranque para distinguir los recursos en pilas de arranque independientes. Para implementar dos versiones diferentes de la pila de arranque en el mismo entorno (AWS cuenta y región), las pilas deben tener calificadores diferentes.

Esta función está diseñada para aislar los nombres entre las pruebas automatizadas del propio CDK. A menos que se puedan delimitar con precisión los permisos de IAM otorgados a la función de AWS CloudFormation ejecución, tener dos pilas de bootstrap diferentes en una sola cuenta no supone ninguna ventaja en cuanto al aislamiento de permisos. Por lo tanto, no suele ser necesario cambiar este valor.

Para cambiar el calificador, configure `DefaultStackSynthesizer` cualquiera de los dos creando una instancia del sintetizador con la propiedad:

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"
    })
});
```

O configurando el calificador como clave de contexto. `cdk.json`

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

Cambiando los nombres de los recursos

Todas las demás `DefaultStackSynthesizer` propiedades se refieren a los nombres de los recursos de la plantilla de arranque. Solo necesita proporcionar alguna de estas propiedades si ha modificado la plantilla de arranque y ha cambiado los nombres de los recursos o el esquema de nomenclatura.

Todas las propiedades aceptan los marcadores de posición especiales`${Qualifier}`, `${AWS::Partition}${AWS::AccountId}`, y. `${AWS::Region}` Estos marcadores de posición se sustituyen por los valores del `qualifier` parámetro y los valores de AWS partición, ID de cuenta y región del entorno de la pila, respectivamente.

En el siguiente ejemplo, se muestran las propiedades más utilizadas `DefaultStackSynthesizer` junto con sus valores predeterminados, como si se estuviera creando una instancia del sintetizador. Para ver una lista completa, consulte DefaultStackSynthesizerProps.

TypeScript

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

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

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

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

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

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

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

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

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

})
```

JavaScript

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

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

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

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

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

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

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

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

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

Python

```
DefaultStackSynthesizer(
  # Name of the S3 bucket for file assets
  file_assets_bucket_name="cdk-${Qualifier}-assets-${AWS::AccountId}-
${AWS::Region}",
  bucket_prefix="",

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

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

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

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

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

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

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

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

Java

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

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

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

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

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

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

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

    // Name of the SSM parameter which describes the bootstrap stack version number
```

```
    .bootstrapStackVersionSsmParameter("/cdk-bootstrap/${Qualifier}/version")

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

C#

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

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

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

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

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

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

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

```
    LookupRoleExternalId = "",

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

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

El contrato de plantilla de arranque

Los requisitos de la pila de arranque dependen del sintetizador de pila que se utilice. Si escribes tu propio sintetizador de pilas, tienes el control total de los recursos de arranque que requiere tu sintetizador y de la forma en que el sintetizador los encuentra.

En esta sección se describen las expectativas que tiene de la plantilla de arranque. `DefaultStackSynthesizer`

Control de versiones

La plantilla debe contener un recurso para crear un parámetro SSM con un nombre conocido y un resultado que refleje la versión de la plantilla.

```
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`Requiere cinco funciones de IAM para cinco propósitos diferentes. Si no utiliza las funciones predeterminadas, debe indicar al sintetizador los ARN de las funciones que desea utilizar.

Los roles son los siguientes:

- La función de despliegue la asumen el AWS CDK kit de herramientas y el AWS CodePipeline despliegue en un entorno. `AssumeRolePolicy`Controla quién puede realizar el despliegue en el entorno. En la plantilla, puede ver los permisos que necesita este rol.
- El AWS CDK kit de herramientas asume la función de búsqueda para realizar búsquedas de contexto en un entorno. `AssumeRolePolicy`Controla quién puede desplegarse en el entorno. Los permisos que necesita este rol se pueden ver en la plantilla.
- El AWS CDK kit de herramientas y los AWS CodeBuild proyectos asumen la función de publicación de archivos y la función de publicación de imágenes para publicar activos en un entorno. Se utilizan para escribir en el bucket de S3 y en el repositorio de ECR, respectivamente. Estas funciones requieren acceso de escritura a estos recursos.
- La función AWS CloudFormation de ejecución se transfiere AWS CloudFormation a la función de ejecución para realizar la implementación real. Sus permisos son los permisos con los que se ejecutará la implementación. Los permisos se transfieren a la pila como un parámetro que enumera los ARN de las políticas administradas.

Salidas

El AWS CDK kit de herramientas requiere que existan los siguientes CloudFormation resultados en la pila de arranque.

- `BucketName`: el nombre del depósito de activos del archivo
- `BucketDomainName`: el depósito de activos de archivos en formato de nombre de dominio
- `BootstrapVersion`: la versión actual de la pila de bootstrap

Historial de plantillas

La plantilla bootstrap está versionada y evoluciona con el tiempo junto con ella misma. AWS CDK Si proporciona su propia plantilla de bootstrap, manténgala actualizada con la plantilla canónica

predeterminada. Quieres asegurarte de que tu plantilla siga funcionando con todas las funciones de CDK.

> **Note**
>
> Las versiones anteriores de la plantilla de arranque creaban un entorno AWS KMS key en cada entorno de arranque de forma predeterminada. Para evitar que se le cobre por la clave KMS, reinicie estos entornos utilizando. `--no-bootstrap-customer-key` El valor predeterminado actual es no tener una clave KMS, lo que ayuda a evitar estos cargos.

Esta sección contiene una lista de los cambios realizados en cada versión.

Versión de plantilla	AWS CDK versión	Cambios
1	1.40.0	Versión inicial de la plantilla con depósito, clave, repositor io y funciones.
2	1.45.0	Divida la función de publicaci ón de activos en funciones independientes de publicación de archivos e imágenes.
3	1.46.0	Añada la opción de `FileAssetKeyArn` exportación para poder añadir permisos de descifrado a los consumidores de activos.
4	1.61.0	AWS KMS los permisos ahora están implícitos a través de Amazon S3 y ya no son necesarios`FileAsetK eyArn` . Agregue `CdkBootst rapVersion` el parámetro SSM para que la versión de la pila de arranque se pueda

Versión de plantilla	AWS CDK versión	Cambios
		verificar sin conocer el nombre de la pila.
5	1.87.0	El rol de despliegue puede leer el parámetro SSM.
6	1.108.0	Añada una función de búsqueda independiente de la función de despliegue.
6	1.109.0	Adjunte una `aws-cdk:bootstrap-role` etiqueta a las funciones de despliegue, publicación de archivos y publicación de imágenes.
7	1.110.0	La función de despliegue ya no puede leer directamente los buckets de la cuenta de destino. (Sin embargo, este rol es, en efecto, el de administrador y, de todos modos, siempre puede usar sus AWS CloudFormation permisos para hacer que el bucket sea legible).
8	1.114.0	La función de búsqueda tiene permisos completos de solo lectura para el entorno de destino y también tiene una `aws-cdk:bootstrap-role` etiqueta.

Versión de plantilla	AWS CDK versión	Cambios
9	2.1.0	Corrige el rechazo de las cargas de activos de Amazon S3 por parte del SCP de cifrado al que se hace referencia habitualmente.
10	2.4.0	Amazon ECR ahora ScanOnPush está activado de forma predeterminada.
11	2.18.0	Añade una política que permite a Lambda extraer datos de los repositorios de Amazon ECR para que sobreviva al reinicio.
12	2.20.0	Añade soporte para experimentoscdk import.
13	2.25.0	Hace que las imágenes de los contenedores de los repositorios Amazon ECR creados por bootstrap sean inmutables.
14	2.34.0	Desactiva de forma predeterminada el escaneo de imágenes de Amazon ECR en el nivel del repositorio para permitir el arranque de regiones que no admiten el escaneo de imágenes.
15	2.60,0	Las claves KMS no se pueden etiquetar.

Versión de plantilla	AWS CDK versión	Cambios
16	2.69,0	Aborda que Security Hub encuentra KMS.2.
17	2.72.0	Aborda la búsqueda de ECR.3 por parte de Security Hub.
18	2.80,0	Se han revertido los cambios realizados en la versión 16, ya que no funcionan en todas las particiones y no se recomiend an.
19	2.106.1	Se han revertido los cambios realizados en la versión 18, en los que se eliminaba la AccessControl propiedad de la plantilla. (#27964)
20	2.119.0	Añada `ssm:GetPa rameters` una acción a la función de AWS CloudForm ation implementación de IAM. Para obtener más información, consulte #28336.

Hallazgos de Security Hub

Si lo está utilizando AWS Security Hub, es posible que vea los resultados de algunos de los recursos creados por el proceso de AWS CDK Bootstrapping. Los resultados de Security Hub le ayudan a encontrar configuraciones de recursos que debe comprobar para garantizar su precisión y seguridad. Hemos revisado estas configuraciones de recursos específicas con AWS Security y estamos seguros de que no constituyen un problema de seguridad.

[KMS.2] Los directores de IAM no deberían tener políticas integradas de IAM que permitan realizar acciones de descifrado en todas las claves de KMS

El rol de implementación (nombre predeterminado`cdk-hnb659fds-deploy-role-ACCOUNT-REGION`) tiene permisos para leer los datos cifrados almacenados en Amazon S3. La política no autoriza ningún dato por sí sola: solo se pueden descifrar los datos leídos desde Amazon S3 y solo desde los buckets que permiten explícitamente al rol de despliegue leer desde ellos a través de su política de bucket, y las claves que permiten explícitamente al rol de despliegue descifrar mediante su política de claves. Esta declaración se utiliza para permitir a AWS CDK Pipelines realizar despliegues entre cuentas.

¿Por qué Security Hub marca esto? La política contiene una `Condition` cláusula `Resource: *` combinada con una; Security Hub marca la`*`. Esto `*` es necesario porque, en el momento en que se inicia la cuenta, la AWS KMS clave creada por AWS CDK Pipelines para CodePipeline Artifact Bucket aún no existe, por lo que no podemos hacer referencia a su ARN. Además, Security Hub no incluye la `Condition` cláusula de la declaración de política en su razonamiento.

¿Qué sucede si quiero corregir este hallazgo? Siempre que las políticas de recursos de sus AWS KMS claves no sean innecesariamente permisivas, la política de roles actual no permite que el rol de implementación acceda a más datos de los que debería. Si aún así quieres deshacerte de este hallazgo, puedes hacerlo personalizando la pila de bootstrap (mediante el proceso descrito anteriormente) de una de estas dos maneras:

- Si no utilizas AWS CDK Pipelines para despliegues entre cuentas: elimina la sentencia de la función `Sid: PipelineCrossAccountArtifactsBucket` de despliegue; o
- Si utilizas AWS CDK Pipelines para despliegues entre cuentas: después de implementar tu AWS CDK Pipeline, busca el AWS KMS ARN clave del Artifact Bucket y sustituye el de la `Sid: PipelineCrossAccountArtifactsBucket` declaración por `Resource: *` el ARN clave real.

Recursos

Los recursos son lo que se configura para usar Servicios de AWS en las aplicaciones. Los recursos son una característica de AWS CloudFormation. Al configurar los recursos y sus propiedades en una AWS CloudFormation plantilla, puede implementarlos AWS CloudFormation para aprovisionar sus recursos. Con él AWS Cloud Development Kit (AWS CDK), puede configurar los recursos mediante componentes fijos. A continuación, debe implementar su aplicación CDK, lo que implica sintetizar

una AWS CloudFormation plantilla e implementarla para AWS CloudFormation aprovisionar sus recursos.

Temas

Configurar los recursos mediante componentes

Como se describe enthe section called “Construcciones”, AWS CDK proporciona una amplia biblioteca de clases de construcciones, denominadas AWS construcciones, que representan todos los recursos. AWS

Para crear una instancia de un recurso utilizando su construcción correspondiente, introduzca el ámbito como primer argumento, el identificador lógico de la construcción y un conjunto de propiedades de configuración (props). Por ejemplo, aquí se explica cómo crear una cola de Amazon SQS con AWS KMS cifrado mediante la construcción SQS.Queue de la biblioteca Construct. 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
});
```

Algunos accesorios de configuración son opcionales y, en muchos casos, tienen valores predeterminados. En algunos casos, todos los accesorios son opcionales y el último argumento se puede omitir por completo.

Atributos de recursos

La mayoría de los recursos de la biblioteca AWS Construct muestran los atributos, que se resuelven en el momento del despliegue mediante. AWS CloudFormation Los atributos se exponen en forma de propiedades en las clases de recursos con el nombre del tipo como prefijo. El siguiente ejemplo muestra cómo obtener la URL de una cola de Amazon SQS mediante la propiedad (`queueUrl`Python:`queue_url`).

TypeScript

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

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

JavaScript

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

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

Python

```
import aws_cdk.aws_sqs as sqs

queue = sqs.Queue(self, "MyQueue")
url = queue.queue_url # => A string representing a deploy-time value
```

Java

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

C#

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

Consulte the section called “Tokens” para obtener información sobre cómo AWS CDK codifica los atributos de tiempo de despliegue como cadenas.

Hacer referencia a los recursos

Al configurar los recursos, a menudo tendrá que hacer referencia a las propiedades de otro recurso. A continuación se muestran algunos ejemplos:

- Un recurso de Amazon Elastic Container Service (Amazon ECS) requiere una referencia al clúster en el que se ejecuta.
- Una CloudFront distribución de Amazon requiere una referencia al bucket de Amazon Simple Storage Service (Amazon S3) que contiene el código fuente.

Puede hacer referencia a los recursos de cualquiera de las siguientes maneras:

- Transfiriendo un recurso definido en tu aplicación de CDK, ya sea en la misma pila o en una diferente
- Al pasar un objeto proxy que hace referencia a un recurso definido en su AWS cuenta, creado a partir de un identificador único del recurso (como un ARN)

Si la propiedad de una construcción representa una construcción de otro recurso, su tipo es el del tipo de interfaz de la construcción. Por ejemplo, la construcción Amazon ECS toma una propiedad `cluster` de tipo`ecs.ICluster`. Otro ejemplo es la construcción CloudFront de distribución que toma una propiedad `sourceBucket` (Python:`source_bucket`) de tipos`s3.IBucket`.

Puedes pasar directamente cualquier objeto de recurso del tipo adecuado definido en la misma AWS CDK aplicación. El siguiente ejemplo define un clúster de Amazon ECS y, a continuación, lo utiliza para definir un servicio de 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 });
```

Hacer referencia a los recursos de una pila diferente

Puedes hacer referencia a los recursos de una pila diferente siempre que estén definidos en la misma aplicación y en el mismo AWS entorno. Por lo general, se utiliza el siguiente patrón:

- Guarde una referencia a la construcción como atributo de la pila que produce el recurso. (Para obtener una referencia a la pila de la construcción actual, utilice`Stack.of(this)`.)
- Pase esta referencia al constructor de la pila que consume el recurso como parámetro o propiedad. Luego, la pila consumidora la pasa como una propiedad a cualquier construcción que la necesite.

El siguiente ejemplo define una pila`stack1`. Esta pila define un bucket de Amazon S3 y almacena una referencia a la construcción del bucket como atributo de la pila. A continuación, la aplicación define una segunda pila`stack2`, que acepta un bucket en la instanciación. `stack2`podría, por ejemplo, definir una AWS Glue tabla que utilice el depósito para el almacenamiento de datos.

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});
```

Si AWS CDK determina que el recurso se encuentra en el mismo entorno, pero en una pila diferente, sintetiza automáticamente AWS CloudFormation las exportaciones en la pila de producción y un Fn:: ImportValue en la pila consumidora para transferir esa información de una pila a otra.

Resolver los puntos muertos de dependencia

Al hacer referencia a un recurso de una pila en una pila diferente, se crea una dependencia entre las dos pilas. Esto garantiza que se desplieguen en el orden correcto. Una vez desplegadas las pilas, esta dependencia es concreta. Después de eso, eliminar el uso del recurso compartido de la pila que lo consume puede provocar un error de implementación inesperado. Esto ocurre si existe otra dependencia entre las dos pilas que obligue a desplegarlas en el mismo orden. También puede ocurrir sin una dependencia si el kit de herramientas del CDK simplemente elige la pila de producción para implementarla primero. La AWS CloudFormation exportación se elimina de la pila de producción porque ya no es necesaria, pero el recurso exportado se sigue utilizando en la pila consumidora porque su actualización aún no se ha implementado. Por lo tanto, se produce un error al implementar la pila de productores.

Para salir de este punto muerto, elimine el uso del recurso compartido de la pila que lo consume. (Esto elimina la exportación automática de la pila de producción). A continuación, añada manualmente la misma exportación a la pila de producción utilizando exactamente el mismo ID lógico que la exportación generada automáticamente. Elimine el uso del recurso compartido en la pila consumidora e implemente ambas pilas. A continuación, elimina la exportación manual (y el recurso compartido si ya no es necesario) y vuelve a implementar ambas pilas. El `exportValue()` método

de la pila es una forma cómoda de crear la exportación manual para este propósito. (Consulte el ejemplo en la referencia del método vinculado).

Hacer referencia a los recursos de tu cuenta AWS

Supongamos que quieres usar un recurso que ya está disponible en tu AWS cuenta en tu AWS CDK aplicación. Puede ser un recurso que se definió a través de la consola, un AWS SDK, directamente con AWS CloudFormation o en una AWS CDK aplicación diferente. Puede convertir el ARN del recurso (u otro atributo de identificación o grupo de atributos) en un objeto proxy. El objeto proxy sirve como referencia al recurso al llamar a un método de fábrica estático de la clase del recurso.

Al crear un proxy de este tipo, el recurso externo no pasa a formar parte de la AWS CDK aplicación. Por lo tanto, los cambios que realices en el proxy de tu AWS CDK aplicación no afectan al recurso implementado. Sin embargo, el proxy se puede pasar a cualquier AWS CDK método que requiera un recurso de ese tipo.

En el siguiente ejemplo, se muestra cómo hacer referencia a un depósito en función de un depósito existente con el ARN arn:aws:s3::: my-bucket-name y a una Amazon Virtual Private Cloud basada en una VPC existente con un ID específico.

TypeScript

```
// Construct a proxy for a bucket by its name (must be same account)
s3.Bucket.fromBucketName(this, 'MyBucket', 'my-bucket-name');

// Construct a proxy for a bucket by its full ARN (can be another account)
s3.Bucket.fromBucketArn(this, 'MyBucket', 'arn:aws:s3:::my-bucket-name');

// Construct a proxy for an existing VPC from its attribute(s)
ec2.Vpc.fromVpcAttributes(this, 'MyVpc', {
  vpcId: 'vpc-1234567890abcde',
});
```

JavaScript

```
// Construct a proxy for a bucket by its name (must be same account)
s3.Bucket.fromBucketName(this, 'MyBucket', 'my-bucket-name');

// Construct a proxy for a bucket by its full ARN (can be another account)
s3.Bucket.fromBucketArn(this, 'MyBucket', 'arn:aws:s3:::my-bucket-name');
```

```
// Construct a proxy for an existing VPC from its attribute(s)
ec2.Vpc.fromVpcAttributes(this, 'MyVpc', {
  vpcId: 'vpc-1234567890abcde'
});
```

Python

```
# Construct a proxy for a bucket by its name (must be same account)
s3.Bucket.from_bucket_name(self, "MyBucket", "my-bucket-name")

# Construct a proxy for a bucket by its full ARN (can be another account)
s3.Bucket.from_bucket_arn(self, "MyBucket", "arn:aws:s3:::my-bucket-name")

# Construct a proxy for an existing VPC from its attribute(s)
ec2.Vpc.from_vpc_attributes(self, "MyVpc", vpc_id="vpc-1234567890abcdef")
```

Java

```
// Construct a proxy for a bucket by its name (must be same account)
Bucket.fromBucketName(this, "MyBucket", "my-bucket-name");

// Construct a proxy for a bucket by its full ARN (can be another account)
Bucket.fromBucketArn(this, "MyBucket",
        "arn:aws:s3:::my-bucket-name");

// Construct a proxy for an existing VPC from its attribute(s)
Vpc.fromVpcAttributes(this, "MyVpc", VpcAttributes.builder()
        .vpcId("vpc-1234567890abcdef").build());
```

C#

```
// Construct a proxy for a bucket by its name (must be same account)
Bucket.FromBucketName(this, "MyBucket", "my-bucket-name");

// Construct a proxy for a bucket by its full ARN (can be another account)
Bucket.FromBucketArn(this, "MyBucket", "arn:aws:s3:::my-bucket-name");

// Construct a proxy for an existing VPC from its attribute(s)
Vpc.FromVpcAttributes(this, "MyVpc", new VpcAttributes
{
    VpcId = "vpc-1234567890abcdef"
});
```

Analicemos más de cerca el método. `Vpc.fromLookup()` Como la `ec2.Vpc` construcción es compleja, hay muchas maneras de seleccionar la VPC que se va a usar con la aplicación de CDK. Para solucionar este problema, la construcción de VPC tiene un método `fromLookup` estático (Python:`from_lookup`) que le permite buscar la Amazon VPC deseada consultando su AWS cuenta en el momento de la síntesis.

Para poder utilizarlo`Vpc.fromLookup()`, el sistema que sintetiza la pila debe tener acceso a la cuenta propietaria de Amazon VPC. Esto se debe a que el CDK Toolkit consulta la cuenta para encontrar la Amazon VPC adecuada en el momento de la síntesis.

Además, solo `Vpc.fromLookup()` funciona en pilas definidas con una cuenta y una región explícitas (consulte). the section called “Entornos” Si AWS CDK intenta buscar una Amazon VPC desde una pila independiente del entorno, el CDK Toolkit no sabrá qué entorno consultar para encontrar la VPC.

Debe proporcionar `Vpc.fromLookup()` atributos suficientes para identificar de forma exclusiva una VPC en su AWS cuenta. Por ejemplo, solo puede haber una VPC predeterminada, por lo que basta con especificar la VPC como predeterminada.

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 });
```

También puedes usar la `tags` propiedad para consultar las VPC por etiqueta. Puede añadir etiquetas a la Amazon VPC en el momento de su creación utilizando AWS CloudFormation o la. AWS CDK Puede editar las etiquetas en cualquier momento después de crearlas mediante el AWS Management Console AWS CLI, el o un AWS SDK. Además de las etiquetas que añada usted mismo, añade AWS CDK automáticamente las siguientes etiquetas a todas las VPC que cree.

- Nombre: el nombre de la VPC.
- aws-cdk:subnet-name: el nombre de la subred.
- aws-cdk:subnet-type: tipo de subred: pública, privada o aislada.

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" });
```

Los resultados de se almacenan en caché en el archivo del proyecto. `Vpc.fromLookup()` `cdk.context.json` (Consulte the section called “Context”). Confirma este archivo al control de versiones para que tu aplicación siga haciendo referencia a la misma Amazon VPC. Esto funciona incluso si posteriormente cambias los atributos de tus VPC de forma que se seleccione una VPC diferente. Esto es especialmente importante si vas a implementar la pila en un entorno que no tiene acceso a la AWS cuenta que define la VPC, como CDK Pipelines.

Aunque puedes usar un recurso externo en cualquier lugar en el que utilices un recurso similar definido en tu AWS CDK aplicación, no puedes modificarlo. Por ejemplo, llamar a `addToResourcePolicy` (Python:`add_to_resource_policy`) en un dispositivo externo `s3.Bucket` no hace nada.

Nombres físicos de los recursos

Los nombres lógicos de los recursos AWS CloudFormation son diferentes de los nombres de los recursos que aparecen AWS Management Console después de su despliegue AWS CloudFormation. A estos nombres finales los AWS CDK llama nombres físicos.

Por ejemplo, AWS CloudFormation podría crear el bucket de Amazon S3 con el ID lógico `Stack2MyBucket4DD88B4F` del ejemplo anterior con el nombre físico`stack2mybucket4dd88b4f-iuv1rbv9z3to`.

Puede especificar un nombre físico al crear componentes fijos que representen recursos mediante la propiedad <resourceType>Name. En el siguiente ejemplo, se crea un bucket de Amazon S3 con el nombre físico`my-bucket-name`.

TypeScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket-name',
});
```

JavaScript

```
const bucket = new s3.Bucket(this, 'MyBucket', {
  bucketName: 'my-bucket-name'
```

```
});
```

Python

```
bucket = s3.Bucket(self, "MyBucket", bucket_name="my-bucket-name")
```

Java

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
        .bucketName("my-bucket-name").build();
```

C#

```
var bucket = new Bucket(this, "MyBucket", new BucketProps { BucketName = "my-bucket-
name" });
```

La asignación de nombres físicos a los recursos tiene algunas desventajas. AWS CloudFormation Y lo que es más importante, cualquier cambio en los recursos desplegados que requiera la sustitución de un recurso, como los cambios en las propiedades de un recurso que sean inmutables tras su creación, fallará si a un recurso se le ha asignado un nombre físico. Si terminas en ese estado, la única solución es eliminar la AWS CloudFormation pila y volver a implementar la AWS CDK aplicación. Consulte la AWS CloudFormation documentación para obtener más información.

En algunos casos, como cuando se crea una AWS CDK aplicación con referencias a varios entornos, se requieren nombres físicos para AWS CDK que funcione correctamente. En esos casos, si no quieres molestarte en crear un nombre físico tú mismo, puedes dejar que se lo AWS CDK ponga por ti. Para hacerlo, usa el valor especial`PhysicalName.GENERATE_IF_NEEDED`, de la siguiente manera.

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 });
```

Pasar identificadores de recursos únicos

Siempre que sea posible, debes pasar los recursos por referencia, como se describe en la sección anterior. Sin embargo, hay casos en los que no tiene otra opción que hacer referencia a un recurso por uno de sus atributos. Entre los ejemplos de casos de uso se incluyen los siguientes:

- Cuando se utilizan AWS CloudFormation recursos de bajo nivel.
- Cuando necesita exponer los recursos a los componentes de tiempo de ejecución de una AWS CDK aplicación, como cuando se hace referencia a las funciones de Lambda a través de variables de entorno.

Estos identificadores están disponibles como atributos en los recursos, como los siguientes.

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

El AWS CDK enlace de Java utiliza métodos de captación para los atributos.

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

C#

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

En el siguiente ejemplo, se muestra cómo pasar el nombre de un bucket generado a una AWS Lambda función.

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
    }
});
```

Otorgar permisos entre recursos

Las construcciones de nivel superior permiten obtener permisos con privilegios mínimos al ofrecer API sencillas y basadas en la intención para expresar los requisitos de permisos. Por ejemplo, muchas construcciones de nivel 2 ofrecen métodos de concesión que se pueden utilizar para

conceder a una entidad (por ejemplo, un rol o un usuario de IAM) permiso para trabajar con el recurso, sin tener que crear manualmente las declaraciones de permiso de IAM.

El siguiente ejemplo crea los permisos para permitir que la función de ejecución de una función de Lambda lea y escriba objetos en un bucket de Amazon S3 concreto. Si el bucket de Amazon S3 está cifrado con una AWS KMS clave, este método también concede permisos a la función de ejecución de la función Lambda para descifrar con la clave.

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)
{
    // ...
}
```

Los métodos de concesión devuelven un `iam.Grant` objeto. Utilice el `success` atributo del `Grant` objeto para determinar si la subvención se aplicó de manera efectiva (por ejemplo, puede que

no se haya aplicado a recursos externos). También puedes usar el método `assertSuccess` (Python:`assert_success`) del `Grant` objeto para garantizar que la concesión se haya aplicado correctamente.

Si un método de concesión específico no está disponible para un caso de uso concreto, puedes usar un método de concesión genérico para definir una nueva concesión con una lista de acciones específica.

El siguiente ejemplo muestra cómo conceder a una función de Lambda acceso a la acción de 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");
```

Muchos recursos, como las funciones Lambda, requieren que se asuma un rol al ejecutar el código. Una propiedad de configuración le permite especificar un`iam.IRole`. Si no se especifica ningún rol, la función crea automáticamente un rol específico para este uso. A continuación, puede utilizar métodos de concesión en los recursos para añadir declaraciones al rol.

Los métodos de concesión se crean con API de nivel inferior para gestionarlos con las políticas de IAM. Las políticas se modelan como objetos. PolicyDocument Agregue sentencias directamente

a los roles (o al rol adjunto de una construcción) mediante el `addToRolePolicy` método (Python:`add_to_role_policy`) o a la política de un recurso (como una `Bucket` política) mediante el método `addToResourcePolicy` (Python:`add_to_resource_policy`).

Métricas y alarmas de recursos

Muchos recursos emiten CloudWatch métricas que se pueden usar para configurar paneles y alarmas de monitoreo. Las construcciones de nivel superior tienen métodos métricos que permiten acceder a las métricas sin tener que buscar el nombre correcto que se va a utilizar.

El siguiente ejemplo muestra cómo definir una alarma cuando la cola `ApproximateNumberOfMessagesNotVisible` de Amazon SQS supera los 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,
    // ..
});
```

Si no hay ningún método para una métrica en particular, puede utilizar el método de métrica general para especificar el nombre de la métrica manualmente.

Las métricas también se pueden añadir a los CloudWatch paneles. Consulte CloudWatch.

Tráfico de red

En muchos casos, debe habilitar los permisos en una red para que una aplicación funcione, por ejemplo, cuando la infraestructura informática necesita acceder a la capa de persistencia. Los recursos que establecen o detectan las conexiones exponen métodos que permiten los flujos de tráfico, como el establecimiento de reglas de grupos de seguridad o ACL de red.

Los recursos de iConnectable tienen una `connections` propiedad que es la puerta de entrada a la configuración de las reglas de tráfico de la red.

Para permitir que los datos fluyan en una ruta de red determinada, se utilizan `allow` métodos. El siguiente ejemplo habilita las conexiones HTTPS a la web y las conexiones entrantes desde el grupo Auto Scaling de Amazon EC2. `fleet2`

TypeScript

```
import * as asg from '@aws-cdk/aws-autoscaling';
import * as ec2 from '@aws-cdk/aws-ec2';

const fleet1: asg.AutoScalingGroup = asg.AutoScalingGroup(/*...*/);

// Allow surfing the (secure) web
fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443,
 toPort: 443 }));

const fleet2: asg.AutoScalingGroup = asg.AutoScalingGroup(/*...*/);
fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic());
```

JavaScript

```
const asg = require('@aws-cdk/aws-autoscaling');
const ec2 = require('@aws-cdk/aws-ec2');

const fleet1 = asg.AutoScalingGroup();

// Allow surfing the (secure) web
fleet1.connections.allowTo(new ec2.Peer.anyIpv4(), new ec2.Port({ fromPort: 443,
 toPort: 443 }));

const fleet2 = asg.AutoScalingGroup();
fleet1.connections.allowFrom(fleet2, ec2.Port.AllTraffic());
```

Python

```
import aws_cdk.aws_autoscaling as asg
import aws_cdk.aws_ec2 as ec2

fleet1 = asg.AutoScalingGroup( ... )

# Allow surfing the (secure) web
fleet1.connections.allow_to(ec2.Peer.any_ipv4(),
  ec2.Port(PortProps(from_port=443, to_port=443)))
```

```
fleet2 = asg.AutoScalingGroup( ... )
fleet1.connections.allow_from(fleet2, ec2.Port.all_traffic())
```

Java

```
import software.amazon.awscdk.services.autoscaling.AutoScalingGroup;
import software.amazon.awscdk.services.ec2.Peer;
import software.amazon.awscdk.services.ec2.Port;

AutoScalingGroup fleet1 = AutoScalingGroup.Builder.create(this, "MyFleet")
        /* ... */.build();

// Allow surfing the (secure) Web
fleet1.getConnections().allowTo(Peer.anyIpv4(),
        Port.Builder.create().fromPort(443).toPort(443).build());

AutoScalingGroup fleet2 = AutoScalingGroup.Builder.create(this, "MyFleet2")
        /* ... */.build();
fleet1.getConnections().allowFrom(fleet2, Port.allTraffic());
```

C#

```
using cdk = Amazon.CDK;
using asg = Amazon.CDK.AWS.AutoScaling;
using ec2 = Amazon.CDK.AWS.EC2;

// Allow surfing the (secure) Web
var fleet1 = new asg.AutoScalingGroup(this, "MyFleet", new asg.AutoScalingGroupProps
 { /* ... */ });
fleet1.Connections.AllowTo(ec2.Peer.AnyIpv4(), new ec2.Port(new ec2.PortProps
  { FromPort = 443, ToPort = 443 });

var fleet2 = new asg.AutoScalingGroup(this, "MyFleet2", new
 asg.AutoScalingGroupProps { /* ... */ });
fleet1.Connections.AllowFrom(fleet2, ec2.Port.AllTraffic());
```

Algunos recursos tienen puertos predeterminados asociados. Los ejemplos incluyen el receptor de un balanceador de carga en el puerto público y los puertos en los que el motor de base de datos acepta conexiones para instancias de una base de datos de Amazon RDS. En estos casos, puede imponer un control estricto de la red sin tener que especificar el puerto

manualmente. Para ello, utilice los `allowToDefaultPort` métodos `allowDefaultPortFrom` y (Python:`allow_default_port_from`,`allow_to_default_port`).

El siguiente ejemplo muestra cómo habilitar las conexiones desde cualquier dirección IPV4 y una conexión desde un grupo de Auto Scaling para acceder a una base de datos.

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");
```

Gestión de eventos

Algunos recursos pueden actuar como fuentes de eventos. Utilice el `addEventNotification` método (Python:`add_event_notification`) para registrar un objetivo de evento en un tipo de

evento concreto emitido por el recurso. Además, los `addXxxNotification` métodos ofrecen una forma sencilla de registrar un controlador para los tipos de eventos más comunes.

El siguiente ejemplo muestra cómo activar una función Lambda cuando se añade un objeto a un bucket de Amazon S3.

TypeScript

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

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

JavaScript

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

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

Python

```
import aws_cdk.aws_s3_notifications as s3_nots

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

Java

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

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

C#

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

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

Políticas de retirada

Los recursos que mantienen datos persistentes, como las bases de datos, los buckets de Amazon S3 y los registros de Amazon ECR, tienen una política de eliminación. La política de eliminación indica si se deben eliminar los objetos persistentes cuando se destruya la AWS CDK pila que los contiene. Los valores que especifican la política de eliminación están disponibles en la `RemovalPolicy` enumeración del AWS CDK `core` módulo.

> **Note**
>
> Los recursos, además de los que almacenan datos de forma persistente, también pueden tener un recurso `removalPolicy` que se utilice para un propósito diferente. Por ejemplo, una versión de una función Lambda usa un `removalPolicy` atributo para determinar si una versión determinada se conserva cuando se implementa una nueva versión. Tienen significados y valores predeterminados diferentes a los de la política de retirada de un bucket de Amazon S3 o una tabla de DynamoDB.

Valor	sentido
`RemovalPolicy.RETENER`	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.

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

AWS CloudFormation no elimina los buckets de Amazon S3 que contienen archivos, incluso si su política de eliminación está establecida en. `DESTROY` Intentar hacerlo es un AWS CloudFormation error. Para AWS CDK eliminar todos los archivos del depósito antes de destruirlo, defina la `autoDeleteObjects` propiedad del depósito en`true`.

A continuación se muestra un ejemplo de cómo crear un bucket de Amazon S3 con `RemovalPolicy` of `DESTROY` y `autoDeleteOjbects` establecido en`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
    });
}
```

También puedes aplicar una política de retirada directamente al AWS CloudFormation recurso subyacente mediante `applyRemovalPolicy()` este método. Este método está disponible en algunos recursos con estado que no tienen una `removalPolicy` propiedad en los accesorios de sus recursos de nivel 2. Algunos ejemplos son los siguientes:

- AWS CloudFormation pilas
- Grupos de usuarios de Amazon Cognito
- Instancias de bases de datos Amazon DocumentDB
- Volúmenes de Amazon EC2
- Dominios OpenSearch de Amazon Service
- Sistemas de archivos Amazon FSx
- Colas de 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 AWS CDK«sRemovalPolicy» se traduce como «s» AWS CloudFormation. `DeletionPolicy` Sin embargo, la entrada predeterminada AWS CDK es conservar los datos, que es lo contrario de lo AWS CloudFormation predeterminado.

Identificadores

Al crear AWS Cloud Development Kit (AWS CDK) aplicaciones, utilizarás muchos tipos de identificadores y nombres. Para utilizarlos de AWS CDK forma eficaz y evitar errores, es importante entender los tipos de identificadores.

Los identificadores deben ser únicos dentro del ámbito en el que se crean; no es necesario que sean únicos a nivel mundial en su AWS CDK aplicación.

Si intentas crear un identificador con el mismo valor dentro del mismo ámbito, se generará AWS CDK una excepción.

Temas

Construye los ID

El identificador más común,`id`, es el identificador que se pasa como segundo argumento al crear una instancia de un objeto de construcción. Este identificador, como todos los identificadores, solo debe ser único dentro del ámbito en el que se creó, que es el primer argumento al instanciar un objeto de construcción.

> **Note**
>
> El `id` de una pila es también el identificador que se utiliza para hacer referencia a ella en. the section called “AWS CDK Kit de herramientas”

Veamos un ejemplo en el que tenemos dos construcciones con el identificador `MyBucket` en nuestra aplicación. La primera se define en el ámbito de la pila con el identificador`Stack1`. El segundo se define en el ámbito de una pila con el identificador`Stack2`. Como se definen en ámbitos diferentes, esto no genera ningún conflicto y pueden coexistir en la misma aplicación sin problemas.

TypeScript

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

class MyStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps = {}) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyBucket');
  }
}

const app = new App();
new MyStack(app, 'Stack1');
new MyStack(app, 'Stack2');
```

JavaScript

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

```
class MyStack extends Stack {
  constructor(scope, id, props = {}) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyBucket');
  }
}

const app = new App();
new MyStack(app, 'Stack1');
new MyStack(app, 'Stack2');
```

Python

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

class MyStack(Stack):

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

        super().__init__(scope, id, **kwargs)
        s3.Bucket(self, "MyBucket")

app = App()
MyStack(app, 'Stack1')
MyStack(app, 'Stack2')
```

Java

```
// MyStack.java
package com.myorg;

import software.amazon.awscdk.App;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.constructs.Construct;
import software.amazon.awscdk.services.s3.Bucket;

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

```
    }

    public MyStack(final Construct scope, final String id, final StackProps props) {
        super(scope, id, props);
        new Bucket(this, "MyBucket");
    }
}

// Main.java
package com.myorg;

import software.amazon.awscdk.App;

public class Main {
    public static void main(String[] args) {
        App app = new App();
        new MyStack(app, "Stack1");
        new MyStack(app, "Stack2");
    }
}
```

C#

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

public class MyStack : Stack
{
    public MyStack(Construct scope, string id, IStackProps props) : base(scope, id,
 props)
    {
        new Bucket(this, "MyBucket");
    }
}

class Program
{
    static void Main(string[] args)
    {
        var app = new App();
        new MyStack(app, "Stack1");
        new MyStack(app, "Stack2");
```

```
    }
}
```

Rutas

Los constructos de una AWS CDK aplicación forman una jerarquía basada en la clase. App Nos referimos a la colección de identificadores de un constructo dado, su constructo padre, su antepasado, etc., hasta la raíz del árbol del constructo, como ruta.

AWS CDK Por lo general, muestra las rutas de las plantillas en forma de cadena. Los identificadores de los niveles se separan mediante barras diagonales, empezando por el nodo situado inmediatamente debajo de la App instancia raíz, que suele ser una pila. Por ejemplo, las rutas de los dos recursos de bucket de Amazon S3 del ejemplo de código anterior son `Stack1/MyBucket` y`Stack2/MyBucket`.

Puede acceder a la ruta de cualquier construcción mediante programación, como se muestra en el siguiente ejemplo. Este es el camino de `myConstruct` (o`my_construct`, como lo escribirían los desarrolladores de Python). Como los ID deben ser únicos dentro del ámbito en el que se crean, sus rutas siempre son únicas dentro de una AWS CDK aplicación.

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 únicos

AWS CloudFormation requiere que todos los ID lógicos de una plantilla sean únicos. Por ello, AWS CDK debe poder generar un identificador único para cada construcción de una aplicación. Los recursos tienen rutas que son únicas a nivel mundial (los nombres de todos los ámbitos de la pila hasta un recurso específico). Por lo tanto, AWS CDK genera los identificadores únicos necesarios concatenando los elementos de la ruta y añadiendo un hash de 8 dígitos. (El hash es necesario para distinguir rutas distintas, como `A/B/C` y`A/BC`, que darían como resultado el mismo identificador. AWS CloudFormation AWS CloudFormation los identificadores son alfanuméricos y no pueden contener barras ni otros caracteres separadores.) Él AWS CDK llama a esta cadena el identificador único de la construcción.

En general, tu AWS CDK aplicación no debería necesitar conocer los identificadores únicos. Sin embargo, puedes acceder al identificador único de cualquier construcción mediante programación, como se muestra en el siguiente ejemplo.

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);
```

La dirección es otro tipo de identificador único que distingue de forma única los recursos de CDK. Derivado del hash SHA-1 de la ruta, no es legible por humanos. Sin embargo, su longitud constante

y relativamente corta (siempre 42 caracteres hexadecimales) lo hace útil en situaciones en las que el identificador único «tradicional» puede ser demasiado largo. Algunas construcciones pueden usar la dirección de la AWS CloudFormation plantilla sintetizada en lugar del identificador único. De nuevo, por lo general, tu aplicación no debería necesitar conocer las direcciones de sus componentes, pero puedes recuperar la dirección de un componente de la siguiente manera.

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;
```

Identificadores lógicos

Los identificadores únicos sirven como identificadores lógicos (o nombres lógicos) de los recursos en las AWS CloudFormation plantillas generadas para las construcciones que representan AWS los recursos.

Por ejemplo, el bucket de Amazon S3 del ejemplo anterior que se crea dentro `Stack2` da como resultado un `AWS::S3::Bucket` recurso. El ID lógico del recurso se encuentra `Stack2MyBucket4DD88B4F` en la AWS CloudFormation plantilla resultante. (Para obtener más información sobre cómo se genera este identificador, consultethe section called “ID únicos”.)

Estabilidad del ID lógico

Evite cambiar el identificador lógico de un recurso una vez creado. AWS CloudFormation identifica los recursos por su identificador lógico. Por lo tanto, si cambia el identificador lógico de un recurso, AWS CloudFormation crea un nuevo recurso con el nuevo identificador lógico y, a continuación, elimina el existente. Según el tipo de recurso, esto puede provocar la interrupción del servicio, la pérdida de datos o ambas cosas.

Tokens

Los tokens representan valores que solo se pueden resolver en un momento posterior del ciclo de vida de la aplicación. Por ejemplo, el nombre de un depósito de Amazon Simple Storage Service (Amazon S3) que defina en su aplicación de CDK solo se asigna cuando se sintetiza AWS CloudFormation la plantilla. Si imprime el `bucket.bucketName` atributo, que es una cadena, verá que contiene algo parecido a lo siguiente:

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

Así es como AWS CDK codifica un token cuyo valor aún no se conoce en el momento de la construcción, pero que estará disponible más adelante. A estos marcadores de posición los AWS CDK llaman «fichas». En este caso, es un token codificado como una cadena.

Puedes pasar esta cadena como si fuera el nombre del depósito. En el siguiente ejemplo, el nombre del depósito se especifica como una variable de entorno para una AWS Lambda función.

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
    }
});
```

Cuando finalmente se sintetiza la AWS CloudFormation plantilla, el token se representa como AWS CloudFormation intrínseco`{ "Ref": "MyBucket" }`. En el momento de la implementación, AWS CloudFormation reemplaza este elemento intrínseco por el nombre real del depósito que se creó.

Temas

- Tokens y codificaciones de tokens

- Tokens codificados en cadenas
- Tokens codificados en forma de lista
- Tokens codificados con números
- Valores perezosos
- Conversión a JSON

Tokens y codificaciones de tokens

Los tokens son objetos que implementan la interfaz IRresolvable, que contiene un único método. `resolve` Utiliza AWS CDK este método durante la síntesis para obtener el valor final de la plantilla. AWS CloudFormation Los tokens participan en el proceso de síntesis para producir valores arbitrarios de cualquier tipo.

> **Note**
>
> Rara vez trabajarás directamente con la `IResolvable` interfaz. Lo más probable es que solo veas versiones de los tokens codificadas en cadenas.

Por lo general, otras funciones solo aceptan argumentos de tipos básicos, como `string` o. `number` Para usar los tokens en estos casos, puedes codificarlos en uno de los tres tipos mediante métodos estáticos de la clase CDK.Token.

- `Token.asString`para generar una codificación de cadena (o invocar `.toString()` el objeto token)
- `Token.asList`para generar una codificación de lista
- `Token.asNumber`para generar una codificación numérica

Estos toman un valor arbitrario, que puede ser un`IResolvable`, y lo codifican en un valor primitivo del tipo indicado.

> **Important**
>
> Como cualquiera de los tipos anteriores puede ser un token codificado, tenga cuidado al analizar o intentar leer su contenido. Por ejemplo, si intentas analizar una cadena para

extraer un valor de ella y la cadena es un token codificado, el análisis no se realizará correctamente. Del mismo modo, si intentas consultar la longitud de una matriz o realizar operaciones matemáticas con un número, primero debes comprobar que no son símbolos codificados.

Para comprobar si un valor contiene un token sin resolver, llama al método `Token.isUnresolved` (Python:`is_unresolved`).

El siguiente ejemplo valida que un valor de cadena, que podría ser un token, no supere los 10 caracteres.

TypeScript

```
if (!Token.isUnresolved(name) && name.length > 10) {
  throw new Error(`Maximum length for name is 10 characters`);
}
```

JavaScript

```
if ( !Token.isUnresolved(name) && name.length > 10) {
  throw ( new Error(`Maximum length for name is 10 characters`));
}
```

Python

```
if not Token.is_unresolved(name) and len(name) > 10:
    raise ValueError("Maximum length for name is 10 characters")
```

Java

```
if (!Token.isUnresolved(name) && name.length() > 10)
    throw new IllegalArgumentException("Maximum length for name is 10 characters");
```

C#

```
if (!Token.IsUnresolved(name) && name.Length > 10)
    throw new ArgumentException("Maximum length for name is 10 characters");
```

Si el nombre es un token, no se realiza la validación y, aun así, podría producirse un error en una fase posterior del ciclo de vida, por ejemplo, durante la implementación.

> **Note**
>
> Puedes usar codificaciones por token para escapar del sistema de tipos. Por ejemplo, puede codificar en cadena un token que produzca un valor numérico en el momento de la síntesis. Si utilizas estas funciones, es tu responsabilidad asegurarte de que la plantilla se resuelva a un estado utilizable después de la síntesis.

Tokens codificados en cadenas

Los tokens codificados en cadenas tienen el siguiente aspecto.

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

Se pueden transmitir como cadenas normales y se pueden concatenar, como se muestra en el siguiente ejemplo.

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";
```

También puede utilizar la interpolación de cadenas, si su idioma la admite, como se muestra en el siguiente ejemplo.

TypeScript

```
const functionName = `${bucket.bucketName}Function`;
```

JavaScript

```
const functionName = `${bucket.bucketName}Function`;
```

Python

```
function_name = f"{bucket.bucket_name}Function"
```

Java

```
String functionName = String.format("%sFunction". bucket.getBucketName());
```

C#

```
string functionName = $"${bucket.bucketName}Function";
```

Evite manipular la cadena de otras formas. Por ejemplo, si se toma una subcadena de una cadena, es probable que se rompa el símbolo de la cadena.

Tokens codificados en forma de lista

Los tokens codificados en forma de lista tienen el siguiente aspecto:

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

Lo único seguro que se puede hacer con estas listas es pasarlas directamente a otras construcciones. Los tokens en forma de lista de cadenas no se pueden concatenar ni se puede extraer ningún elemento del token. La única forma segura de manipularlos es mediante el uso de funciones AWS CloudFormation intrínsecas como fn.Select.

Tokens codificados con números

Los símbolos codificados con números son un conjunto de pequeños números negativos de punto flotante que tienen el siguiente aspecto.

```
-1.8881545897087626e+289
```

Al igual que ocurre con los símbolos de lista, no se puede modificar el valor numérico, ya que es probable que se rompa el identificador numérico. La única operación permitida es transferir el valor a otra construcción.

Valores perezosos

Además de representar los valores del tiempo de despliegue, como los AWS CloudFormation parámetros, los tokens también se utilizan habitualmente para representar los valores perezosos del tiempo de síntesis. Estos son valores para los que el valor final se determinará antes de que se complete la síntesis, pero no en el punto en el que se construye el valor. Utilice símbolos para pasar un valor numérico o de cadena literal a otra construcción, mientras que el valor real en el momento de la síntesis puede depender de algún cálculo que aún no se haya realizado.

Puedes construir fichas que representen valores perezosos en el tiempo de síntesis utilizando los métodos estáticos de la `Lazy` clase, como Lazy.String y Lazy.Number. Estos métodos aceptan un objeto cuya `produce` propiedad es una función que acepta un argumento de contexto y devuelve el valor final cuando se llama.

El siguiente ejemplo crea un grupo de Auto Scaling cuya capacidad se determina después de su creación.

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;
```

Conversión a JSON

A veces, desea generar una cadena JSON de datos arbitrarios y es posible que no sepa si los datos contienen símbolos. Para codificar correctamente en JSON cualquier estructura de datos, independientemente de si contiene o no tokens, utilice la pila de métodos. toJsonString, como se muestra en el siguiente ejemplo.

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
});
```

Parámetros

Los parámetros son valores personalizados que se proporcionan en el momento de la implementación. Los parámetros son una característica de AWS CloudFormation. Dado que AWS

Cloud Development Kit (AWS CDK) sintetiza AWS CloudFormation plantillas, también ofrece soporte para los parámetros del tiempo de implementación.

Temas

Acerca de los parámetros

Con el AWS CDK, puede definir parámetros, que luego se pueden usar en las propiedades de los componentes fijos que cree. También puede implementar pilas que contengan parámetros.

Al implementar la AWS CloudFormation plantilla mediante el AWS CDK kit de herramientas, debe proporcionar los valores de los parámetros en la línea de comandos. Si despliega la plantilla a través de la AWS CloudFormation consola, se le solicitarán los valores de los parámetros.

En general, no se recomienda utilizar AWS CloudFormation parámetros con AWS CDK. Las formas habituales de pasar valores a AWS CDK las aplicaciones son los valores de contexto y las variables de entorno. Como no están disponibles en el momento de la síntesis, los valores de los parámetros no se pueden usar fácilmente para controlar el flujo y otros fines en su aplicación de CDK.

> **Note**
>
> Para controlar el flujo con parámetros, puede utilizar `CfnCondition`construcciones, aunque esto resulta incómodo en comparación con las sentencias nativas`if`.

El uso de parámetros requiere que tengas en cuenta cómo se comporta el código que estás escribiendo en el momento de la implementación y también en el momento de la síntesis. Esto hace que sea más difícil entender y razonar acerca de la AWS CDK aplicación y, en muchos casos, ofrece pocos beneficios.

Por lo general, es mejor hacer que la aplicación de CDK acepte la información necesaria de forma bien definida y que la utilice directamente para declarar las construcciones en la aplicación de CDK. Una AWS CloudFormation plantilla ideal AWS CDK generada es concreta, sin que queden valores por especificar en el momento de la implementación.

Sin embargo, hay casos de uso para los que AWS CloudFormation los parámetros son especialmente adecuados. Si tiene equipos independientes que definen e implementan la infraestructura, por ejemplo, puede usar los parámetros para que las plantillas generadas sean más útiles. Además, dado que AWS CloudFormation los parámetros de AWS CDK soporte, se pueden utilizar AWS CDK con AWS servicios que utilizan AWS CloudFormation plantillas (como Service Catalog). Estos AWS servicios utilizan parámetros para configurar la plantilla que se va a implementar.

Definir parámetros

Utilice la `CfnParameter`clase para definir un parámetro. Deberá especificar al menos un tipo y una descripción para la mayoría de los parámetros, aunque ambos son técnicamente opcionales. La descripción aparece cuando se le pide al usuario que introduzca el valor del parámetro en la AWS CloudFormation consola. Para obtener más información sobre los tipos disponibles, consulte Tipos.

> **Note**
>
> Puede definir parámetros en cualquier ámbito. Sin embargo, recomendamos definir los parámetros a nivel de pila para que su ID lógico no cambie al refactorizar el código.

TypeScript

```
const uploadBucketName = new CfnParameter(this, "uploadBucketName", {
  type: "String",
  description: "The name of the Amazon S3 bucket where uploaded files will be
 stored."});
```

JavaScript

```
const uploadBucketName = new CfnParameter(this, "uploadBucketName", {
  type: "String",
  description: "The name of the Amazon S3 bucket where uploaded files will be
 stored."});
```

Python

```
upload_bucket_name = CfnParameter(self, "uploadBucketName", type="String",
```

```
    description="The name of the Amazon S3 bucket where uploaded files will be
 stored.")
```

Java

```
CfnParameter uploadBucketName = CfnParameter.Builder.create(this,
 "uploadBucketName")
        .type("String")
        .description("The name of the Amazon S3 bucket where uploaded files will be
 stored")
        .build();
```

C#

```
var uploadBucketName = new CfnParameter(this, "uploadBucketName", new
 CfnParameterProps
{
    Type = "String",
    Description = "The name of the Amazon S3 bucket where uploaded files will be
 stored"
});
```

Uso de parámetros

Una `CfnParameter` instancia expone su valor a tu AWS CDK aplicación mediante un token. Como todos los tokens, el token del parámetro se resuelve en el momento de la síntesis. Sin embargo, se basa en una referencia al parámetro definido en la AWS CloudFormation plantilla (que se resolverá en el momento de la implementación), más que en un valor concreto.

Puede recuperar el token como una instancia de la `Token` clase o en una cadena, lista de cadenas o codificación numérica. La elección depende del tipo de valor que requiera la clase o el método con el que desee utilizar el parámetro.

TypeScript

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

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

JavaScript

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

Python

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

Java

Property	kind of value
`getValue ()`	`Token` class instance
`getValueAsLista ()`	The token represented as a string list

Property	kind of value
`getValueAsNúmero ()`	The token represented as a number
`getValueAsCadena ()`	The token represented as a string

C#

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

Por ejemplo, para usar un parámetro en una `Bucket` definición:

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
};
```

Implementación con parámetros

Una plantilla generada que contiene parámetros se puede implementar de la forma habitual a través de la AWS CloudFormation consola. Se le solicitarán los valores de cada parámetro.

El AWS CDK kit de herramientas (herramienta de línea de `cdk` comandos) también permite especificar parámetros durante el despliegue. Los proporciona en la línea de comandos siguiendo la `--parameters` marca. Puede implementar una pila que utilice el `uploadBucketName` parámetro, como en el ejemplo siguiente.

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

Para definir varios parámetros, utilice varios `--parameters` indicadores.

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

Si va a implementar varias pilas, puede especificar un valor diferente de cada parámetro para cada pila. Para ello, añada al nombre del parámetro el nombre de la pila y dos puntos.

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

De forma predeterminada, AWS CDK conserva los valores de los parámetros de las implementaciones anteriores y los utiliza en las implementaciones posteriores si no se especifican de forma explícita. Utilice la `--no-previous-parameters` marca para solicitar que se especifiquen todos los parámetros.

Etiquetado

Las etiquetas son elementos clave-valor informativos que puedes agregar a las construcciones de tu aplicación. AWS CDK Una etiqueta aplicada a una construcción determinada también se aplica a todos sus elementos secundarios etiquetables. Las etiquetas se incluyen en la AWS CloudFormation plantilla sintetizada a partir de la aplicación y se aplican a los AWS recursos que implementa. Puedes usar etiquetas para identificar y clasificar los recursos con los siguientes fines:

- Simplificar la administración
- Asignación de costos
- Control de acceso
- ¿Algún otro propósito que se proponga

> **Tip**
>
> Para obtener más información sobre cómo utilizar las etiquetas con AWS los recursos, consulte las prácticas recomendadas para etiquetar AWS los recursos en el AWS documento técnico.

Temas

Uso de etiquetas

La `Tags`clase incluye el método estáticoof(), mediante el cual puede añadir o eliminar etiquetas de la construcción especificada.

- `Tags.of(SCOPE).add()`aplica una nueva etiqueta a la construcción dada y a todos sus elementos secundarios.

- `Tags.of(SCOPE).remove()` elimina una etiqueta de la construcción dada y de cualquiera de sus elementos secundarios, incluidas las etiquetas que una construcción secundaria pueda haberse aplicado a sí misma.

> **Note**
>
> El etiquetado se implementa mediante the section called “Aspectos”. Los aspectos son una forma de aplicar una operación (como el etiquetado) a todos los constructos de un ámbito determinado.

El siguiente ejemplo aplica la clave de etiqueta con el valor del valor a una construcción.

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");
```

En el siguiente ejemplo, se elimina la clave de etiqueta de una construcción.

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");
```

Si utiliza `Stage` componentes fijos, aplique la etiqueta en el `Stage` nivel o por debajo. Las etiquetas no se aplican a través de `Stage` los límites.

Prioridades de etiquetas

AWS CDK Aplica y elimina las etiquetas de forma recursiva. Si hay conflictos, gana la operación de etiquetado con la prioridad más alta. (Las prioridades se establecen mediante la `priority` propiedad opcional). Si las prioridades de dos operaciones son las mismas, gana la operación de etiquetado que se encuentre más cerca de la parte inferior del árbol de construcción. De forma predeterminada, aplicar una etiqueta tiene una prioridad de 100 (excepto en el caso de las etiquetas que se añaden directamente a un AWS CloudFormation recurso, que tienen una prioridad de 50). La prioridad predeterminada para eliminar una etiqueta es 200.

A continuación se aplica una etiqueta con una prioridad de 300 a un componente fijo.

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 });
```

Propiedades opcionales

Las etiquetas permiten ajustar con precisión la forma en `properties`que las etiquetas se aplican a los recursos o se eliminan de ellos. Todas las propiedades son opcionales.

`applyToLaunchedInstances`(Python:`apply_to_launched_instances`)

Disponible solo para add (). De forma predeterminada, las etiquetas se aplican a las instancias lanzadas en un grupo de Auto Scaling. Establezca esta propiedad en false para ignorar las instancias lanzadas en un grupo de Auto Scaling.

`includeResourceTypes`/`excludeResourceTypes`(Python:`include_resource_types`/`exclude_res`

Úselos para manipular las etiquetas solo en un subconjunto de recursos, en función de los tipos AWS CloudFormation de recursos. De forma predeterminada, la operación se aplica a todos los recursos del subárbol de construcción, pero esto se puede cambiar si se incluyen o excluyen determinados tipos de recursos. La exclusión tiene prioridad sobre la inclusión, si se especifican ambas opciones.

`priority`

Utilice esta opción para establecer la prioridad de esta operación con respecto a otras `Tags.add()` `Tags.remove()` operaciones. Los valores más altos tienen prioridad sobre los valores más bajos. El valor predeterminado es 100 para las operaciones de adición (50 para las etiquetas aplicadas directamente a AWS CloudFormation los recursos) y 200 para las operaciones de eliminación.

En el siguiente ejemplo, se aplica la etiqueta tagname con el valor y la prioridad 100 a los recursos del tipo de AWS::Xxx::Yyyla construcción. No aplica la etiqueta a las instancias lanzadas en un grupo de Auto Scaling de Amazon EC2 ni a recursos de este tipo. AWS::Xxx::Zzz (Son marcadores de posición para dos tipos de AWS CloudFormation recursos arbitrarios pero diferentes).

TypeScript

```
Tags.of(myConstruct).add('tagname', 'value', {
  applyToLaunchedInstances: false,
  includeResourceTypes: ['AWS::Xxx::Yyy'],
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 100,
});
```

JavaScript

```
Tags.of(myConstruct).add('tagname', 'value', {
  applyToLaunchedInstances: false,
  includeResourceTypes: ['AWS::Xxx::Yyy'],
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 100
});
```

Python

```
Tags.of(my_construct).add("tagname", "value",
    apply_to_launched_instances=False,
    include_resource_types=["AWS::Xxx::Yyy"],
    exclude_resource_types=["AWS::Xxx::Zzz"],
    priority=100)
```

Java

```
Tags.of(myConstruct).add("key", "value", TagProps.builder()
                .applyToLaunchedInstances(false)
                .includeResourceTypes(Arrays.asList("AWS::Xxx::Yyy"))
                .excludeResourceTypes(Arrays.asList("AWS::Xxx::Zzz"))
                .priority(100).build());
```

C#

```
Tags.Of(myConstruct).Add("tagname", "value", new TagProps
{
    ApplyToLaunchedInstances = false,
    IncludeResourceTypes = ["AWS::Xxx::Yyy"],
    ExcludeResourceTypes = ["AWS::Xxx::Zzz"],
    Priority = 100
});
```

En el siguiente ejemplo, se elimina la etiqueta tagname con prioridad 200 de los recursos de tipo de AWS::Xxx::Yyyla construcción, pero no de los recursos de 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
});
```

Ejemplo

En el siguiente ejemplo, se agrega la clave de etiqueta StackTypecon valor TheBesta cualquier recurso creado dentro del `Stack` nombre`MarketingSystem`. A continuación, lo vuelve a eliminar de todos los recursos, excepto de las subredes de VPC de Amazon EC2. El resultado es que solo las subredes tienen la etiqueta aplicada.

TypeScript

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

const app = new App();
const theBestStack = new Stack(app, 'MarketingSystem');

// Add a tag to all constructs in the stack
Tags.of(theBestStack).add('StackType', 'TheBest');

// Remove the tag from all resources except subnet resources
Tags.of(theBestStack).remove('StackType', {
  excludeResourceTypes: ['AWS::EC2::Subnet']
```

```
});
```

JavaScript

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

const app = new App();
const theBestStack = new Stack(app, 'MarketingSystem');

// Add a tag to all constructs in the stack
Tags.of(theBestStack).add('StackType', 'TheBest');

// Remove the tag from all resources except subnet resources
Tags.of(theBestStack).remove('StackType', {
  excludeResourceTypes: ['AWS::EC2::Subnet']
});
```

Python

```
from aws_cdk import App, Stack, Tags

app = App();
the_best_stack = Stack(app, 'MarketingSystem')

# Add a tag to all constructs in the stack
Tags.of(the_best_stack).add("StackType", "TheBest")

# Remove the tag from all resources except subnet resources
Tags.of(the_best_stack).remove("StackType",
    exclude_resource_types=["AWS::EC2::Subnet"])
```

Java

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

// Add a tag to all constructs in the stack
Tags.of(theBestStack).add("StackType", "TheBest");

// Remove the tag from all resources except subnet resources
Tags.of(theBestStack).remove("StackType", TagProps.builder()
        .excludeResourceTypes(Arrays.asList("AWS::EC2::Subnet"))
```

```
        .build());
```

C#

```
using Amazon.CDK;

var app = new App();
var theBestStack = new Stack(app, 'MarketingSystem');

// Add a tag to all constructs in the stack
Tags.Of(theBestStack).Add("StackType", "TheBest");

// Remove the tag from all resources except subnet resources
Tags.Of(theBestStack).Remove("StackType", new TagProps
{
    ExcludeResourceTypes = ["AWS::EC2::Subnet"]
});
```

Con el siguiente código se obtiene el mismo resultado. Considera qué enfoque (inclusión o exclusión) aclara tu intención.

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"]
});
```

Etiquetar construcciones individuales

`Tags.of(scope).add(key, value)`es la forma estándar de añadir etiquetas a los componentes fijos del. AWS CDK Su comportamiento de caminar por los árboles, que etiqueta de forma recursiva todos los recursos etiquetables dentro de un ámbito determinado, es casi siempre lo que se busca. Sin embargo, a veces es necesario etiquetar un constructo (o constructos) arbitrarios específicos.

Uno de estos casos implica la aplicación de etiquetas cuyo valor se deriva de alguna propiedad de la construcción que se está etiquetando. El enfoque de etiquetado estándar aplica de forma recursiva la misma clave y valor a todos los recursos coincidentes del ámbito. Sin embargo, en este caso el valor podría ser diferente para cada construcción etiquetada.

Las etiquetas se implementan mediante aspectos y la CDK llama al `visit()` método de la etiqueta para cada construcción dentro del ámbito especificado. `Tags.of(scope)` Podemos llamar `Tag.visit()` directamente para aplicar una etiqueta a una sola construcción.

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);
```

Puede etiquetar todos los componentes fijos de un ámbito, pero dejar que los valores de las etiquetas se deriven de las propiedades de cada componente. Para ello, escriba un aspecto y aplique la etiqueta en el `visit()` método del aspecto, tal y como se muestra en el ejemplo anterior. A continuación, añada el aspecto al ámbito deseado utilizando`Aspects.of(scope).add(aspect)`.

El siguiente ejemplo aplica una etiqueta a cada recurso de una pila que contiene la ruta del recurso.

TypeScript

```
class PathTagger implements cdk.IAspect {
  visit(node: IConstruct) {
    new cdk.Tag("aws-cdk-path", node.node.path).visit(node);
  }
}

stack = new MyStack(app);
cdk.Aspects.of(stack).add(new PathTagger())
```

JavaScript

```
class PathTagger {
  visit(node) {
    new cdk.Tag("aws-cdk-path", node.node.path).visit(node);
  }
}

stack = new MyStack(app);
cdk.Aspects.of(stack).add(new PathTagger())
```

Python

```
@jsii.implements(cdk.IAspect)
class PathTagger:
    def visit(self, node: IConstruct):
        cdk.Tag("aws-cdk-path", node.node.path).visit(node)
```

```
stack = MyStack(app)
cdk.Aspects.of(stack).add(PathTagger())
```

Java

```
final class PathTagger implements IAspect {
 public void visit(IConstruct node) {
  Tag.Builder.create("aws-cdk-path", node.getNode().getPath()).build().visit(node);
 }
}

stack stack = new MyStack(app);
Aspects.of(stack).add(new PathTagger());
```

C#

```
public class PathTagger : IAspect
{
    public void Visit(IConstruct node)
    {
        new Tag("aws-cdk-path", node.Node.Path).Visit(node);
    }
}

var stack = new MyStack(app);
Aspects.Of(stack).Add(new PathTagger);
```

Tip

La lógica del etiquetado condicional, que incluye las prioridades, los tipos de recursos, etc., está integrada en la `Tag` clase. Puede utilizar estas funciones al aplicar etiquetas a recursos arbitrarios; la etiqueta no se aplica si no se cumplen las condiciones. Además, la `Tag` clase solo etiqueta los recursos que se pueden etiquetar, por lo que no es necesario comprobar si una construcción es etiquetable antes de aplicar una etiqueta.

Activos

Los activos son archivos locales, directorios o imágenes de Docker que se pueden agrupar en AWS CDK bibliotecas y aplicaciones. Por ejemplo, un activo puede ser un directorio que contiene el código del controlador de una función. AWS Lambda Los activos pueden representar cualquier artefacto que la aplicación necesite para funcionar.

El siguiente vídeo tutorial proporciona una visión general completa de los activos de CDK y explica cómo puede utilizarlos en su infraestructura como código (IaC).

Explicación de los activos de CDK

Los activos se agregan a través de API que están expuestas por AWS estructuras específicas. Por ejemplo, cuando se define una construcción Lambda.function, la propiedad de código permite pasar un activo (directorio). `Function`usa activos para agrupar el contenido del directorio y usarlo para el código de la función. Del mismo modo, ecs. ContainerImage.fromAsset utiliza una imagen de Docker creada a partir de un directorio local al definir una definición de tarea de Amazon ECS.

Activos en detalle

Cuando hace referencia a un activo de su aplicación, el ensamblaje de nube que se sintetiza a partir de su aplicación incluye información de metadatos con instrucciones para la AWS CDK CLI. Las instrucciones incluyen dónde encontrar el activo en el disco local y qué tipo de agrupamiento se debe realizar en función del tipo de activo, como comprimir un directorio (zip) o crear una imagen de Docker.

AWS CDK Genera un hash de origen para los activos. Esto se puede utilizar en el momento de la construcción para determinar si el contenido de un activo ha cambiado.

De forma predeterminada, AWS CDK crea una copia del activo en el directorio de ensamblaje de la nube, que de forma predeterminada es`cdk.out`, bajo el hash de origen. De esta forma, el ensamblaje de la nube es autónomo, por lo que si se trasladó a un host diferente para su implementación, aún se puede implementar. Para obtener más información, consulte the section called “Conjuntos en la nube”.

Cuando AWS CDK implementa una aplicación que hace referencia a activos (ya sea directamente mediante el código de la aplicación o a través de una biblioteca), la AWS CDK CLI primero prepara y publica los activos en un bucket de Amazon S3 o en un repositorio de Amazon ECR. (El depósito o

repositorio de S3 se crea durante el arranque). Solo entonces se implementan los recursos definidos en la pila.

En esta sección se describen las API de bajo nivel disponibles en el marco.

Tipos de activos

AWS CDK Admite los siguientes tipos de activos:

Activos de Amazon S3

Se trata de archivos y directorios locales que AWS CDK carga en Amazon S3.

Imagen de Docker

Estas son imágenes de Docker que luego se AWS CDK cargan en Amazon ECR.

Estos tipos de activos se explican en las siguientes secciones.

Activos de Amazon S3

Puede definir los archivos y directorios locales como activos y los AWS CDK paquetes y cargarlos en Amazon S3 a través del módulo aws-s3-assets.

El siguiente ejemplo define un activo de directorio local y un activo de archivo.

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")),
})
```

En la mayoría de los casos, no es necesario utilizar directamente las API del `aws-s3-assets` módulo. Los módulos que admiten activos, por ejemplo`aws-lambda`, tienen métodos prácticos para que pueda usarlos. Para las funciones Lambda, el método estático fromAsset () permite especificar un directorio o un archivo.zip en el sistema de archivos local.

Ejemplo de función Lambda

Un caso de uso común es la creación de funciones Lambda con el código del controlador como un activo de Amazon S3.

El siguiente ejemplo usa un activo de Amazon S3 para definir un controlador de Python en el directorio `handler` local. También crea una función Lambda con el activo del directorio local como propiedad. `code` A continuación se presenta el código Python para el controlador.

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

El código de la AWS CDK aplicación principal debería tener el siguiente aspecto.

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
}
```

El `Function` método usa activos para agrupar el contenido del directorio y usarlo para el código de la función.

> **Tip**
>
> Los `.jar` archivos Java son archivos ZIP con una extensión diferente. Se cargan tal cual en Amazon S3, pero cuando se implementan como una función de Lambda, se extraen los archivos que contienen, algo que quizás no le interese. Para evitarlo, coloque el `.jar` archivo en un directorio y especifique ese directorio como activo.

Ejemplo de atributos en tiempo de despliegue

Los tipos de activos de Amazon S3 también exponen atributos de tiempo de implementación a los que se puede hacer referencia en AWS CDK bibliotecas y aplicaciones. El comando AWS CDK CLI cdk synth muestra las propiedades de los activos como AWS CloudFormation parámetros.

En el siguiente ejemplo, se utilizan atributos de tiempo de despliegue para pasar la ubicación de un activo de imagen a una función Lambda como variables de entorno. (El tipo de archivo no importa; la imagen PNG que se usa aquí es solo un ejemplo).

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(),
  },
})
```

Permisos

Si utiliza los activos de Amazon S3 directamente a través del módulo aws-s3-assets, las funciones, los usuarios o los grupos de IAM y necesita leer los activos en tiempo de ejecución, conceda permisos de IAM a esos activos mediante el método asset.grantRead.

El siguiente ejemplo otorga a un grupo de IAM permisos de lectura sobre un activo de archivo.

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)
```

Activos de imagen de Docker

AWS CDK Admite la agrupación de imágenes de Docker locales como activos a través del módulo. aws-ecr-assets

El siguiente ejemplo define una imagen de Docker que se crea localmente y se envía a Amazon ECR. Las imágenes se crean a partir de un directorio contextual de Docker local (con un Dockerfile) y la AWS CDK CLI o la canalización de CI/CD de la aplicación las cargan en Amazon ECR. Se puede hacer referencia a las imágenes de forma natural en su aplicación. 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")),
})
```

El `my-image` directorio debe incluir un Dockerfile. La AWS CDK CLI crea una imagen de Docker a partir de ella`my-image`, la envía a un repositorio de Amazon ECR y especifica el nombre del repositorio como AWS CloudFormation parámetro de la pila. Los tipos de activos de imagen de Docker muestran los atributos de tiempo de implementación a los que se puede hacer referencia en bibliotecas y aplicaciones. AWS CDK El comando AWS CDK CLI cdk synth muestra las propiedades de los activos como AWS CloudFormation parámetros.

Ejemplo de definición de tareas de Amazon ECS

Un caso de uso común es crear un Amazon ECS TaskDefinitionpara ejecutar contenedores de Docker. El siguiente ejemplo especifica la ubicación de un activo de imagen de Docker que AWS CDK se compila localmente y se envía a 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),
})
```

Ejemplo de atributos en tiempo de implementación

El siguiente ejemplo muestra cómo utilizar los atributos de tiempo de implementación `repository` y cómo `imageUri` crear una definición de tarea de Amazon ECS con el tipo de AWS Fargate lanzamiento. Ten en cuenta que la búsqueda en el repositorio de Amazon ECR requiere la etiqueta de la imagen, no su URI, por lo que la cortamos del final del URI del activo.

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()),
})
```

Ejemplo de argumentos de construcción

Puede proporcionar argumentos de compilación personalizados para el paso de compilación de Docker mediante la opción de propiedad `buildArgs` (Python:`build_args`) cuando la AWS CDK CLI compila la imagen durante la implementación.

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"),
  },
})
```

Permisos

Si utilizas un módulo compatible con los activos de imagen de Docker, como aws-ecs, te AWS CDK gestionará los permisos cuando utilices los activos de forma directa o directa. ContainerImage fromEcrRepository(Python:`from_ecr_repository`). Si utilizas los activos de imagen de Docker directamente, asegúrate de que el usuario principal tenga permisos para extraer la imagen.

En la mayoría de los casos, debe utilizar el método asset.repository.GrantPull (Python:. `grant_pull` Esto modifica la política de IAM del director para permitirle extraer imágenes de este repositorio. Si el principal que está extrayendo la imagen no está en la misma cuenta, o si se trata de un AWS servicio que no asume ninguna función en tu cuenta (por ejemplo AWS CodeBuild), debes conceder

permisos de extracción a la política de recursos y no a la política del principal. Usa asset.repository.addToResourceMétodo de política (Python:`add_to_resource_policy`) para conceder los permisos principales adecuados.

AWS CloudFormation metadatos de recursos

> **Note**
>
> Esta sección es relevante solo para los autores de constructos. En determinadas situaciones, las herramientas necesitan saber que un determinado recurso de CFN está utilizando un activo local. Por ejemplo, puede usar la AWS SAM CLI para invocar funciones de Lambda localmente con fines de depuración. Para obtener más información, consulte the section called “AWS SAM integración”.

Para habilitar estos casos de uso, las herramientas externas consultan un conjunto de entradas de metadatos en los recursos: AWS CloudFormation

- `aws:asset:path`— Señala la ruta local del activo.
- `aws:asset:property`— El nombre de la propiedad del recurso en la que se utiliza el activo.

Al utilizar estas dos entradas de metadatos, las herramientas pueden identificar qué recursos utiliza un recurso determinado y posibilitar experiencias locales avanzadas.

Para añadir estas entradas de metadatos a un recurso, utilice el método `asset.addResourceMetadata` (Python:`add_resource_metadata`).

Permisos

La biblioteca AWS Construct utiliza algunos modismos comunes y ampliamente implementados para administrar el acceso y los permisos. El módulo IAM le proporciona las herramientas que necesita para utilizar estas expresiones idiomáticas.

AWS CDK utiliza AWS CloudFormation para implementar cambios. Cada implementación involucra a un actor (ya sea un desarrollador o un sistema automatizado) que inicia una AWS CloudFormation implementación. Mientras lo hace, el actor asumirá una o más identidades de IAM (usuario o roles) y, si lo desea, le transferirá un rol. AWS CloudFormation

Si se utiliza AWS IAM Identity Center para autenticarse como usuario, el proveedor de inicio de sesión único proporciona credenciales de sesión de corta duración que le autorizan a actuar como una función de IAM predefinida. Para obtener información sobre cómo AWS CDK obtiene las AWS credenciales de la autenticación del Centro de Identidad de IAM, consulte Descripción de la autenticación del Centro de Identidad de IAM en la Guía de referencia de herramientas y SDK.AWS

Entidades principales

Una entidad principal de IAM es una AWS entidad autenticada que representa a un usuario, servicio o aplicación que puede llamar a las API. AWS La biblioteca AWS Construct permite especificar los directores de varias formas flexibles para permitirles acceder a sus recursos. AWS

En contextos de seguridad, el término «principal» se refiere específicamente a las entidades autenticadas, como los usuarios. Los objetos, como los grupos y las funciones, no representan a los usuarios (ni a otras entidades autenticadas), sino que los identifican indirectamente con el fin de conceder permisos.

Por ejemplo, si crea un grupo de IAM, puede conceder al grupo (y, por lo tanto, a sus miembros) acceso de escritura a una tabla de Amazon RDS. Sin embargo, el grupo en sí no es un principal porque no representa a una sola entidad (además, no puede iniciar sesión en un grupo).

En la biblioteca de IAM del CDK, las clases que identifican directa o indirectamente a los principales implementan la `IPrincipal`interfaz, lo que permite que estos objetos se usen indistintamente en las políticas de acceso. Sin embargo, no todos son principios desde el punto de vista de la seguridad. Estos objetos incluyen:

1. Recursos de IAM como `RoleUser`, y `Group`
2. Principios de servicio () `new iam.ServicePrincipal('service.amazonaws.com')`
3. Directores federados () `new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`
4. Responsables de cuentas (`new iam.AccountPrincipal('0123456789012')`)
5. Principales de usuario canónicos () `new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`
6. AWS Organizations directores () `new iam.OrganizationPrincipal('org-id')`
7. Principios de ARN arbitrarios () `new iam.ArnPrincipal(res.arn)`
8. Y confiar `iam.CompositePrincipal(principal1, principal2, ...)` en varios principios

Concesiones

Cada construcción que representa un recurso al que se puede acceder, como un bucket de Amazon S3 o una tabla de Amazon DynamoDB, tiene métodos que conceden acceso a otra entidad. Todos estos métodos tienen nombres que comienzan por grant.

Por ejemplo, los buckets de Amazon S3 tienen los métodos `grantRead` y `grantReadWrite` (Python:`grant_read`,`grant_read_write`) para permitir el acceso de lectura y lectura/escritura, respectivamente, desde una entidad al bucket. La entidad no necesita saber exactamente qué permisos de IAM de Amazon S3 son necesarios para realizar estas operaciones.

El primer argumento de un método de concesión es siempre del tipo iGrantable. Esta interfaz representa las entidades a las que se les pueden conceder permisos. Es decir, representa los recursos con funciones, como los objetos de IAM `RoleUser`, y`Group`.

También se pueden conceder permisos a otras entidades. Por ejemplo, más adelante en este tema, mostraremos cómo conceder a un CodeBuild proyecto acceso a un bucket de Amazon S3. Por lo general, el rol asociado se obtiene a través de una `role` propiedad de la entidad a la que se concede el acceso.

Los recursos que utilizan funciones de ejecución, por ejemplo`lambda.Function`, también se implementan`IGrantable`, por lo que puedes concederles acceso directamente en lugar de concederles acceso a su función. Por ejemplo, si `bucket` es un bucket de Amazon S3 y `function` es una función Lambda, el siguiente código concede a la función acceso de lectura al bucket.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

A veces, los permisos se deben aplicar mientras se implementa la pila. Uno de estos casos es cuando concedes a un recurso AWS CloudFormation personalizado acceso a otro recurso. El recurso personalizado se invocará durante la implementación, por lo que debe tener los permisos especificados en el momento de la implementación.

Otro caso es cuando un servicio verifica que la función que se le transfiere tiene aplicadas las políticas correctas. (Varios AWS servicios lo hacen para asegurarse de que no te olvides de configurar las políticas). En esos casos, la implementación podría fallar si los permisos se aplican demasiado tarde.

Para forzar la aplicación de los permisos de la concesión antes de que se cree otro recurso, puedes añadir una dependencia a la propia concesión, como se muestra aquí. Si bien el valor devuelto por los métodos de concesión suele descartarse, de hecho, todos los métodos de concesión devuelven un `iam.Grant` objeto.

TypeScript

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

JavaScript

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

Python

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

Java

```
Grant grant = bucket.grantRead(function);
```

```
CustomResource custom = new CustomResource(...);
custom.node.addDependency(grant);
```

C#

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

Roles

El paquete de IAM contiene una `Role` construcción que representa las funciones de IAM. El siguiente código crea un nuevo rol que confía en el servicio 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
});
```

Puede agregar permisos a un rol llamando al `addToPolicy` método del rol (Python:`add_to_policy`) y pasando una `PolicyStatement` que defina la regla que se va a agregar. La declaración se agrega a la política predeterminada del rol; si no tiene ninguna, se crea una.

El siguiente ejemplo agrega una declaración de `Deny` política al rol para las acciones `ec2:SomeAction` y `s3:AnotherAction` los recursos `bucket` y `otherRole` (Python:`other_role`), con la condición de que el servicio autorizado lo sea 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"
        }
    }
}));
```

En el ejemplo anterior, hemos creado una nueva `PolicyStatement` línea con la llamada `addToPolicy` (Python:`add_to_policy`). También puedes incluir una declaración de política

existente o una que hayas modificado. El PolicyStatementobjeto tiene numerosos métodos para añadir principios, recursos, condiciones y acciones.

Si utiliza una construcción que requiere un rol para funcionar correctamente, puede realizar una de las siguientes acciones:

- Transfiere un rol existente al crear una instancia del objeto de construcción.
- Deje que la construcción cree un nuevo rol para usted, confiando en el director de servicio apropiado. En el siguiente ejemplo, se utiliza una construcción de este tipo: un CodeBuild proyecto.

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 vez creado el objeto, el rol (ya sea el rol transferido o el predeterminado creado por la construcción) está disponible como propiedad`role`. Sin embargo, esta propiedad no está disponible

en los recursos externos. Por lo tanto, estas construcciones tienen un método `addToRolePolicy` (Python:`add_to_role_policy`).

El método no hace nada si la construcción es un recurso externo y, de lo contrario, llama al método `addToPolicy` (Python:`add_to_policy`) de la `role` propiedad. Esto le ahorra la molestia de tratar el caso indefinido de forma explícita.

El siguiente ejemplo demuestra lo siguiente:

TypeScript

```
// project is imported into the CDK application
const project = codebuild.Project.fromProjectName(this, 'Project', 'ProjectName');

// project is imported, so project.role is undefined, and this call has no effect
project.addToRolePolicy(new iam.PolicyStatement({
  effect: iam.Effect.ALLOW,   // ... and so on defining the policy
}));
```

JavaScript

```
// project is imported into the CDK application
const project = codebuild.Project.fromProjectName(this, 'Project', 'ProjectName');

// project is imported, so project.role is undefined, and this call has no effect
project.addToRolePolicy(new iam.PolicyStatement({
  effect: iam.Effect.ALLOW   // ... and so on defining the policy
}));
```

Python

```
# project is imported into the CDK application
project = codebuild.Project.from_project_name(self, 'Project', 'ProjectName')

# project is imported, so project.role is undefined, and this call has no effect
project.add_to_role_policy(iam.PolicyStatement(
  effect=iam.Effect.ALLOW,   # ... and so on defining the policy
)
```

Java

```
// project is imported into the CDK application
```

```
Project project = Project.fromProjectName(this, "Project", "ProjectName");

// project is imported, so project.getRole() is null, and this call has no effect
project.addToRolePolicy(PolicyStatement.Builder.create()
        .effect(Effect.ALLOW)   // .. and so on defining the policy
        .build());
```

C#

```
// project is imported into the CDK application
var project = Project.FromProjectName(this, "Project", "ProjectName");

// project is imported, so project.role is null, and this call has no effect
project.AddToRolePolicy(new PolicyStatement(new PolicyStatementProps
{
    Effect = Effect.ALLOW, // ... and so on defining the policy
}));
```

Políticas de recursos

Algunos recursos AWS, como los buckets de Amazon S3 y las funciones de IAM, también tienen una política de recursos. Estas construcciones tienen un `addToResourcePolicy` método (Python:`add_to_resource_policy`), que toma a `PolicyStatement` como argumento. Cada declaración de política agregada a una política de recursos debe especificar al menos un principal.

En el siguiente ejemplo, el bucket de Amazon S3 se `bucket` otorga un rol con el `s3:SomeAction` permiso para sí mismo.

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 }
}));
```

Uso de objetos de IAM externos

Si has definido un usuario, director, grupo o rol de IAM fuera de tu AWS CDK aplicación, puedes usar ese objeto de IAM en tu aplicación. AWS CDK Para ello, cree una referencia a él mediante su ARN o su nombre. (Use el nombre para los usuarios, grupos y roles). A continuación, la referencia devuelta se puede utilizar para conceder permisos o para elaborar declaraciones de políticas, como se ha explicado anteriormente.

- Para los usuarios, llame `User.fromUserArn()` o`User.fromUserName()`. `User.fromUserAttributes()`también está disponible, pero actualmente ofrece la misma funcionalidad que`User.fromUserArn()`.
- Para los principales, cree una instancia de un objeto. `ArnPrincipal`
- Para grupos, llama o. `Group.fromGroupArn()` `Group.fromGroupName()`
- Para funciones, llama `Role.fromRoleArn()` o`Role.fromRoleName()`.

Las políticas (incluidas las políticas gestionadas) se pueden utilizar de forma similar mediante los siguientes métodos. Puede utilizar las referencias a estos objetos en cualquier lugar donde se requiera una política de IAM.

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

> **Note**
>
> Como ocurre con todas las referencias a AWS recursos externos, no puedes modificar los objetos de IAM externos en tu aplicación CDK.

Contexto del tiempo de ejecución

Los valores de contexto son pares clave-valor que pueden asociarse a una aplicación, pila o constructo. Se pueden suministrar a tu aplicación desde un archivo (normalmente, uno de ellos `cdk.json` o `cdk.context.json` en el directorio del proyecto) o desde la línea de comandos.

El kit de herramientas CDK usa el contexto para almacenar en caché los valores recuperados de tu AWS cuenta durante la síntesis. Los valores incluyen las zonas de disponibilidad de su cuenta o los ID de Amazon Machine Image (AMI) disponibles actualmente para las instancias de Amazon EC2. Como estos valores los proporciona su AWS cuenta, pueden cambiar de una ejecución a otra de la aplicación de CDK. Esto los convierte en una fuente potencial de cambios no intencionados. El comportamiento de almacenamiento en caché del kit de herramientas de CDK «congela» estos valores para tu aplicación de CDK hasta que decidas aceptar los nuevos valores.

Imagine el siguiente escenario sin almacenamiento en caché contextual. Supongamos que especificó la «última versión de Amazon Linux» como AMI para sus instancias de Amazon EC2 y se publicó una nueva versión de esta AMI. Luego, la próxima vez que implemente su pila de CDK, las instancias ya implementadas usarán la AMI obsoleta («incorrecta») y deberán actualizarse. La actualización implicaría la sustitución de todas las instancias existentes por otras nuevas, lo que probablemente sería inesperado e indeseable.

En su lugar, la CDK registra las AMI disponibles de su cuenta en el `cdk.context.json` archivo del proyecto y utiliza el valor almacenado para futuras operaciones de síntesis. De esta forma, la lista de AMI deja de ser una fuente potencial de cambios. También puede estar seguro de que sus pilas siempre se sintetizarán en las mismas AWS CloudFormation plantillas.

No todos los valores de contexto son valores en caché de su entorno. AWS the section called "Banderas destacadas"también son valores de contexto. También puedes crear tus propios valores de contexto para que los usen tus aplicaciones o construcciones.

Las claves de contexto son cadenas. Los valores pueden ser de cualquier tipo compatible con JSON: números, cadenas, matrices u objetos.

> **Tip**
>
> Si tus construcciones crean sus propios valores de contexto, incorpora el nombre del paquete de la biblioteca en sus claves para que no entren en conflicto con los valores de contexto de otros paquetes.

Muchos valores de contexto están asociados a un AWS entorno concreto, y una aplicación de CDK determinada se puede implementar en más de un entorno. La clave de dichos valores incluye la AWS cuenta y la región, de modo que los valores de distintos entornos no entren en conflicto.

La siguiente clave de contexto ilustra el formato utilizado por la cuenta y la región AWS CDK, incluidas la cuenta y la región.

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

> **Important**
>
> Los valores de contexto almacenados en caché son gestionados por el AWS CDK y sus componentes, incluidos los componentes que pueda escribir. No añada ni modifique

los valores de contexto almacenados en caché editando los archivos manualmente. Sin embargo, puede resultar útil revisarlos de `cdk.context.json` vez en cuando para ver qué valores se están almacenando en caché. Los valores de contexto que no representan valores en caché deben almacenarse con la `context` clave de. `cdk.json` De esta forma, no se borrarán cuando se borren los valores en caché.

Fuentes de valores contextuales

Los valores de contexto se pueden proporcionar a tu AWS CDK aplicación de seis maneras diferentes:

- Automáticamente desde la AWS cuenta corriente.
- A través de la --context opción al cdk comando. (Estos valores son siempre cadenas).
- En el `cdk.context.json` archivo del proyecto.
- En la `context` clave del `cdk.json` archivo del proyecto.
- En la `context` clave de tu `~/.cdk.json` archivo.
- En tu AWS CDK aplicación usando el `construct.node.setContext()` método.

El archivo del proyecto `cdk.context.json` es donde se almacenan en AWS CDK caché los valores de contexto recuperados de tu AWS cuenta. Esta práctica evita cambios inesperados en las implementaciones cuando, por ejemplo, se introduce una nueva zona de disponibilidad. AWS CDK No escribe datos de contexto en ninguno de los demás archivos de la lista.

Important

Porque forman parte del estado de tu aplicación `cdk.json` y `cdk.context.json` deben estar sujetos al control de código fuente junto con el resto del código fuente de la aplicación. De lo contrario, las implementaciones en otros entornos (por ejemplo, una canalización de CI) podrían producir resultados incoherentes.

Los valores de contexto se basan en el constructo que los creó; son visibles para los constructos secundarios, pero no para los padres o hermanos. Los valores de contexto que establece el AWS CDK kit de herramientas (el cdk comando) se pueden configurar automáticamente, desde un archivo

o desde la opción. --context Los valores de contexto de estas fuentes se establecen implícitamente en la App construcción. Por lo tanto, son visibles para todas las construcciones de cada pila de la aplicación.

Tu aplicación puede leer un valor de contexto mediante el `construct.node.tryGetContext` método. Si la entrada solicitada no se encuentra en la construcción actual ni en ninguno de sus elementos principales, el resultado sí lo es`undefined`. (Como alternativa, el resultado podría ser el equivalente de su idioma, como `None` en Python).

Métodos de context

AWS CDK Admite varios métodos de contexto que permiten a AWS CDK las aplicaciones obtener información contextual del AWS entorno. Por ejemplo, puede obtener una lista de las zonas de disponibilidad que están disponibles en una AWS cuenta y región determinadas mediante el método stack.AvailabilityZones.

Los siguientes son los métodos de contexto:

HostedZone.fromLookup

: Obtiene las zonas alojadas de tu cuenta.

Stack.AvailabilityZones

: Obtiene las zonas de disponibilidad compatibles.

StringParameter.valueFromLookup

: Obtiene un valor del almacén de parámetros de Amazon EC2 Systems Manager de la región actual.

VPC. De Lookup

: Obtiene las Amazon Virtual Private Clouds existentes en sus cuentas.

LookupMachineImage

: Busca una imagen de máquina para usarla con una instancia de NAT en Amazon Virtual Private Cloud.

Si el valor de contexto requerido no está disponible, la AWS CDK aplicación notifica al CDK Toolkit que falta la información de contexto. A continuación, la CLI consulta la información en la AWS cuenta

corriente y almacena la información de contexto resultante en el `cdk.context.json` archivo. A continuación, vuelve a ejecutar la AWS CDK aplicación con los valores de contexto.

Visualización y administración del contexto

Utilice el cdk context comando para ver y gestionar la información del `cdk.context.json` archivo. Para ver esta información, utilice el cdk context comando sin ninguna opción. El resultado debería ser similar al siguiente.

```
Context found in cdk.json:

##############################################################################################
# # # Key                                                          # Value
                                    #
##############################################################################################
# 1 # availability-zones:account=123456789012:region=eu-central-1 # [ "eu-central-1a",
 "eu-central-1b", "eu-central-1c" ]   #
##############################################################################################
# 2 # availability-zones:account=123456789012:region=eu-west-1    # [ "eu-west-1a",
 "eu-west-1b", "eu-west-1c" ]            #
##############################################################################################

Run cdk context --reset KEY_OR_NUMBER to remove a context key. If it is a cached value,
 it will be refreshed on the next cdk synth.
```

Para eliminar un valor de contextocdk context --reset, ejecute especificando la clave o el número correspondiente al valor. En el siguiente ejemplo, se elimina el valor que corresponde a la segunda clave del ejemplo anterior. Este valor representa la lista de zonas de disponibilidad de la región 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.
```

Por lo tanto, si desea actualizar a la última versión de la AMI de Amazon Linux, utilice el ejemplo anterior para realizar una actualización controlada del valor de contexto y restablecerlo. A continuación, sintetice y vuelva a implementar la aplicación.

```
cdk synth
```

Para borrar todos los valores de contexto almacenados para tu aplicacióncdk context --clear, ejecuta lo siguiente.

```
cdk context --clear
```

Solo se `cdk.context.json` pueden restablecer o borrar los valores de contexto almacenados. No AWS CDK afecta a otros valores de contexto. Por lo tanto, para evitar que un valor de contexto se restablezca mediante estos comandos, puede copiar el valor en`cdk.json`.

AWS CDK Bandera del kit de herramientas **`--context`**

Usa la opción `--context` (`-c`abreviada) para pasar los valores de contexto de tiempo de ejecución a tu aplicación CDK durante la síntesis o la implementación.

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

Para especificar varios valores de contexto, repita la --context opción tantas veces como desee, proporcionando un par clave-valor cada vez.

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

Al sintetizar varias pilas, los valores de contexto especificados se transfieren a todas las pilas. Para proporcionar diferentes valores de contexto a pilas individuales, utilice diferentes claves para los valores o utilice varios comandos OR. cdk synth cdk deploy

Los valores de contexto que se pasan desde la línea de comandos son siempre cadenas. Si un valor suele ser de otro tipo, el código debe estar preparado para convertir o analizar el valor. Es posible que los valores contextuales que no sean cadenas se proporcionen de otras formas (por ejemplo, en`cdk.context.json`). Para asegurarse de que este tipo de valor funciona según lo esperado, confirme que el valor es una cadena antes de convertirlo.

Ejemplo

A continuación se muestra un ejemplo del uso de una Amazon VPC existente mediante AWS CDK el contexto.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import { Construct } from 'constructs';

export class ExistsVpcStack extends cdk.Stack {

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {

    super(scope, id, props);

    const vpcid = this.node.tryGetContext('vpcid');
    const vpc = ec2.Vpc.fromLookup(this, 'VPC', {
      vpcId: vpcid,
    });

    const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

    new cdk.CfnOutput(this, 'publicsubnets', {
      value: pubsubnets.subnetIds.toString(),
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const ec2 = require('aws-cdk-lib/aws-ec2');

class ExistsVpcStack extends cdk.Stack {

  constructor(scope, id, props) {

    super(scope, id, props);

    const vpcid = this.node.tryGetContext('vpcid');
    const vpc = ec2.Vpc.fromLookup(this, 'VPC', {
      vpcId: vpcid
    });

    const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});
```

```
    new cdk.CfnOutput(this, 'publicsubnets', {
      value: pubsubnets.subnetIds.toString()
    });
  }
}

module.exports = { ExistsVpcStack }
```

Python

```
import aws_cdk as cdk
import aws_cdk.aws_ec2 as ec2
from constructs import Construct

class ExistsVpcStack(cdk.Stack):

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

        super().__init__(scope, id, **kwargs)

        vpcid = self.node.try_get_context("vpcid")
        vpc = ec2.Vpc.from_lookup(self, "VPC", vpc_id=vpcid)

        pubsubnets = vpc.select_subnets(subnetType=ec2.SubnetType.PUBLIC)

        cdk.CfnOutput(self, "publicsubnets",
            value=pubsubnets.subnet_ids.to_string())
```

Java

```
import software.amazon.awscdk.CfnOutput;

import software.amazon.awscdk.services.ec2.Vpc;
import software.amazon.awscdk.services.ec2.VpcLookupOptions;
import software.amazon.awscdk.services.ec2.SelectedSubnets;
import software.amazon.awscdk.services.ec2.SubnetSelection;
import software.amazon.awscdk.services.ec2.SubnetType;
import software.constructs.Construct;

public class ExistsVpcStack extends Stack {
    public ExistsVpcStack(Construct context, String id) {
        this(context, id, null);
    }
```

```
    public ExistsVpcStack(Construct context, String id, StackProps props) {
        super(context, id, props);

        String vpcId = (String)this.getNode().tryGetContext("vpcid");
        Vpc vpc = (Vpc)Vpc.fromLookup(this, "VPC", VpcLookupOptions.builder()
                .vpcId(vpcId).build());

        SelectedSubnets pubSubNets = vpc.selectSubnets(SubnetSelection.builder()
                .subnetType(SubnetType.PUBLIC).build());

        CfnOutput.Builder.create(this, "publicsubnets")
                .value(pubSubNets.getSubnetIds().toString()).build();

    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.EC2;
using Constructs;

class ExistsVpcStack : Stack
{
    public ExistsVpcStack(Construct scope, string id, StackProps props) :
 base(scope, id, props)
    {
        var vpcId = (string)this.Node.TryGetContext("vpcid");
        var vpc = Vpc.FromLookup(this, "VPC", new VpcLookupOptions
        {
            VpcId = vpcId
        });

        SelectedSubnets pubSubNets = vpc.SelectSubnets([new SubnetSelection
        {
            SubnetType = SubnetType.PUBLIC
        }]);

        new CfnOutput(this, "publicsubnets", new CfnOutputProps {
            Value = pubSubNets.SubnetIds.ToString()
        });
    }
```

```
}
```

Puede usarlo cdk diff para ver los efectos de pasar un valor de contexto en la línea de comandos:

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

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

Los valores de contexto resultantes se pueden ver como se muestra aquí.

```
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"
    ]
  }
}
```

Banderas de características

AWS CDK Utiliza indicadores de funciones para habilitar posibles comportamientos disruptivos en una versión. Los indicadores se almacenan como the section called “Context” valores en `cdk.json` (o`~/.cdk.json`). Los cdk context --clear comandos cdk context --reset o no los eliminan.

Los indicadores de funciones están desactivados de forma predeterminada. Los proyectos existentes que no especifiquen el indicador seguirán funcionando como antes en las AWS CDK versiones posteriores. Los nuevos proyectos creados con indicadores cdk init incluyen indicadores que permiten todas las funciones disponibles en la versión en la que se creó el proyecto. `cdk.json`Edítelo para deshabilitar los indicadores cuyo comportamiento anterior prefiera. También puede añadir indicadores para activar nuevos comportamientos después de actualizar el AWS CDK.

Puede encontrar una lista de todos los indicadores de características actuales en el AWS CDK GitHub repositorio de `FEATURE_FLAGS.md`. Consulte la sección `CHANGELOG` de una versión determinada para ver una descripción de las nuevas marcas de características que se hayan agregado en esa versión.

Volviendo al comportamiento de la versión 1

En la versión 2 de CDK, los valores predeterminados de algunos indicadores de funciones se han cambiado con respecto a la versión 1. Puede volver a configurarlos para volver `false` a un comportamiento específico AWS CDK de la versión 1. Usa el `cdk diff` comando para inspeccionar los cambios en la plantilla sintetizada y ver si se necesita alguno de estos indicadores.

`@aws-cdk/core:newStyleStackSynthesis`

> Utilice el nuevo método de síntesis de pilas, que presupone recursos bootstrap con nombres conocidos. Requiere un arranque moderno, pero a su vez permite la CI/CD a través de CDK Pipelines y los despliegues multicuenta desde el primer momento.

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

Si su aplicación utiliza varias claves de API de Amazon API Gateway y las asocia a planes de uso.

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

Si su aplicación utiliza instancias de bases de datos o clústeres de bases de datos de Amazon RDS y especifica de forma explícita el identificador de los mismos.

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

Si su aplicación usa la política de seguridad TLS_V1_2_2019 con las distribuciones. Amazon CloudFront El CDK v2 usa la política de seguridad TLSv1.2_2021 de forma predeterminada.

`@aws-cdk/core:stackRelativeExports`

Si su aplicación utiliza varias pilas y usted hace referencia a los recursos de una pila en otra, esto determinará si se utiliza la ruta absoluta o relativa para generar las exportaciones. AWS CloudFormation

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

Si se establece en`false`, la CDK incluye metadatos al detectar si una función Lambda ha cambiado. Esto puede provocar errores de implementación cuando solo se han modificado los metadatos, ya que no se permiten versiones duplicadas. No es necesario revertir este indicador si ha realizado al menos un cambio en todas las funciones Lambda de la aplicación.

La sintaxis para revertir estos indicadores `cdk.json` se muestra aquí.

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

Aspectos

Los aspectos son una forma de aplicar una operación a todos los constructos de un ámbito determinado. El aspecto podría modificar las construcciones, por ejemplo, añadiendo etiquetas. O podría verificar algo sobre el estado de las construcciones, como asegurarse de que todos los depósitos estén cifrados.

Para aplicar un aspecto a un componente fijo y a todos los componentes fijos del mismo ámbito, utilice un `Aspects.of(SCOPE).add()`aspecto nuevo, como se muestra en el siguiente ejemplo.

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 Utiliza aspectos para etiquetar recursos, pero el marco también se puede utilizar para otros fines. Por ejemplo, puede usarlo para validar o cambiar los AWS CloudFormation recursos que definan para usted las construcciones de nivel superior.

Aspectos en detalle

Los aspectos emplean el patrón de visitantes. Un aspecto es una clase que implementa la siguiente interfaz.

TypeScript

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

JavaScript

JavaScript no tiene interfaces como característica del lenguaje. Por lo tanto, un aspecto es simplemente una instancia de una clase que tiene un `visit` método que acepta el nodo en el que se va a operar.

Python

Python no tiene interfaces como característica del lenguaje. Por lo tanto, un aspecto es simplemente una instancia de una clase que tiene un `visit` método que acepta el nodo en el que se va a operar.

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)
}
```

Al llamar`Aspects.of(SCOPE).add(...)`, la construcción añade el aspecto a una lista interna de aspectos. Puede obtener la lista con`Aspects.of(SCOPE)`.

Durante la fase de preparación, AWS CDK llama al `visit` método del objeto para la construcción y a cada uno de sus elementos secundarios en orden descendente.

El `visit` método es libre de cambiar cualquier elemento de la construcción. En lenguajes fuertemente tipados, convierta la construcción recibida en un tipo más específico antes de acceder a las propiedades o métodos específicos de la construcción.

Los aspectos no se propagan a través de los límites de los `Stage` constructos, porque `Stages` son autónomos e inmutables después de la definición. Aplica los aspectos en el propio componente `Stage` fijo (o en una parte inferior) si quieres que visiten los componentes fijos que se encuentran dentro del. `Stage`

Ejemplo

El siguiente ejemplo valida que todos los depósitos creados en la pila tengan activado el control de versiones. El aspecto añade una anotación de error a las construcciones que no superan la validación. Esto provoca un error en la synth operación e impide implementar el ensamblaje de nube resultante.

TypeScript

```
class BucketVersioningChecker implements IAspect {
  public visit(node: IConstruct): void {
    // See that we're dealing with a CfnBucket
    if (node instanceof s3.CfnBucket) {

      // Check for versioning property, exclude the case where the property
      // can be a token (IResolvable).
      if (!node.versioningConfiguration
        || (!Tokenization.isResolvable(node.versioningConfiguration)
            && node.versioningConfiguration.status !== 'Enabled')) {
        Annotations.of(node).addError('Bucket versioning is not enabled');
      }
    }
  }
}

// Later, apply to the stack
```

```
Aspects.of(stack).add(new BucketVersioningChecker());
```

JavaScript

```
class BucketVersioningChecker {
   visit(node) {
    // See that we're dealing with a CfnBucket
    if ( node instanceof s3.CfnBucket) {

      // Check for versioning property, exclude the case where the property
      // can be a token (IResolvable).
      if (!node.versioningConfiguration
        || !Tokenization.isResolvable(node.versioningConfiguration)
            && node.versioningConfiguration.status !== 'Enabled')) {
        Annotations.of(node).addError('Bucket versioning is not enabled');
      }
    }
  }
}

// Later, apply to the stack
Aspects.of(stack).add(new BucketVersioningChecker());
```

Python

```
@jsii.implements(cdk.IAspect)
class BucketVersioningChecker:

  def visit(self, node):
    # See that we're dealing with a CfnBucket
    if isinstance(node, s3.CfnBucket):

      # Check for versioning property, exclude the case where the property
      # can be a token (IResolvable).
      if (not node.versioning_configuration or
              not Tokenization.is_resolvable(node.versioning_configuration)
                  and node.versioning_configuration.status != "Enabled"):
          Annotations.of(node).add_error('Bucket versioning is not enabled')

# Later, apply to the stack
Aspects.of(stack).add(BucketVersioningChecker())
```

Java

```
public class BucketVersioningChecker implements IAspect
{
    @Override
    public void visit(Construct node)
    {
        // See that we're dealing with a CfnBucket
        if (node instanceof CfnBucket)
        {
            CfnBucket bucket = (CfnBucket)node;
            Object versioningConfiguration = bucket.getVersioningConfiguration();
            if (versioningConfiguration == null ||
                    !Tokenization.isResolvable(versioningConfiguration.toString())
 &&
                    !versioningConfiguration.toString().contains("Enabled"))
                Annotations.of(bucket.getNode()).addError("Bucket versioning is not
 enabled");
        }
    }
}


// Later, apply to the stack
Aspects.of(stack).add(new BucketVersioningChecker());
```

C#

```
class BucketVersioningChecker : Amazon.Jsii.Runtime.Deputy.DeputyBase, IAspect
{
    public void Visit(IConstruct node)
    {
        // See that we're dealing with a CfnBucket
        if (node is CfnBucket)
        {
            var bucket = (CfnBucket)node;
            if (bucket.VersioningConfiguration is null ||
                    !Tokenization.IsResolvable(bucket.VersioningConfiguration) &&
                    !bucket.VersioningConfiguration.ToString().Contains("Enabled"))
                Annotations.Of(bucket.Node).AddError("Bucket versioning is not
 enabled");
        }
    }
```

```
}

// Later, apply to the stack
Aspects.Of(stack).add(new BucketVersioningChecker());
```

Cómo empezar con el AWS CDK

Comience con la AWS Cloud Development Kit (AWS CDK) instalando AWS CDK CLI y creando su primera aplicación CDK.

Temas

Requisitos previos

Recursos recomendados

Antes de empezar con el AWS CDK, recomendamos tener conocimientos básicos de lo siguiente:

- Una introducción a la AWS CDK. Para obtener más información, consulte ¿Qué es el AWS CDK?
- Conceptos básicos detrás de AWS CDK. Para obtener más información, consulte AWS CDK conceptos.
- Lo Servicios de AWS que quieres gestionar con el AWS CDK.
- AWS Identity and Access Management. Para obtener más información, consulte ¿Qué es IAM? y ¿Qué es el Centro de identidad de IAM?
- AWS CloudFormation ya que AWS CDK utiliza el AWS CloudFormation servicio para aprovisionar los recursos creados en el CDK. Para obtener más información, consulte ¿Qué es AWS CloudFormation?
- El lenguaje de programación compatible que planea usar con el. AWS CDK

Prepare su entorno local

Todos los AWS CDK desarrolladores, independientemente del idioma que prefieran, necesitan la versión Node.js14.15.0 o una versión posterior. Todos los lenguajes de programación compatibles utilizan el mismo backend, que se ejecuta en él. Node.js Recomendamos una versión con soporte activo a largo plazo. Es posible que su organización tenga una recomendación diferente.

> **Important**
>
> Las versiones 13.0.0 a 13.6.0 de Node.js no son compatibles con él AWS CDK debido a problemas de compatibilidad con sus dependencias.

Otros requisitos previos dependen del idioma en el que se desarrollen AWS CDK las aplicaciones y son los siguientes.

TypeScript

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

JavaScript

Sin requisitos adicionales

Python

- Python 3.7 o posterior, incluidos `pip` y `virtualenv`

Java

- Java Development Kit (JDK) 8 (también conocido como 1.8) o posterior
- Apache Maven 3.5 o posterior

Se recomienda el IDE de Java (utilizamos Eclipse en algunos ejemplos de esta guía). El IDE debe poder importar proyectos de Maven. Asegúrese de que su proyecto esté configurado para usar Java 1.8. Establezca la variable de entorno JAVA_HOME en la ruta en la que instaló el JDK.

C#

.NET Core 3.1 o posterior, o .NET 6.0 o posterior.

Se recomienda Visual Studio 2019 (cualquier edición) o Visual Studio Code.

Go

Utilice la versión 1.1.8 o una versión posterior.

Para obtener información más detallada, consulte la sección de requisitos previos para su idioma:

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

> **Depreciación de idiomas de terceros**
>
> Solo se admite la versión en cada idioma hasta el final de su vida útil y está sujeta a cambios con previo aviso. EOL

Paso 1: Crea una Cuenta de AWS

Si es la primera vez que lo usa AWS, debe registrarse Cuenta de AWS y crear un usuario administrativo. Para obtener más información, consulte Cómo configurar IAM en la Guía del usuario de IAM.

Cuando interactúa con AWS, especifica sus credenciales de AWS seguridad para comprobar quién es y si tiene permiso para acceder a los recursos que solicita. AWS utiliza las credenciales de seguridad para autenticar y autorizar sus solicitudes. Para obtener más información, consulte las credenciales AWS de seguridad en la Guía del usuario de IAM.

Paso 2: Configurar el acceso mediante programación

Al desarrollar AWS CDK en su entorno local, confiará en ellos AWS CDK CLI para interactuar con sus AWS recursos Servicios de AWS y administrarlos. Para utilizar el AWS CDK CLI, debe configurar el acceso programático. Para obtener más información sobre las diferentes formas de configurar el acceso mediante programación, consulte Autenticación y acceso en la Guía de referencia de herramientas y AWS SDK.

Para los nuevos usuarios a los que su empresa no les haya proporcionado un método de autenticación, les recomendamos que lo utilicen. AWS IAM Identity Center Este método incluye instalar el AWS Command Line Interface (AWS CLI) y usarlo para configurar e iniciar sesión en el portal de AWS acceso. Para configurar el acceso mediante programación mediante el Centro de identidad de IAM, consulte la autenticación del Centro de identidad de IAM en la Guía de referencia de herramientas y AWS SDK. Una vez finalizado, su entorno debe contener los siguientes elementos:

- El AWS CLI, que se utiliza para iniciar una sesión en el portal de AWS acceso antes de ejecutar la aplicación.
- Un AWS`config`archivo compartido que tiene un `[default]` perfil con un conjunto de valores de configuración a los que se puede hacer referencia desde AWS CDK. Para encontrar la ubicación de este archivo, consulte Ubicación de los archivos compartidos en la Guía de referencia de las herramientas y los SDK de AWS.
- El archivo compartido de `config` establece la configuración de `region`. Esto establece los AWS CDK usos predeterminados Región de AWS para AWS las solicitudes.
- AWS CDK Utiliza la configuración del proveedor de tokens de SSO del perfil para adquirir las credenciales antes de enviar las solicitudes a AWS ellas. El `sso_role_name` valor, que es un rol de IAM conectado a un conjunto de permisos del Centro de Identidad de IAM, debería permitir el acceso a los Servicios de AWS utilizados en la aplicación.

  El siguiente archivo `config` de ejemplo muestra la configuración de un perfil predeterminado con el proveedor de token de SSO. La configuración `sso_session` del perfil hace referencia a la sección llamada `sso-session`. La `sso-session` sección contiene la configuración para iniciar una sesión en el portal de AWS acceso.

```
[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
```

Inicie una sesión en el portal de AWS acceso

Antes de acceder Servicios de AWS, necesita una sesión activa en el portal de AWS acceso para AWS CDK poder utilizar la autenticación del Centro de Identidad de IAM para resolver las credenciales. En función de la duración de las sesiones configuradas, el acceso eventualmente caducará y AWS CDK se producirá un error de autenticación. Ejecute el siguiente comando AWS CLI para iniciar sesión en el portal de AWS acceso.

```
aws sso login
```

Si la configuración de su proveedor de token de SSO utiliza un perfil con nombre en lugar del perfil predeterminado, el comando es`aws sso login --profile NAME`. Especifique también este perfil al emitir cdk comandos mediante la --profile opción o la variable de `AWS_PROFILE` entorno.

Para comprobar si ya tiene una sesión activa, ejecute el siguiente AWS CLI comando.

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

La respuesta a este comando debe indicar la cuenta y el conjunto de permisos del Centro de identidades de IAM configurados en el archivo compartido `config`.

> **Note**
>
> Si ya tiene una sesión activa en el portal de AWS acceso y la ejecuta`aws sso login`, no tendrá que proporcionar credenciales.
> Es posible que el proceso de inicio de sesión le pida que permita el AWS CLI acceso a sus datos. Dado que AWS CLI se basa en el SDK para Python, los mensajes de permiso pueden contener variaciones del `botocore` nombre.

Paso 3: Instala el AWS CDKCLI

Instálelo AWS CDK CLI globalmente mediante el siguiente comando de Node Package Manager.

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

> **Note**
>
> Si aparece un error de permiso y tienes acceso de administrador a tu sistema, inténtalosudo `npm install -g aws-cdk`.

Ejecute el siguiente comando para comprobar que la instalación se ha realizado correctamente. AWS CDK CLIDebería mostrar el número de versión:

```
cdk --version
```

Si recibe un mensaje de error, intente desinstalarlo AWS CDK CLI ejecutando lo siguiente:

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

A continuación, repita los pasos para volver a instalar el AWS CDK CLI.

Si sigue apareciendo un error, elimine la `node-modules` carpeta del proyecto actual y también de la `node-modules` carpeta global. Para localizar esta carpeta, ejecutenpm `config get prefix`.

AWS CDK CLIObtendrá las credenciales de seguridad de las fuentes que configuró en los pasos anteriores.

> **Note**
>
> El kit de herramientas de CDK v2 funciona con los proyectos de CDK v1 existentes. Sin embargo, no puede inicializar nuevos proyectos del CDK v1. Comprueba the section called “Nuevos requisitos previos” si necesitas poder hacerlo.

Paso 4: Inicie su entorno

Se debe iniciar cada AWS entorno en el que planea implementar recursos.

Para arrancar, ejecute lo siguiente:

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

Tip

Si no tienes tu número de AWS cuenta a mano, puedes obtenerlo en. AWS Management Console O bien, si lo tiene AWS CLI instalado, el siguiente comando muestra la información de su cuenta predeterminada, incluido el número de cuenta.

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

Si creó perfiles con nombre en su AWS configuración local, puede usar la `--profile` opción para mostrar la información de la cuenta de un perfil específico. El siguiente ejemplo muestra cómo mostrar la información de la cuenta del perfil de producción.

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

Para mostrar la región predeterminada, utiliceaws `configure get`.

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

AWS CDK Herramientas opcionales

AWS Toolkit for Visual Studio CodeEs un complemento de código abierto para Visual Studio Code que le ayuda a crear, depurar e implementar aplicaciones en AWS ellas. El kit de herramientas proporciona una experiencia integrada para desarrollar AWS CDK aplicaciones. Incluye la función AWS CDK Explorer para enumerar sus AWS CDK proyectos y explorar los distintos componentes de la aplicación CDK. Instale el complemento y obtenga más información sobre el uso del AWS CDK Explorador.

Siguientes pasos

Ahora que lo has instalado AWS CDK CLI, úsalo para crear tu primera AWS CDK aplicación.

Para obtener más información sobre el uso de AWS CDK en su lenguaje de programación preferido, consulteTrabajar con AWS CDK los lenguajes de programación compatibles.

AWS CDK Se trata de un proyecto de código abierto. Para contribuir, consulte Contribuir al AWS Cloud Development Kit (AWS CDK).

Más información

Para obtener más información sobre el AWS CDK, consulte lo siguiente:

- Taller CDK: taller práctico en profundidad.
- Referencia de API: explore las construcciones disponibles para las Servicios de AWS que utilizará.
- Construct Hub: encuentra construcciones de la comunidad de CDK.
- AWS CDK ejemplos: explore ejemplos de código de AWS CDK proyectos.

Tu primera AWS CDK aplicación

Comience a utilizarla AWS Cloud Development Kit (AWS CDK) creando su primera aplicación CDK.

Antes de comenzar este tutorial, le recomendamos que complete lo siguiente:

- Consulte ¿Qué es el AWS CDK? para obtener una introducción al AWS CDK.
- Consulte AWS CDK conceptos para aprender los conceptos básicos del AWS CDK.
- Consulte los requisitos previos y los pasos AWS CDK de configuración enCómo empezar con el AWS CDK.

Temas

Acerca de este tutorial

En este tutorial, crearás e implementarás una AWS CDK aplicación sencilla. Esta aplicación contiene una pila con un único recurso de bucket de Amazon Simple Storage Service (Amazon S3). A través de este tutorial, aprenderá lo siguiente:

- La estructura de un AWS CDK proyecto.
- Cómo crear una AWS CDK aplicación.
- Cómo usar la biblioteca AWS Construct para definir aplicaciones, pilas y AWS recursos.
- Cómo usar la CDK CLI para sintetizar, diferenciar, implementar y eliminar tu aplicación de CDK.
- Cómo modificar y volver a implementar su aplicación CDK para actualizar los recursos implementados.

El flujo de trabajo AWS CDK de desarrollo estándar consta de los siguientes pasos:

1. Cree su AWS CDK aplicación: aquí, utilizará una plantilla proporcionada por AWS CDK CLI.
2. Defina sus pilas y recursos: utilice construcciones para definir sus pilas y AWS recursos dentro de su aplicación.
3. Cree su aplicación: este paso es opcional. Realiza este paso AWS CDK CLI automáticamente si es necesario. Se recomienda realizar este paso para identificar los errores de sintaxis y tipo.
4. Sintetiza tus pilas: este paso crea una AWS CloudFormation plantilla para cada pila de tu aplicación. Este paso es útil para identificar los errores lógicos en los recursos definidos AWS .
5. Implemente su aplicación: impleméntela en su AWS entorno mediante AWS CloudFormation el aprovisionamiento de sus recursos. Durante la implementación, identificará cualquier problema de permisos con su aplicación.

Siguiendo un flujo de trabajo típico, volverás atrás y repetirás los pasos anteriores para modificar o depurar tu aplicación.

Te recomendamos que utilices el control de versiones para tus AWS CDK proyectos.

Paso 1: Crea la aplicación

Una aplicación CDK debe estar en su propio directorio, con sus propias dependencias de módulos locales. En tu máquina de desarrollo, crea un directorio nuevo. El siguiente es un ejemplo que crea un `hello-cdk` directorio nuevo:

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

> **Important**
>
> Asegúrese de asignar un nombre al directorio `hello-cdk` del proyecto exactamente como se muestra aquí. La plantilla AWS CDK del proyecto usa el nombre del directorio para nombrar las cosas del código generado. Si utilizas un nombre diferente, el código de este tutorial no funcionará.

A continuación, desde el nuevo directorio, inicialice la aplicación mediante el cdk init comando. Especifique la `app` plantilla y el lenguaje de programación preferido con la `--language` opción. A continuación, se muestra un ejemplo:

TypeScript

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

JavaScript

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

Python

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

Una vez creada la aplicación, introduce también los dos comandos siguientes. Estos activan el entorno virtual Python de la aplicación e instalan las dependencias AWS CDK principales.

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

Java

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

Si utilizas un IDE, ahora puedes abrir o importar el proyecto. En Eclipse, por ejemplo, elija Archivo > Importar > Maven > Proyectos Maven existentes. Asegúrese de que los ajustes del proyecto estén configurados para utilizar Java 8 (1.8).

C#

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

Si utiliza Visual Studio, abra el archivo de la solución en el `src` directorio.

Go

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

Una vez creada la aplicación, introduzca también el siguiente comando para instalar los módulos de AWS Construct Library que requiere la aplicación.

```
go get
```

El cdk init comando crea varios archivos y carpetas dentro del `hello-cdk` directorio para ayudarte a organizar el código fuente de la AWS CDK aplicación. En conjunto, esto se denomina AWS CDK proyecto. Tómate un momento para explorar el proyecto CDK.

Si lo ha Git instalado, cada proyecto que cree con él también cdk init se inicializa como un Git repositorio.

Paso 2: Crea la aplicación

En la mayoría de los entornos de programación, el código se crea o compila después de realizar cambios. Con el, esto no es necesario, AWS CDK ya que el CDK CLI realizará este paso automáticamente. Sin embargo, puedes seguir compilando manualmente si quieres atrapar errores de sintaxis y de tipo. A continuación, se muestra un ejemplo:

TypeScript

```
npm run build
```

JavaScript

No es necesario realizar ningún paso de compilación.

Python

No es necesario ningún paso de construcción.

Java

```
mvn compile -q
```

O presiona Control-B en Eclipse (otros IDE de Java pueden variar)

C#

```
dotnet build src
```

O bien, presione F6 en Visual Studio

Go

```
go build
```

Paso 3: Enumere las pilas de la aplicación

Comprueba que la aplicación se ha creado correctamente haciendo una lista de las pilas de la aplicación. Ejecute lo siguiente:

```
cdk ls
```

Debería `HelloCdkStack` mostrarse el resultado. Si no ve este resultado, compruebe que está en el directorio de trabajo correcto del proyecto e inténtelo de nuevo. Si sigues sin ver tu pila, repite el proceso the section called “Paso 1: Crea la aplicación” e inténtalo de nuevo.

Paso 4: Añadir un bucket de Amazon S3

En este punto, la aplicación de CDK contiene una sola pila. A continuación, definirá un recurso de bucket de Amazon Simple Storage Service (Amazon S3) dentro de su pila. Para ello, importará y utilizará la construcción `Bucket` L2 de la biblioteca de AWS construcciones.

Modifique su aplicación de CDK importando la `Bucket` construcción y definiendo su recurso de bucket de Amazon S3. A continuación, se muestra un ejemplo:

TypeScript

En `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

En `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

En `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

En `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#

En `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

En `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
}
```

Echemos un vistazo más de cerca a la `Bucket` construcción. Como todas las construcciones, la `Bucket` clase utiliza tres parámetros:

- ámbito: define la `Stack` clase como la matriz de la `Bucket` construcción. Todas las construcciones que definen AWS los recursos se crean dentro del ámbito de una pila. Puede definir componentes fijos dentro de otros componentes, creando una jerarquía (árbol). Aquí, y en la mayoría de los casos, el ámbito está `this` (`self`dentro). Python
- ID: el ID lógico de `Bucket` la AWS CDK aplicación. Este ID, más un hash basado en la ubicación del depósito dentro de la pila, identifica al depósito de forma exclusiva durante la implementación. AWS CDK También hace referencia a este ID cuando actualizas la construcción en tu aplicación y la vuelves a implementar para actualizar el recurso implementado. Aquí, tu ID lógico es`MyFirstBucket`. Los cubos también pueden tener un nombre, que se especifica con la `bucketName` propiedad. Es diferente del identificador lógico.
- props: conjunto de valores que definen las propiedades del depósito. Aquí ha definido la `versioned` propiedad como`true`, que permite el control de versiones de los archivos del depósito.

  Los accesorios se representan de forma diferente en los idiomas compatibles con. AWS CDK

  - En TypeScript y JavaScript, `props` es un argumento único y se pasa un objeto que contiene las propiedades deseadas.
  - En Python, los props se pasan como argumentos de palabras clave.

- En Java, se proporciona un constructor para pasar los accesorios. Hay dos: uno para `BucketProps` y otro para `Bucket` que puedas construir la construcción y su objeto de apoyo en un solo paso. Este código usa este último.
- En C#, se crea una instancia de un `BucketProps` objeto mediante un inicializador de objetos y se pasa como tercer parámetro.

Si los accesorios de una construcción son opcionales, puedes omitir el parámetro por completo. `props`

Todos los constructos utilizan los mismos tres argumentos, por lo que es fácil mantenerse orientado a medida que se aprenden otros nuevos. Y como es de esperar, puede subclasificar cualquier construcción para ampliarla según sus necesidades o si desea cambiar sus valores predeterminados.

Paso 5: Sintetizar una plantilla AWS CloudFormation

Sintetiza una AWS CloudFormation plantilla para la aplicación, de la siguiente manera:

```
cdk synth
```

Si tu aplicación contiene más de una pila, debes especificar qué pilas quieres sintetizar. Como la aplicación contiene una sola pila, el CDK detecta CLI automáticamente la pila que se va a sintetizar.

Si no la ejecutas`cdk synth`, la CDK CLI realizará este paso automáticamente cuando la implementes. Sin embargo, le recomendamos que ejecute este paso antes de cada implementación.

> **Tip**
>
> Si recibe un error como`--app is required ...`, por ejemplo, compruebe el directorio desde el que está ejecutando los CLI comandos del CDK. Deberías estar en el directorio principal de tu aplicación.

El `cdk synth` comando ejecuta tu aplicación. Esto crea una AWS CloudFormation plantilla para cada pila de tu aplicación. El CDK CLI mostrará una versión con formato YAML de la plantilla en la línea de comandos y guardará una versión con formato JSON de la plantilla en el directorio. `cdk.out` A continuación, se muestra un fragmento del resultado de la línea de comandos en el que se muestra el bucket que se está definiendo en la plantilla: AWS CloudFormation

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

> **Note**
>
> Cada plantilla generada contiene un `AWS::CDK::Metadata` recurso de forma predeterminada. El AWS CDK equipo utiliza estos metadatos para obtener información sobre el AWS CDK uso y encontrar formas de mejorarlo. Para obtener más información, incluida la forma de excluirse de los informes de versiones, consulte Informes de versiones.

La plantilla generada se puede implementar a través de la AWS CloudFormation consola o de cualquier herramienta de AWS CloudFormation implementación. También puede usar el CDK CLI para realizar la implementación. En el siguiente paso, utilizará la CDK CLI para realizar la implementación.

Paso 6: Implemente su pila

Para implementar su pila de CDK para AWS CloudFormation utilizar la CDKCLI, ejecute lo siguiente:

```
cdk deploy
```

> **Important**
>
> Debe realizar un arranque único de su entorno antes de la implementación. AWS Para obtener instrucciones, consulte Bootstrap your environment.

Del mismo `cdk synth` modo, no es necesario especificar la AWS CDK pila, ya que la aplicación contiene una sola pila.

Si el código tiene implicaciones de seguridad, el CDK CLI generará un resumen. Deberá confirmarlas para continuar con la implementación. La aplicación de este tutorial no tiene estas implicaciones.

Tras ejecutarse`cdk deploy`, la CDK CLI muestra información sobre el progreso a medida que se despliega la pila. Cuando haya terminado, podrá ir a la AWS CloudFormation consola para ver su `HelloCdkStack` pila. También puede ir a la consola de Amazon S3 para ver su `MyFirstBucket` recurso.

¡Enhorabuena! Implementó su primera pila con AWS CDK. A continuación, modificará la aplicación y la volverá a implementar para actualizar el recurso.

Paso 7: Modifica tu aplicación

En este paso, modificará su bucket de Amazon S3 configurándolo para que se elimine automáticamente cuando se elimine su pila. Esta modificación implica cambiar la `RemovalPolicy` propiedad del bucket. También configurará la `autoDeleteObjects` propiedad para configurar el CDK de forma CLI que elimine los objetos del depósito antes de destruirlo. De forma predeterminada, AWS CloudFormation no elimina los buckets de Amazon S3 que contienen objetos.

Utilice el siguiente ejemplo para modificar el recurso:

TypeScript

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

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

JavaScript

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

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

Python

Actualizar `hello_cdk/hello_cdk_stack.py`.

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

Java

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

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

C#

Actualizar `src/HelloCdk/HelloCdkStack.cs`.

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

Go

Actualizar `hello-cdk.go`.

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

Actualmente, los cambios en el código no han realizado ninguna actualización directa en el recurso de bucket de Amazon S3 implementado. Su código define el estado deseado de su recurso. Para

modificar el recurso desplegado, utilizará la CDK CLI para sintetizar el estado deseado en una nueva AWS CloudFormation plantilla. A continuación, implementará la nueva AWS CloudFormation plantilla como un conjunto de cambios. Los conjuntos de cambios solo realizan los cambios necesarios para alcanzar el nuevo estado deseado.

Para ver estos cambios, utilice el `cdk diff` comando. Ejecute lo siguiente:

```
cdk diff
```

El CDK CLI consulta tu Cuenta de AWS cuenta para obtener la AWS CloudFormation plantilla más reciente de la `HelloCdkStack` pila. A continuación, compara la plantilla más reciente con la plantilla que acaba de sintetizar en tu aplicación. El resultado debe tener el siguiente aspecto.

```
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
```

Esta diferencia tiene cuatro secciones:

- Cambios en la declaración de IAM y cambios en la política de IAM: estos cambios de permisos se deben a que usted configuró la `AutoDeleteObjects` propiedad en su bucket de Amazon S3. La función de eliminación automática utiliza un recurso personalizado para eliminar los objetos del depósito antes de que se elimine el propio depósito. Los objetos de IAM permiten que el código del recurso personalizado acceda al depósito.
- Parámetros: AWS CDK utiliza estas entradas para localizar el activo de AWS Lambda función del recurso personalizado.
- Recursos: los recursos nuevos y modificados de esta pila. Podemos ver cómo se añaden los objetos de IAM mencionados anteriormente, el recurso personalizado y su función Lambda asociada. También podemos ver que el bucket `DeletionPolicy` y los `UpdateReplacePolicy` atributos se están actualizando. Esto permite eliminar el depósito junto con la pila y sustituirlo por uno nuevo.

Puede que te des cuenta de que lo especificamos `RemovalPolicy` en nuestra AWS CDK aplicación, pero tenemos una `DeletionPolicy` propiedad en la AWS CloudFormation plantilla resultante. Esto se debe a que AWS CDK usa un nombre diferente para la propiedad. El AWS CDK valor predeterminado es conservar el depósito cuando se elimina la pila, mientras que el AWS CloudFormation valor predeterminado es eliminarlo. Para obtener más información, consulte the section called “Políticas de retirada”.

Para ver tu nueva AWS CloudFormation plantilla, puedes ejecutarcdk synth. Al realizar algunos cambios en la aplicación CDK, la nueva AWS CloudFormation plantilla ahora incluye muchas líneas de código adicionales en comparación con la AWS CloudFormation plantilla original.

A continuación, implementa tu aplicación ejecutando lo siguiente:

```
cdk deploy
```

Le AWS CDK informará sobre los cambios en la política de seguridad que ya hemos visto en el diferencial. yIngresa para aprobar los cambios e implementar la pila actualizada. El CDK CLI desplegará su pila para realizar los cambios que desee. El siguiente es un ejemplo de salida:

```
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
```

Paso 8: Destruir los recursos de la aplicación

Ahora que has completado este tutorial, puedes eliminar la AWS CloudFormation pila desplegada y todos los recursos asociados a ella. Esta es una buena práctica para minimizar los costes innecesarios y mantener limpio el entorno. Ejecute lo siguiente:

```
cdk destroy
```

yIngresa para aprobar los cambios y eliminar tu pila.

> **Note**
>
> Si no cambiabas el depósito`RemovalPolicy`, la eliminación de la pila se completaría correctamente, pero el depósito quedaría huérfano (ya no estaría asociado a la pila).

Siguientes pasos

¡Enhorabuena! Ha completado este tutorial y lo ha utilizado AWS CDK para crear, modificar y eliminar correctamente los recursos del Nube de AWS. Ahora está listo para empezar a utilizar el AWS CDK.

Para obtener más información sobre el uso de AWS CDK en su lenguaje de programación preferido, consulteTrabajar con AWS CDK los lenguajes de programación compatibles.

Para obtener recursos adicionales, consulte lo siguiente:

- Pruebe el taller sobre el CDK para realizar un recorrido más detallado sobre un proyecto más complejo.
- Profundice en conceptos como the section called “Entornos”the section called “Activos”,the section called “Permisos”, the section called “Context”the section called “Parámetros”, ythe section called “Personalización de componentes fijos”.
- Consulta la referencia de la API para empezar a explorar las estructuras de CDK disponibles para tus servicios favoritos AWS .
- Visite Construct Hub para descubrir construcciones creadas por AWS y otras personas.

- Explore ejemplos de uso del AWS CDK.

AWS CDK Es un proyecto de código abierto. Para contribuir, consulte Contribuir al AWS Cloud Development Kit (AWS CDK).

Migración de la AWS CDK versión 1 a la versión 2 AWS CDK

La versión 2 de AWS Cloud Development Kit (AWS CDK) está diseñada para facilitar la escritura de la infraestructura como código en su lenguaje de programación preferido. En este tema se describen los cambios entre la versión 1 y la versión 2 del AWS CDK.

> **Tip**
>
> Para identificar las pilas implementadas con la AWS CDK versión 1, utilice la utilidad awscdk-v1-stack-finder.

Los principales cambios de la versión 1 a la versión 2 de CDK son los siguientes. AWS CDK

- AWS CDK La versión 2 consolida las partes estables de la biblioteca AWS Construct, incluida la biblioteca principal, en un solo paquete. `aws-cdk-lib` Los desarrolladores ya no necesitan instalar paquetes adicionales para los AWS servicios individuales que utilizan. Este enfoque de paquete único también significa que no es necesario sincronizar las versiones de los distintos paquetes de la biblioteca de CDK.

  Las construcciones L1 (CFNxxxx), que representan los recursos exactos disponibles, siempre se consideran estables y, por lo tanto AWS CloudFormation, se incluyen en ellas. `aws-cdk-lib`
- Los módulos experimentales, en los que seguimos trabajando con la comunidad para desarrollar nuevas construcciones de nivel 2 o nivel 3, no están incluidos. `aws-cdk-lib` En cambio, se distribuyen como paquetes individuales. Los paquetes experimentales se nombran con un `alpha` sufijo y un número de versión semántico. El número de versión semántica coincide con la primera versión de la biblioteca AWS Construct con la que son compatibles, también con un sufijo. `alpha` Las construcciones pasan a `aws-cdk-lib` ser designadas estables, lo que permite que la biblioteca de construcciones principal siga un estricto control de versiones semántico.

  La estabilidad se especifica a nivel de servicio. Por ejemplo, si empezamos a crear una o más construcciones de L2 para Amazon AppFlow, que al momento de escribir este artículo solo tiene construcciones de L1, aparecen primero en un módulo denominado. `@aws-cdk/aws-appflow-alpha` Luego, pasan a un `aws-cdk-lib` momento en el que consideramos que los nuevos diseños satisfacen las necesidades fundamentales de los clientes.

Una vez que un módulo se ha designado estable y se ha incorporado a élaws-cdk-lib, se añaden nuevas API mediante la convención «BetAN» que se describe en el siguiente bullet.

Se publica una nueva versión de cada módulo experimental con cada versión del AWS CDK. Sin embargo, en su mayor parte, no es necesario mantenerlos sincronizados. Puede actualizar aws-cdk-lib el módulo experimental cuando lo desee. La excepción es que cuando dos o más módulos experimentales relacionados dependen uno del otro, deben ser de la misma versión.

- En el caso de los módulos estables a los que se añade una nueva funcionalidad, las API nuevas (ya se trate de construcciones completamente nuevas o de métodos o propiedades nuevas de una construcción existente) reciben un Beta1 sufijo mientras se está trabajando en ellas. (Seguido de Beta2Beta3, y así sucesivamente cuando sea necesario realizar cambios importantes). Cuando la API se designa estable, se agrega una versión de la API sin el sufijo. Todos los métodos, excepto el más reciente (ya sea beta o final), quedarán obsoletos.

  Por ejemplo, si añadimos un método nuevo grantPower() a una construcción, inicialmente aparecerá comograntPowerBeta1(). Si se necesitan cambios importantes (por ejemplo, un nuevo parámetro o propiedad obligatorio), se asignará un nombre grantPowerBeta2() a la siguiente versión del método y así sucesivamente. Una vez finalizado el trabajo y finalizada la API, se añade el método grantPower() (sin sufijo) y los métodos betAN quedan obsoletos.

  Todas las API beta permanecen en la biblioteca Construct hasta la próxima versión principal (3.0) y sus firmas no cambiarán. Si las utilizas, verás advertencias de obsolescencia, por lo que deberías pasar a la versión final de la API lo antes posible. Sin embargo, ninguna versión AWS CDK 2.x futura interrumpirá su aplicación.
- La Construct clase se ha extraído de AWS CDK una biblioteca independiente, junto con los tipos relacionados. Esto se hace para apoyar los esfuerzos por aplicar el modelo de programación de Construct a otros dominios. Si está escribiendo sus propias construcciones o utiliza API relacionadas, debe declarar el constructs módulo como una dependencia y realizar cambios menores en las importaciones. Si utilizas funciones avanzadas, como conectarte al ciclo de vida de la aplicación CDK, es posible que necesites realizar más cambios. Para obtener más información, consulta la RFC.
- Las propiedades, métodos y tipos obsoletos de la AWS CDK versión 1.x y su biblioteca Construct se han eliminado por completo de la API de CDK v2. En la mayoría de los lenguajes compatibles, estas API generan advertencias en la versión 1.x, por lo que es posible que ya hayas migrado a las API de reemplazo. Encontrará una lista completa de las API obsoletas de la versión 1.x de CDK en. GitHub

- El comportamiento que estaba restringido por indicadores de características en la versión AWS CDK 1.x está activado de forma predeterminada en la versión CDK v2. Los indicadores de características anteriores ya no son necesarios y, en la mayoría de los casos, no son compatibles. Algunas todavía están disponibles para que puedas volver al comportamiento de la versión CDK v1 en circunstancias muy específicas. Para obtener más información, consulte the section called “Actualizar los indicadores de funciones”.
- Con CDK v2, los entornos en los que se despliega deben arrancarse con la moderna pila de bootstrap. La pila de bootstrap antigua (la predeterminada en la versión 1) ya no es compatible. Además, CDK v2 requiere una nueva versión de la pila moderna. Para actualizar sus entornos existentes, vuelva a iniciarlos. Ya no es necesario establecer ningún indicador de función o variable de entorno para utilizar la pila de bootstrap moderna.

> **Important**
>
> La plantilla bootstrap moderna concede de forma efectiva los permisos implícitos `--cloudformation-execution-policies` a cualquier AWS cuenta de la `--trust` lista. De forma predeterminada, esto extiende los permisos de lectura y escritura a cualquier recurso de la cuenta de arranque. Asegúrese de configurar la pila de arranque con políticas y cuentas de confianza con las que se sienta cómodo.

Nuevos requisitos previos

La mayoría de los requisitos de la AWS CDK versión 2 son los mismos que los de la AWS CDK versión 1.x. Los requisitos adicionales se enumeran aquí.

- Para TypeScript los desarrolladores, se requiere la versión TypeScript 3.8 o una versión posterior.
- Se necesita una nueva versión del kit de herramientas del CDK para su uso con el CDK v2. Ahora que el CDK v2 está disponible de forma general, la v2 es la versión predeterminada al instalar el kit de herramientas del CDK. Es compatible con versiones anteriores de los proyectos del CDK v1, por lo que no es necesario mantener instalada la versión anterior a menos que desee crear proyectos del CDK v1. Para actualizar, emita. `npm install -g aws-cdk`

Actualización desde la AWS CDK versión 2 Developer Preview

Si utilizas la versión preliminar para desarrolladores de CDK v2, tu proyecto depende de una versión Release Candidate de AWS CDK, por ejemplo. `2.0.0-rc1` Actualízalas a los módulos instalados en tu proyecto y`2.0.0`, a continuación, actualiza los módulos instalados en tu proyecto.

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
```

Tras actualizar las dependencias, actualice el kit de herramientas del CDK a la versión de lanzamiento. `npm update -g aws-cdk`

Migración de la versión 1 a la versión 2 del CDK AWS CDK

Para migrar su aplicación a la AWS CDK versión 2, primero actualice las marcas de funciones. `cdk.json` A continuación, actualiza las dependencias e importaciones de la aplicación según sea necesario para el lenguaje de programación en el que está escrita.

¿Se está actualizando a una versión reciente

Estamos viendo que varios clientes se están actualizando de una versión antigua de la versión AWS CDK 1 a la versión más reciente de la versión 2 en un solo paso. Si bien es cierto que es posible hacerlo, estarías actualizando después de varios años de cambios (aunque, lamentablemente, no todos hayan tenido la misma cantidad de pruebas de evolución que tenemos en la actualidad), además de actualizar versiones con nuevos valores predeterminados y una organización de código diferente.

Para disfrutar de una experiencia de actualización más segura y diagnosticar más fácilmente las fuentes de cualquier cambio inesperado, le recomendamos que separe estos dos pasos: primero actualice a la última versión v1 y, a continuación, realice el cambio a la v2.

Actualizar los indicadores de funciones

Elimine los siguientes indicadores de características de la versión 1, `cdk.json` si existen, ya que están todos activos de forma predeterminada en la AWS CDK versión 2. Si su efecto anterior es importante para su infraestructura, tendrá que realizar cambios en el código fuente. Consulte la lista de banderas GitHub para obtener más información.

- `@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`

Se pueden configurar algunos indicadores de características de la versión 1 para `false` volver a comportamientos específicos de la versión AWS CDK 1; consulte the section called “Volviendo al comportamiento de la versión 1” o consulte la lista GitHub para obtener una referencia completa.

Para ambos tipos de indicadores, utilice el `cdk diff` comando para inspeccionar los cambios en la plantilla sintetizada y comprobar si los cambios en alguno de estos indicadores afectan a su infraestructura.

Compatibilidad con el kit de herramientas CDK

El CDK v2 requiere la versión 2 o posterior del CDK Toolkit. Esta versión es compatible con versiones anteriores de las aplicaciones del CDK v1. Por lo tanto, puede usar una única versión del CDK Toolkit instalada globalmente en todos sus AWS CDK proyectos, tanto si utilizan la versión 1 como la versión 2. Una excepción es que CDK Toolkit v2 solo crea proyectos de CDK v2.

Si necesita crear proyectos de CDK tanto de la versión 1 como de la versión 2, no instale el CDK Toolkit v2 de forma global. (Quítelo si ya lo tiene instalado:.) `npm remove -g aws-cdk` Para invocar el kit de herramientas del CDK, utilícelo npx para ejecutar la versión 1 o la versión 2 del kit de herramientas del CDK según lo desee.

```
npx aws-cdk@1.x init app --language typescript
npx aws-cdk@2.x init app --language typescript
```

> **Tip**
>
> Configure los alias de la línea de comandos para poder usar los cdk1 comandos cdk y para invocar la versión deseada del kit de herramientas del CDK.
>
> 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 $*
> ```

Actualización de las dependencias y las importaciones

Actualiza las dependencias de tu aplicación y, a continuación, instala los nuevos paquetes. Por último, actualiza las importaciones en tu código.

TypeScript

Aplicaciones

En el caso de las aplicaciones CDK, actualízalas de `package.json` la siguiente manera. Elimine las dependencias de los módulos estables individuales tipo v1 y establezca la versión más baja `aws-cdk-lib` que necesite para su aplicación (aquí la versión 2.0.0).

Las construcciones experimentales se proporcionan en paquetes separados y con versiones independientes con nombres que terminan en `alpha` y un número de versión alfabético. El número de versión alfa corresponde a la primera versión `aws-cdk-lib` con la que son compatibles. En este caso, hemos fijado la versión `aws-codestar` 2.0.0-alpha.1.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
    "constructs": "^10.0.0"
  }
}
```

Construye bibliotecas

Para crear bibliotecas, establezca la versión más baja `aws-cdk-lib` que necesite para su aplicación (aquí la 2.0.0) y actualícela de la `package.json` siguiente manera.

Tenga en cuenta que `aws-cdk-lib` aparece tanto como una dependencia entre pares como una dependencia de desarrollo.

```
{
  "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**
>
> Cuando publiques una biblioteca compatible con la versión 2, deberías introducir un cambio importante en el número de versión de tu biblioteca, ya que se trata de un cambio

radical para los usuarios de la biblioteca. No es posible admitir tanto la versión 1 como la versión 2 de CDK con una sola biblioteca. Para seguir dando soporte a los clientes que aún utilizan la versión 1, puede mantener la versión anterior en paralelo o crear un paquete nuevo para la versión 2.
Tú decides cuánto tiempo quieres seguir ofreciendo soporte a los clientes de la AWS CDK versión 1. Puede seguir el ejemplo del ciclo de vida del propio CDK v1, que entró en mantenimiento el 1 de junio de 2022 y estará operativo el 1 de end-of-life junio de 2023. Para obtener más información, consulte la Política de AWS CDK mantenimiento.

Tanto bibliotecas como aplicaciones

Instale las nuevas dependencias ejecutando `npm install` o `yarn install`.

Cambie sus importaciones para importar `Construct` desde el nuevo `constructs` módulo, los tipos principales, como `App` y `Stack` desde el nivel superior `aws-cdk-lib`, y los módulos estables de Construct Library para los servicios que utilice desde los espacios de nombres de abajo. `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

Actualice de la siguiente `package.json` manera. Elimine las dependencias de los módulos estables individuales de estilo v1 y establezca la versión más baja `aws-cdk-lib` que necesite para su aplicación (aquí la versión 2.0.0).

Las construcciones experimentales se proporcionan en paquetes separados y con versiones independientes con nombres que terminan en `alpha` y un número de versión alfabético.
El número de versión alfa corresponde a la primera versión `aws-cdk-lib` con la que son compatibles. En este caso, hemos fijado la versión `aws-codestar` 2.0.0-alpha.1.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
```

```
    "constructs": "^10.0.0"
  }
}
```

Instale las nuevas dependencias ejecutando o. `npm install` `yarn install`

Cambia las importaciones de tu aplicación para hacer lo siguiente:

- Importa `Construct` desde el nuevo `constructs` módulo
- Importe los tipos principales, como `App` y`Stack`, desde el nivel superior de `aws-cdk-lib`
- Importe los módulos de AWS Construct Library desde los espacios de nombres de `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

Actualice `requirements.txt` la `install_requires` definición de la siguiente `setup.py` manera. Elimine las dependencias de los módulos estables individuales de estilo v1.

Las construcciones experimentales se proporcionan en paquetes separados y con versiones independientes con nombres que terminan en `alpha` y un número de versión alfabético. El número de versión alfa corresponde a la primera versión `aws-cdk-lib` con la que son compatibles. En este caso, hemos fijado la versión `aws-codestar` 2.0.0alpha1.

```
install_requires=[
     "aws-cdk-lib>=2.0.0",
     "constructs>=10.0.0",
     "aws-cdk.aws-codestar-alpha>=2.0.0alpha1",
     # ...
],
```

> **Tip**
>
> Desinstale cualquier otra versión de los AWS CDK módulos que ya estén instalados en el entorno virtual de su aplicación utilizando. `pip uninstall` A continuación, instale las nuevas dependencias con`python -m pip install -r requirements.txt`.

Cambia las importaciones de tu aplicación para hacer lo siguiente:

- Importa `Construct` desde el nuevo `constructs` módulo
- Importe los tipos principales, como `App` y`Stack`, desde el nivel superior de `aws_cdk`
- Importe los módulos de AWS Construct Library desde los espacios de nombres de `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

En`pom.xml`, elimine todas `software.amazon.awscdk` las dependencias de los módulos estables y sustitúyalas por las dependencias de `software.constructs` (para) y. `Construct` `software.amazon.awscdk`

Las construcciones experimentales se proporcionan en paquetes separados y versionados de forma independiente con nombres que terminan en `alpha` y un número de versión alfabético. El número de versión alfa corresponde a la primera versión `aws-cdk-lib` con la que son compatibles. En este caso, hemos fijado la versión `aws-codestar` 2.0.0-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>
```

Instala las nuevas dependencias ejecutándolas. `mvn package`

Cambie el código para hacer lo siguiente:

- Importa `Construct` desde la nueva `software.constructs` biblioteca
- Importa las clases principales, como `Stack` y `App` desde `software.amazon.awscdk`
- Importe construcciones de servicios desde `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#

La forma más sencilla de actualizar las dependencias de una aplicación CDK de C# es editar el archivo manualmente. `.csproj` Elimine todas las referencias de `Amazon.CDK.*` paquetes estables y sustitúyalas por referencias a los `Amazon.CDK.Lib` paquetes y. `Constructs`

Las construcciones experimentales se proporcionan en paquetes separados y con versiones independientes con nombres que terminan en `alpha` y un número de versión alfabético. El número de versión alfa corresponde a la primera versión `aws-cdk-lib` con la que son compatibles. En este caso, hemos fijado la versión `aws-codestar` 2.0.0-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" />
```

Instala las nuevas dependencias ejecutándolas. `dotnet restore`

Cambie las importaciones en sus archivos fuente de la siguiente manera.

```
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

Ejecute este error go get para actualizar sus dependencias a la última versión y actualizar el `.mod` archivo de su proyecto.

Probar la aplicación migrada antes de implementarla

Antes de implementar tus pilas, úsalas `cdk diff` para comprobar si hay cambios inesperados en los recursos. No se esperan cambios en los ID lógicos (que provoquen la sustitución de recursos).

Los cambios esperados incluyen, pero no se limitan a:

- Cambios en el `CDKMetadata` recurso.
- Se han actualizado los hashes de los activos.
- Cambios relacionados con el nuevo estilo de síntesis de pilas. Se aplica si tu aplicación utilizaba el sintetizador de pilas antiguo de la versión 1. (El CDK v2 no es compatible con el sintetizador de pilas anterior).
- La adición de una regla. `CheckBootstrapVersion`

Por lo general, los cambios inesperados no se deben a la actualización a la AWS CDK versión 2 en sí misma. Por lo general, son el resultado de un comportamiento obsoleto que anteriormente había sido modificado por las marcas de características. Esto es un síntoma de una actualización desde una versión de CDK anterior a la 1.85.x, aproximadamente. Verá los mismos cambios al actualizar a la versión más reciente de la versión 1.x. Por lo general, esto se puede resolver de la siguiente manera:

1. Actualiza tu aplicación a la versión más reciente, la versión 1.x
2. Elimina los indicadores de funciones
3. Revisa tu código según sea necesario
4. Implementación
5. Actualice a la versión 2

> **Note**
>
> Si la aplicación actualizada no se puede implementar después de la actualización en dos etapas, informa del problema.

Cuando esté listo para implementar las pilas en su aplicación, considere implementar primero una copia para poder probarla. La forma más sencilla de hacerlo es implementarla en una región diferente. Sin embargo, también puedes cambiar los ID de tus pilas. Tras la prueba, asegúrate de destruir la copia de prueba con cdk destroy ella.

Solución de problemas

TypeScript **`'from' expected`**o **`';' expected`** error en las importaciones

Actualice a TypeScript 3.8 o una versión posterior.

Ejecute 'cdk bootstrap'

Si ve un error como el siguiente:

```
#  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 La versión 2 requiere una pila de arranque actualizada y, además, todas las implementaciones de la versión 2 requieren recursos de arranque. (Con la versión 1, puede

implementar pilas sencillas sin necesidad de arrancar). Para ver todos los detalles, consulte the section called “Bootstrapping (Proceso de arranque)”.

¿Cómo encontrar pilas de la versión 1

Al migrar su aplicación de CDK de la versión 1 a la versión 2, es posible que desee identificar las AWS CloudFormation pilas implementadas que se crearon con la versión 1. Para ello, ejecute el siguiente comando:

```
npx awscdk-v1-stack-finder
```

Para obtener información sobre el uso, consulte el archivo README awscdk-v1-stack-finder.

Migre los recursos y AWS CloudFormation plantillas existentes a AWS CDK

La función CDK Migrate se encuentra en versión preliminar AWS CDK y está sujeta a cambios.

Utilice la interfaz de línea de AWS Cloud Development Kit (AWS CDK) comandos (AWS CDK CLI) para migrar AWS los recursos implementados, las AWS CloudFormation pilas implementadas y las AWS CloudFormation plantillas locales a. AWS CDK

Temas

Cómo funciona la migración

Utilice el AWS CDK CLI `cdk migrate` comando para migrar desde las siguientes fuentes:

- AWS Recursos desplegados.
- AWS CloudFormation Pilas desplegadas.
- AWS CloudFormation Plantillas locales.

AWS Recursos desplegados

Puede migrar AWS los recursos desplegados desde un entorno específico (Cuenta de AWS y Región de AWS) que no estén asociados a una AWS CloudFormation pila.

AWS CDK CLIUtiliza el servicio generador de iAC para buscar recursos en su AWS entorno y recopilar detalles sobre los recursos. Para obtener más información sobre el generador IaC, consulte Generación de plantillas para los recursos existentes en la Guía del AWS CloudFormation usuario.

Tras recopilar los detalles de los recursos, AWS CDK CLI crea una nueva aplicación de CDK que incluye una única pila que contiene los recursos migrados.

Pilas implementadas AWS CloudFormation

Puede migrar una sola AWS CloudFormation pila a una nueva AWS CDK aplicación. AWS CDK CLIRecuperará la AWS CloudFormation plantilla de tu pila y creará una nueva aplicación de CDK. La aplicación CDK constará de una sola pila que contendrá la pila migrada AWS CloudFormation .

Plantillas locales AWS CloudFormation

Puede migrar desde una AWS CloudFormation plantilla local. Las plantillas locales pueden contener recursos implementados o no. AWS CDK CLICreará una nueva aplicación de CDK que contendrá una sola pila con tus recursos.

Tras la migración, puede administrar, modificar e implementar su aplicación de CDK para AWS CloudFormation aprovisionar o actualizar sus recursos.

Ventajas de CDK Migrate

Históricamente, la migración de recursos a un proceso manual AWS CDK ha requerido tiempo y experiencia, e incluso AWS CDK empezar. AWS CloudFormation Con CDK Migrate, le AWS CDK CLI facilita la mayor parte del esfuerzo de migración en una fracción del tiempo. CDK Migrate le permitirá empezar rápidamente a utilizarla AWS CDK para desarrollar y gestionar aplicaciones nuevas y existentes. AWS

Consideraciones

Consideraciones generales

CDK Migrate frente a CDK Import

El `cdk import` comando puede importar los recursos implementados a una aplicación de CDK nueva o existente. Al importar, cada recurso tendrá que definirse manualmente como una

construcción de nivel 1 en la aplicación. Te recomendamos que lo `cdk import` utilices para importar uno o más recursos a la vez a una aplicación CDK nueva o existente. Para obtener más información, consulte Importación de recursos existentes en una pila.

El `cdk migrate` comando migra desde los recursos desplegados, las AWS CloudFormation pilas desplegadas o las AWS CloudFormation plantillas locales a una nueva aplicación de CDK. Durante la migración, se AWS CDK CLI utiliza `cdk import` para importar tus recursos a la nueva aplicación de CDK. AWS CDK CLITambién genera estructuras de nivel 1 para cada recurso. Te recomendamos que lo `cdk migrate` utilices al importar desde una fuente de migración compatible a una nueva AWS CDK aplicación.

CDK Migrate crea únicamente construcciones de capa 1

La aplicación CDK recién creada incluirá únicamente las construcciones de L1. Puede agregar construcciones de nivel superior a su aplicación después de la migración.

CDK Migrate crea aplicaciones de CDK que contienen una sola pila

La aplicación CDK recién creada contendrá una sola pila.

Al migrar los recursos implementados, todos los recursos migrados se incluirán en una sola pila en la nueva aplicación de CDK.

Al migrar AWS CloudFormation pilas, solo puede migrar una pila a una única AWS CloudFormation pila en la nueva aplicación CDK.

Migración de activos

Los activos del proyecto, como el AWS Lambda código, no se migrarán directamente a la nueva aplicación CDK. Tras la migración, puede especificar los valores de los activos para incluirlos en la aplicación CDK.

Migración de recursos con estado

Al migrar recursos con estado, como bases de datos y buckets de Amazon Simple Storage Service (Amazon S3), lo más habitual es migrar el recurso existente en lugar de crear uno nuevo.

Para migrar y conservar los recursos con estado, haga lo siguiente:

- Compruebe que el recurso con estado admite la importación. Para obtener más información, consulte la compatibilidad con los tipos de recursos en la Guía del AWS CloudFormation usuario.

- Tras la migración, compruebe que el ID lógico del recurso migrado en la nueva aplicación CDK coincide con el ID lógico del recurso implementado.
- Si está migrando desde una AWS CloudFormation pila, compruebe que el nombre de la pila de la nueva aplicación de CDK coincide con la pila. AWS CloudFormation
- Implemente la aplicación CDK con la misma AWS cuenta y el recurso Región de AWS migrado.

Consideraciones a la hora de migrar desde una plantilla AWS CloudFormation

CDK Migrate admite la migración de una sola plantilla

Al migrar AWS CloudFormation plantillas, puede seleccionar una sola plantilla para migrar. No se admiten las plantillas anidadas.

Migración de plantillas con funciones intrínsecas

Al migrar desde una AWS CloudFormation plantilla que usa funciones intrínsecas, AWS CDK CLI intentará migrar tu lógica a la aplicación CDK con la clase. `Fn` Para obtener más información, consulta la clase Fn en la referencia de la AWS Cloud Development Kit (AWS CDK) API.

Consideraciones a la hora de migrar desde recursos implementados

Limitaciones de escaneo

Al escanear su entorno en busca de recursos, el generador IaC tiene limitaciones específicas en cuanto a los datos que puede recuperar y limitaciones de cuota al escanear. Para obtener más información, consulte las consideraciones en la Guía del AWS CloudFormation usuario.

Requisitos previos

Antes de usar el `cdk migrate` comando, haga lo siguiente:

1. Establezca la autenticación con AWS. Para ver instrucciones, consulte Paso 2: Configurar el acceso mediante programación.
2. Instalar o actualizar la AWS CDK CLI Para obtener las instrucciones de instalación, consulte Paso 3: Instala el AWS CDKCLI.

Comience a utilizar CDK Migrate

Para empezar, ejecute el AWS CDK CLI `cdk migrate` comando desde el directorio que elija. Proporcione las opciones obligatorias y opcionales, según el tipo de migración que vaya a realizar.

Para obtener una lista completa y una descripción de las opciones que puede utilizar`cdk migrate`, consulte[Referencia de comandos de cdk migrate](#).

A continuación se muestran algunas opciones importantes que puede que desee proporcionar.

Nombre de pila

La única opción obligatoria es`--stack-name`. Use esta opción para especificar un nombre para la pila que se creará en la AWS CDK aplicación después de la migración. El nombre de la pila también se utilizará como nombre de la AWS CloudFormation pila en el momento de la implementación.

Idioma

Se usa `--language` para especificar el lenguaje de programación de la nueva aplicación CDK.

AWS cuenta y Región de AWS

AWS CDK CLIRecupera la AWS cuenta y la Región de AWS información de las fuentes predeterminadas. Para obtener más información, consulte [Paso 2: Configurar el acceso mediante programación](#). Puede utilizar `--account` cualquier `--region` opción `cdk migrate` para proporcionar otros valores.

Directorio de salida de su nuevo proyecto de CDK

De forma predeterminada, AWS CDK CLI creará un nuevo proyecto de CDK en su directorio de trabajo y utilizará el valor que proporcione para asignar un nombre `--stack-name` a la carpeta del proyecto. Si actualmente existe una carpeta con el mismo nombre, la AWS CDK CLI sobrescribirá.

Puede especificar una ruta de salida diferente para la nueva carpeta del proyecto CDK con la `--output-path` opción.

Fuente de migración

Proporcione una opción para especificar la fuente desde la que está migrando.

- `--from-path`— Migre desde una AWS CloudFormation plantilla local.

- `--from-scan`— Migre desde los recursos desplegados en una AWS cuenta y Región de AWS.
- `--from-stack`— Migre desde una AWS CloudFormation pila.

En función de la fuente de migración, puede proporcionar opciones adicionales para personalizar el `cdk migrate` comando.

Migre desde una AWS CloudFormation pila

Para migrar desde una AWS CloudFormation pila implementada, ofrezca la `--from-stack` opción. Proporcione el nombre de la AWS CloudFormation pila implementada con`--stack-name`. A continuación, se muestra un ejemplo:

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

AWS CDK CLIHará lo siguiente:

1. Recupera la AWS CloudFormation plantilla de tu pila desplegada.
2. Ejecute `cdk init` para inicializar una nueva aplicación de CDK.
3. Cree una pila dentro de la aplicación CDK que contenga la pila migrada. AWS CloudFormation

Al migrar desde una AWS CloudFormation pila implementada, AWS CDK CLI intenta hacer coincidir los ID lógicos de los recursos implementados y el nombre de la AWS CloudFormation pila implementada con los recursos migrados y la pila de la nueva aplicación de CDK.

Tras la migración, puede administrar y modificar su aplicación de CDK con normalidad. Al realizar la implementación, AWS CloudFormation identificará la implementación como una actualización de la AWS CloudFormation pila debido al nombre de la AWS CloudFormation pila coincidente. Se actualizarán los recursos con identificadores lógicos coincidentes. Para obtener más información sobre la implementación, consulteAdministre e implemente su aplicación CDK.

Migre desde una AWS CloudFormation plantilla

CDK Migrate admite la migración desde AWS CloudFormation plantillas formateadas en o. `JSON` `YAML`

Para migrar desde una AWS CloudFormation plantilla local, utilice la `--from-path` opción y proporcione una ruta a la plantilla local. También debe proporcionar la `--stack-name` opción requerida. A continuación, se muestra un ejemplo:

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

AWS CDK CLIHará lo siguiente:

1. Recupera tu AWS CloudFormation plantilla local.
2. Ejecute `cdk init` para inicializar una nueva aplicación de CDK.
3. Cree una pila dentro de la aplicación CDK que contenga la plantilla migrada. AWS CloudFormation

Tras la migración, puede administrar y modificar su aplicación de CDK con normalidad. En el momento de la implementación, tiene las siguientes opciones:

- Actualizar una AWS CloudFormation pila: si la AWS CloudFormation plantilla local se implementó anteriormente, puede actualizar la AWS CloudFormation pila implementada.
- Implementar una AWS CloudFormation pila nueva: si la AWS CloudFormation plantilla local nunca se implementó o si desea crear una nueva pila a partir de una plantilla implementada anteriormente, puede implementar una nueva AWS CloudFormation pila.

Migre desde una AWS SAM plantilla

Para migrar desde una plantilla AWS Serverless Application Model (AWS SAM), primero debes convertirla en una AWS CloudFormation plantilla o implementarla para crear una AWS CloudFormation pila.

Para convertir una AWS SAM plantilla en AWS CloudFormation, puedes usar el AWS SAM CLI `sam validate --debug` comando. Puede que tenga que `lint` configurarlo `false` en su `samconfig.toml` archivo antes de ejecutar este comando.

Para convertirla en una AWS CloudFormation pila, despliegue la AWS SAM plantilla mediante AWS SAM CLI. A continuación, migre desde la pila implementada.

Migre desde los recursos desplegados

Para migrar desde AWS los recursos desplegados, ofrezca la `--from-scan` opción. También debe proporcionar la `--stack-name` opción requerida. A continuación, se muestra un ejemplo:

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

AWS CDK CLIHará lo siguiente:

1. Escanee su cuenta en busca de detalles de recursos y propiedades: AWS CDK CLI utiliza el generador iAC para escanear su cuenta y recopilar detalles.
2. Genere una AWS CloudFormation plantilla: después de escanear, AWS CDK CLI utiliza el generador iAC para crear una plantilla. AWS CloudFormation
3. Inicialice una nueva aplicación de CDK y migre su plantilla: AWS CDK CLI se ejecuta `cdk init` para inicializar una nueva AWS CDK aplicación y migra la AWS CloudFormation plantilla a la aplicación de CDK como una sola pila.

Usa filtros

De forma predeterminada, AWS CDK CLI escaneará todo el AWS entorno y migrará los recursos hasta el límite máximo de cuota del generador IaC. Puede proporcionar filtros AWS CDK CLI para especificar un criterio para migrar los recursos de su cuenta a la nueva aplicación CDK. Para obtener más información, consulte `--filter`.

Escaneo de recursos con el generador iAC

Según la cantidad de recursos de su cuenta, el escaneo puede tardar unos minutos. Aparecerá una barra de progreso durante el proceso de digitalización.

Tipos de recursos admitidos

> AWS CDK CLIMigrará los recursos compatibles con el generador IaC. Para obtener una lista completa, consulte el soporte de tipos de recursos en la Guía del AWS CloudFormation usuario.

Resuelva las propiedades de solo escritura

Algunos recursos compatibles contienen propiedades de solo escritura. Se puede escribir en estas propiedades para configurar la propiedad, pero el generador IaC no las puede leer ni AWS

CloudFormation para obtener el valor. Por ejemplo, una propiedad utilizada para especificar la contraseña de una base de datos puede ser de solo escritura por motivos de seguridad.

Al analizar los recursos durante la migración, el generador IaC detectará los recursos que puedan contener propiedades de solo escritura y los clasificará en cualquiera de los siguientes tipos:

- `MUTUALLY_EXCLUSIVE_PROPERTIES`— Se trata de propiedades de solo escritura para un recurso específico que son intercambiables y tienen un propósito similar. Se requiere una de las propiedades que se excluyen mutuamente para configurar el recurso. Por ejemplo, las propiedades `S3BucketImageUri`, y de un `AWS::Lambda::Function` recurso son `ZipFile` propiedades de solo escritura que se excluyen mutuamente. Puede usar cualquiera de ellas para especificar los activos de su función, pero debe usar una.
- `MUTUALLY_EXCLUSIVE_TYPES`— Se trata de propiedades obligatorias de solo escritura que aceptan varios tipos de configuración. Por ejemplo, la `Body` propiedad de un `AWS::ApiGateway::RestApi` recurso acepta un tipo de objeto o cadena.
- `UNSUPPORTED_PROPERTIES`— Se trata de propiedades de solo escritura que no se incluyen en las otras dos categorías. Son propiedades opcionales o propiedades obligatorias que aceptan una matriz de objetos.

Para obtener más información sobre las propiedades de solo escritura y cómo las gestiona el generador de IaC al buscar recursos desplegados y crear AWS CloudFormation plantillas, consulte el generador de IaC y las propiedades de solo escritura en la Guía del usuario.AWS CloudFormation

Tras la migración, debe especificar los valores de las propiedades de solo escritura en la nueva aplicación CDK. AWS CDK CLIAñadirá una sección de advertencias al `ReadMe` archivo del proyecto CDK para documentar cualquier propiedad de solo escritura que haya sido identificada por el generador IaC. A continuación, se muestra un ejemplo:

```
# 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
```

- Las advertencias se organizan bajo encabezados que identifican el identificador lógico del recurso al que están asociadas.
- Las advertencias se clasifican por tipo. Estos tipos provienen directamente del generador de iAc.

Para resolver las propiedades de solo escritura

1. Identifique las propiedades de solo escritura que desee resolver en la sección Advertencias del archivo de su proyecto de CDK. `ReadMe` Aquí puede tomar nota de los recursos de su aplicación de CDK que pueden contener propiedades de solo escritura e identificar los tipos de propiedades de solo escritura que se descubrieron.

   a. Por ejemplo`MUTUALLY_EXCLUSIVE_PROPERTIES`, determine qué propiedad que se excluyen mutuamente configurar en su aplicación. AWS CDK

   b. Para `MUTUALLY_EXCLUSIVE_TYPES` ello, determine qué tipo aceptado utilizará para configurar la propiedad.

   c. Para`UNSUPPORTED_PROPERTIES`, determine si la propiedad es opcional o obligatoria. A continuación, configúrela según sea necesario.

2. Utilice las instrucciones del generador de IaC y las propiedades de solo escritura para hacer referencia al significado de los tipos de advertencia.
3. En tu aplicación CDK, los valores de las propiedades de solo escritura que se deben resolver también se especificarán en la sección de tu aplicación. `Props` Introduce aquí los valores

correctos. Para obtener instrucciones y descripciones de las propiedades, puedes consultar la referencia de la AWS CDK API.

A continuación, se muestra un ejemplo de la `Props` sección de una aplicación de CDK migrada con dos propiedades de solo escritura que resolver:

```
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 vez que haya resuelto todos los valores de las propiedades de solo escritura, estará listo para prepararse para la implementación.

El archivo migrate.json

AWS CDK CLICrea un `migrate.json` archivo en el AWS CDK proyecto durante la migración. Este archivo contiene información de referencia sobre los recursos implementados. Al implementar la aplicación CDK por primera vez, AWS CDK CLI utiliza este archivo para hacer referencia a los recursos implementados, los asocia a la nueva AWS CloudFormation pila y elimina el archivo.

Administre e implemente su aplicación CDK

Al migrar a AWS CDK, es posible que la nueva aplicación de CDK no esté lista para su implementación inmediata. En este tema se describen las medidas que se deben tener en cuenta al administrar e implementar su nueva aplicación de CDK.

Preparación para la implementación

Antes de implementarla, debe preparar la aplicación CDK.

Sintetice su aplicación

Usa el `cdk synth` comando para sintetizar la pila de tu aplicación de CDK en una plantilla. AWS CloudFormation

Si migraste desde una AWS CloudFormation pila o plantilla implementada, puedes comparar la plantilla sintetizada con la plantilla migrada para verificar los valores de los recursos y las propiedades.

Para obtener más información acerca de la `cdk synth`, consulte Sintetizando pilas.

Realice una diferencia

Si migraste desde una AWS CloudFormation pila implementada, puedes usar el comando cdk diff para compararla con la pila de tu nueva aplicación de CDK.

Para obtener más información sobre cdk diff, consulte. Comparación de pilas

Inicie su entorno

Si va a realizar una implementación desde un AWS entorno por primera vez, úsela `cdk bootstrap` para preparar su entorno. Para obtener más información, consulte Bootstrapping (Proceso de arranque).

Implemente su aplicación CDK

Cuando implementa una aplicación de CDK, AWS CDK CLI utiliza el AWS CloudFormation servicio para aprovisionar sus recursos. Los recursos se agrupan en una sola pila en la aplicación CDK y se implementan como una sola pila. AWS CloudFormation

Según el lugar desde el que haya migrado, puede realizar la implementación para crear una AWS CloudFormation pila nueva o actualizar una pila existente AWS CloudFormation .

Implemente para crear una AWS CloudFormation pila nueva

Si migró desde los recursos desplegados, AWS CDK CLI se creará automáticamente una nueva AWS CloudFormation pila en el momento de la implementación. Los recursos desplegados se incluirán en la nueva AWS CloudFormation pila.

Si migraste desde una AWS CloudFormation plantilla local que nunca se implementó, se AWS CDK CLI creará automáticamente una nueva AWS CloudFormation pila en el momento de la implementación.

Si migró desde una AWS CloudFormation pila implementada o una AWS CloudFormation plantilla local que se implementó anteriormente, puede realizar la implementación para crear una AWS CloudFormation pila nueva. Para crear una pila nueva, haga lo siguiente:

- Implemente en un AWS entorno nuevo. Consiste en utilizar una AWS cuenta diferente o realizar la implementación en una diferente Región de AWS.
- Si quieres implementar una pila nueva en el mismo AWS entorno que la pila o plantilla migrada, debes modificar el nombre de la pila en tu aplicación de CDK por un nuevo valor. También debe modificar todos los ID lógicos de los recursos de su aplicación de CDK. A continuación, puede realizar la implementación en el mismo entorno para crear una pila nueva y nuevos recursos.

Implemente para actualizar una AWS CloudFormation pila existente

Si migró desde una AWS CloudFormation pila implementada o una AWS CloudFormation plantilla local que se implementó anteriormente, puede realizar la implementación para actualizar la AWS CloudFormation pila existente.

Compruebe que el nombre de la pila de su aplicación de CDK coincida con el nombre de la AWS CloudFormation pila implementada e impleméntela en el mismo AWS entorno.

Trabajar con AWS CDK los lenguajes de programación compatibles

Úselo AWS Cloud Development Kit (AWS CDK) para definir su Nube de AWS infraestructura con un lenguaje de programación compatible.

Temas

Importación de la biblioteca AWS Construct

AWS CDK Incluye la biblioteca AWS Construct, una colección de componentes fijos organizados por AWS servicio. Las construcciones estables de la biblioteca se ofrecen en un solo módulo, denominado por su nombre de TypeScript paquete:. `aws-cdk-lib` El nombre real del paquete varía según el idioma.

TypeScript

Instalación	`npm install aws-cdk-lib`
Importación	`const cdk = require (''); aws-cdk-lib`

JavaScript

Instalación	`npm install aws-cdk-lib`
Importación	`const cdk = require (''); aws-cdk-lib`

Python

Instalación	`python -m pip install aws-cdk-lib`
Importación	`importar aws_cdk como cdk`

Java

Añadir a `pom.xml`	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Importación	`importar Software.Amazon.AWSCDK.app;` (for example)

C#

Instalación	`dotnet agrega paquete Amazon.CDK.lib`
Importación	`utilizando Amazon.cdk;`

La clase `construct` base y el código de soporte se encuentran en el módulo. `constructs` Las construcciones experimentales, en las que la API aún se está perfeccionando, se distribuyen como módulos separados.

La referencia de la API AWS CDK

La referencia de la AWS CDK API proporciona documentación detallada de las construcciones (y otros componentes) de la biblioteca. Se proporciona una versión de la referencia de la API para cada lenguaje de programación compatible.

El material de referencia de cada módulo se divide en las siguientes secciones.

- Descripción general: material introductorio que necesitará conocer para trabajar con el servicio que se ofrece en él AWS CDK, que incluye conceptos y ejemplos.
- Construcciones: clases de biblioteca que representan uno o más AWS recursos concretos. Estos son los recursos o patrones «seleccionados» (L2) (recursos L3) que proporcionan una interfaz de alto nivel con valores predeterminados adecuados.
- Clases: clases que no son de construcción y que proporcionan la funcionalidad utilizada por las construcciones del módulo.
- Estructuras: estructuras de datos (paquetes de atributos) que definen la estructura de los valores compuestos, como las propiedades (el `props` argumento de las construcciones) y las opciones.
- Interfaces: las interfaces, cuyos nombres comienzan todos por «I», definen la funcionalidad mínima absoluta para la construcción correspondiente u otra clase. El CDK usa interfaces de construcción para representar AWS los recursos que están definidos fuera de la AWS CDK aplicación y a los que se hace referencia mediante métodos como`Bucket.fromBucketArn()`.
- Enums: colecciones de valores con nombre que se utilizan para especificar determinados parámetros de construcción. El uso de un valor enumerado permite a la CDK comprobar la validez de estos valores durante la síntesis.
- CloudFormation Recursos: Estas construcciones de L1, cuyos nombres comienzan por «Cfn», representan exactamente los recursos definidos en la especificación. CloudFormation Se generan automáticamente a partir de esa especificación con cada versión de CDK. Cada construcción L2 o L3 encapsula uno o más recursos. CloudFormation
- CloudFormation Tipos de propiedades: conjunto de valores con nombre que definen las propiedades de cada recurso. CloudFormation

Comparación de las interfaces con las clases de construcción

AWS CDK Utiliza las interfaces de una manera específica que puede no resultar obvia incluso si está familiarizado con las interfaces como concepto de programación.

AWS CDK Soporta el uso de recursos definidos fuera de las aplicaciones de CDK utilizando métodos como`Bucket.fromBucketArn()`. Los recursos externos no se pueden modificar y es posible que no tengan todas las funciones disponibles con los recursos definidos en su aplicación de CDK utilizando, por ejemplo, la `Bucket` clase. Por lo tanto, las interfaces representan la funcionalidad mínima disponible en la CDK para un tipo de AWS recurso determinado, incluidos los recursos externos.

Por lo tanto, al crear instancias de recursos en tu aplicación de CDK, siempre debes usar clases concretas, como. `Bucket` Cuando especifiques el tipo de argumento que vas a aceptar en una de tus propias construcciones, usa el tipo de interfaz, por ejemplo, `IBucket` si estás preparado para trabajar con recursos externos (es decir, no necesitarás cambiarlos). Si necesita una construcción definida por CDK, especifique el tipo más general que pueda utilizar.

Algunas interfaces son versiones mínimas de propiedades o paquetes de opciones asociados a clases específicas, en lugar de construcciones. Estas interfaces pueden resultar útiles a la hora de subclasificar para aceptar argumentos que pasarás a tu clase principal. Si necesita una o más propiedades adicionales, querrá implementarla o derivarla de esta interfaz, o de un tipo más específico.

> **Note**
>
> Algunos lenguajes de programación compatibles AWS CDK no tienen ninguna función de interfaz. En estos lenguajes, las interfaces son simplemente clases normales. Puede identificarlos por sus nombres, que siguen el patrón de una «I» inicial seguida del nombre de alguna otra construcción (por ejemplo`IBucket`). Se aplican las mismas reglas.

Administrar las dependencias

Las dependencias de tu AWS CDK aplicación o biblioteca se administran mediante herramientas de administración de paquetes. Estas herramientas se utilizan habitualmente con los lenguajes de programación.

Por lo general, es AWS CDK compatible con la herramienta de administración de paquetes estándar u oficial del lenguaje, si existe. De lo contrario, AWS CDK será compatible con el idioma más popular o más compatible del idioma. Es posible que también puedas usar otras herramientas, especialmente si funcionan con las herramientas compatibles. Sin embargo, el soporte oficial para otras herramientas es limitado.

AWS CDK Es compatible con los siguientes administradores de paquetes:

Idioma	Herramienta de administración de paquetes compatible
TypeScript/JavaScript	NPM (Node Package Manager) o Yarn
Python	PIP (Package Installer para Python)
Java	Maven
C#	NuGet
Go	Módulos Go

Al crear un nuevo proyecto mediante el AWS CDK CLI `cdk init` comando, las dependencias de las bibliotecas principales y las construcciones estables del CDK se especifican automáticamente.

Para obtener más información sobre la administración de las dependencias de los lenguajes de programación compatibles, consulte lo siguiente:

- Administrar las dependencias en TypeScript.
- Administrar las dependencias en JavaScript.
- Administrar las dependencias en Python.
- Administrar las dependencias en Java.
- Administrar las dependencias en C#.
- Administrar las dependencias en Go.

Comparación AWS CDKTypeScript con otros lenguajes

TypeScript fue el primer lenguaje compatible para el desarrollo de AWS CDK aplicaciones. Por lo tanto, está escrita una cantidad sustancial de código CDK de ejemplo. TypeScript Si estás desarrollando en otro lenguaje, puede ser útil comparar la forma en que se implementa el AWS CDK código en TypeScript comparación con el idioma de tu elección. Esto puede ayudarle a utilizar los ejemplos de toda la documentación.

Importación de un módulo

TypeScript/JavaScript

TypeScript admite la importación de un espacio de nombres completo o de objetos individuales de un espacio de nombres. Cada espacio de nombres incluye componentes fijos y otras clases para su uso con un servicio determinado. AWS

```
// Import main CDK library as cdk
import * as cdk from 'aws-cdk-lib';   // ES6 import preferred in TS
const cdk = require('aws-cdk-lib');   // Node.js require() preferred in JS

// Import specific core CDK classes
import { Stack, App } from 'aws-cdk-lib';
const { Stack, App } = require('aws-cdk-lib');

// Import AWS S3 namespace as s3 into current namespace
import { aws_s3 as s3 } from 'aws-cdk-lib';   // TypeScript
const s3 = require('aws-cdk-lib/aws-s3');     // JavaScript

// Having imported cdk already as above, this is also valid
const s3 = cdk.aws_s3;

// Now use s3 to access the S3 types
const bucket = s3.Bucket(...);

// Selective import of s3.Bucket
import { Bucket } from 'aws-cdk-lib/aws-s3';        // TypeScript
const { Bucket } = require('aws-cdk-lib/aws-s3');   // JavaScript

// Now use Bucket to instantiate an S3 bucket
const bucket = Bucket(...);
```

Python

Por ejemplo TypeScript, Python admite la importación de módulos con espacios de nombres y las importaciones selectivas. Los espacios de nombres en Python se parecen a aws_cdk. xxx, donde xxx representa el nombre de un AWS servicio, como s3 para Amazon S3. (Amazon S3 se utiliza en estos ejemplos).

```
# 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

Las importaciones de Java funcionan de forma diferente a las TypeScript de Java. Cada sentencia de importación importa un único nombre de clase de un paquete determinado o todas las clases definidas en ese paquete (mediante`*`). Se puede acceder a las clases utilizando el nombre de la clase en sí mismo si se ha importado, o el nombre de la clase cualificada, incluido su paquete.

Las bibliotecas reciben el mismo `software.amazon.awscdk.services.xxx` nombre que la biblioteca AWS Construct (la biblioteca principal es`software.amazon.awscdk`). El identificador de grupo de Maven para AWS CDK los paquetes es`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#

En C#, los tipos se importan con la `using` directiva. Hay dos estilos. Uno le da acceso a todos los tipos del espacio de nombres especificado utilizando sus nombres simples. Con el otro, puede hacer referencia al propio espacio de nombres mediante un alias.

Los paquetes reciben el mismo nombre que los paquetes `Amazon.CDK.AWS.xxx` de AWS Construct Library. (El módulo principal es`Amazon.CDK`.)

```
// Make CDK base classes available under cdk
using cdk = Amazon.CDK;

// Make all Amazon S3 construct library classes available
using Amazon.CDK.AWS.S3;

// Now we can access any S3 type using its name
var bucket = new Bucket(...);

// Import the S3 namespace under an alias
using s3 = Amazon.CDK.AWS.S3;

// Now we can access an S3 type through the namespace alias
var bucket = new s3.Bucket(...);

// We can always use the qualified name of a type (including its namespace) even
 without a
// using directive
var bucket = new Amazon.CDK.AWS.S3.Bucket(...)
```

Go

Cada módulo de AWS Construct Library se proporciona como un paquete 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(...)
```

Creación de una instancia de una construcción

AWS CDK las clases de construcción tienen el mismo nombre en todos los lenguajes compatibles. La mayoría de los lenguajes usan la `new` palabra clave para crear instancias de una clase (Python y Go no lo hacen). Además, en la mayoría de los lenguajes, la palabra clave `this` se refiere a la instancia actual. (Python se usa `self` por convención). Deberías pasar una referencia a la instancia actual como `scope` parámetro a cada construcción que crees.

El tercer argumento de una AWS CDK construcción es `props` un objeto que contiene los atributos necesarios para construir la construcción. Este argumento puede ser opcional, pero cuando es necesario, los lenguajes compatibles lo manejan de forma idiomática. Los nombres de los atributos también se adaptan a los patrones de nomenclatura estándar del idioma.

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 no usa una `new` palabra clave al crear instancias de una clase. El argumento de propiedades se representa mediante argumentos de palabras clave y los argumentos se nombran mediante. `snake_case`

Si un valor de props es en sí mismo un conjunto de atributos, se representa mediante una clase con el nombre de la propiedad, que acepta argumentos de palabras clave para las subpropiedades.

En Python, la instancia actual se pasa a los métodos como primer argumento, que se denomina `self` por convención.

```
# 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

En Java, el argumento props se representa mediante una clase denominada `XxxxProps` (por ejemplo, `BucketProps` para los props de la `Bucket` construcción). El argumento props se construye utilizando un patrón de construcción.

Cada `XxxxProps` clase tiene un constructor. También hay un práctico constructor para cada construcción que construye los accesorios y la construcción en un solo paso, como se muestra en el siguiente ejemplo.

Los accesorios reciben el mismo nombre que en TypeScript, usando. `camelCase`

```
// Instantiate default Bucket
Bucket bucket = Bucket(self, "MyBucket");

// Instantiate Bucket with bucketName and versioned properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .bucketName("my-bucket").versioned(true)
                      .build();

# Instantiate Bucket with websiteRedirect, which has its own sub-properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .websiteRedirect(new websiteRedirect.Builder()
                          .hostName("aws.amazon.com").build())
                      .build();
```

C#

En C#, los accesorios se especifican mediante un inicializador de objetos para una clase llamada `XxxxProps` (por ejemplo, `BucketProps` para los accesorios de la `Bucket` construcción).

Los accesorios se nombran de forma similar a, excepto por el uso de. TypeScript `PascalCase`

Es conveniente utilizar la `var` palabra clave al crear una instancia de una construcción, por lo que no es necesario escribir el nombre de la clase dos veces. Sin embargo, la guía de estilo del código local puede variar.

```
// Instantiate default Bucket
var bucket = Bucket(self, "MyBucket");

// Instantiate Bucket with BucketName and Versioned properties
var bucket =  Bucket(self, "MyBucket", new BucketProps {
                      BucketName = "my-bucket",
                      Versioned  = true});

// Instantiate Bucket with WebsiteRedirect, which has its own sub-properties
var bucket = Bucket(self, "MyBucket", new BucketProps {
                      WebsiteRedirect = new WebsiteRedirect {
                              HostName = "aws.amazon.com"
                      }});
```

Go

Para crear una construcción en Go, llama a la función `NewXxxxxx` donde `Xxxxxxx` es el nombre de la construcción. Las propiedades de las construcciones se definen como una estructura.

En Go, todos los parámetros de construcción son punteros, incluidos valores como números, valores booleanos y cadenas. Usa funciones prácticas como `jsii.String` para crear estos punteros.

```
// 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"),
 }})
```

Acceder a los miembros

Es habitual hacer referencia a los atributos o propiedades de los componentes fijos y otras AWS CDK clases y utilizar estos valores como, por ejemplo, entradas para crear otros componentes fijos. Las diferencias de nomenclatura descritas anteriormente para los métodos también se aplican aquí. Además, en Java, no es posible acceder directamente a los miembros. En su lugar, se proporciona un método de captación.

TypeScript/JavaScript

Los nombres son `camelCase`.

```
bucket.bucketArn
```

Python

Los nombres son`snake_case`.

```
bucket.bucket_arn
```

Java

Se proporciona un método de captación para cada propiedad; estos nombres son`camelCase`.

```
bucket.getBucketArn()
```

C#

Los nombres son`PascalCase`.

```
bucket.BucketArn
```

Go

Los nombres son`PascalCase`.

```
bucket.BucketArn
```

Constantes de enumeración

Las constantes de enumeración se refieren a una clase y tienen nombres en mayúscula con guiones bajos en todos los idiomas (a veces denominadas así). `SCREAMING_SNAKE_CASE` Como los nombres de clase también utilizan las mismas mayúsculas y minúsculas en todos los idiomas compatibles, excepto Go, los nombres de enumeración válidos también son los mismos en estos idiomas.

```
s3.BucketEncryption.KMS_MANAGED
```

En Go, las constantes de enumeración son atributos del espacio de nombres del módulo y se escriben de la siguiente manera.

```
awss3.BucketEncryption_KMS_MANAGED
```

Interfaces de objetos

AWS CDK Utiliza interfaces de TypeScript objetos para indicar que una clase implementa un conjunto esperado de métodos y propiedades. Puede reconocer una interfaz de objetos porque su nombre empieza por`I`. Una clase concreta indica las interfaces que implementa mediante la `implements` palabra clave.

TypeScript/JavaScript

> **Note**
>
> JavaScript no tiene una función de interfaz. Puede ignorar la `implements` palabra clave y los nombres de clase que la siguen.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';

class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

Python no tiene una función de interfaz. Sin embargo, para eso AWS CDK puedes indicar la implementación de la interfaz decorando tu clase con ella`@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

Las estructuras Go no necesitan declarar explícitamente qué interfaces implementan. El compilador Go determina la implementación en función de los métodos y propiedades disponibles en la estructura. Por ejemplo, en el código siguiente, `MyAspect` implementa la `IAspect` interfaz porque proporciona un `Visit` método que toma una construcción.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
 fmt.Println("Visited", *node.Node().Path())
}
```

Trabajando con la AWS CDK tinta TypeScript

TypeScript es un idioma de cliente totalmente compatible AWS Cloud Development Kit (AWS CDK) y se considera estable. Para trabajar con el AWS CDK in se TypeScript utilizan herramientas conocidas, como el TypeScript compilador (`tsc`) de Microsoft, Node.js y el Node Package Manager (`npm`). También puedes usar Yarn si lo prefieres, aunque en los ejemplos de esta guía se usa NPM.

Los módulos que componen la biblioteca AWS Construct se distribuyen a través del repositorio de NPM, npmjs.org.

Puede utilizar cualquier editor o IDE. Muchos AWS CDK desarrolladores utilizan Visual Studio Code (o su equivalente de código abierto, VSCodium), que cuenta con un excelente soporte. TypeScript

Temas

- Comience con TypeScript
- Creación de un proyecto
- Utilización de elementos locales tsc y cdk
- Administrar los módulos de AWS Construct Library
- Administrar las dependencias en TypeScript
- AWS CDK modismos en TypeScript
- Construir, sintetizar e implementar

Comience con TypeScript

Para trabajar con el AWS CDK, debe tener una AWS cuenta y credenciales y haber instalado Node.js y el AWS CDK kit de herramientas. Consulte Cómo empezar con el AWS CDK.

También TypeScript se necesita a sí mismo (versión 3.8 o posterior). Si aún no lo tiene, puede instalarlo usandonpm.

```
npm install -g typescript
```

> **Note**
>
> Si aparece un error de permiso y tienes acceso de administrador a tu sistema, inténtalosudo `npm install -g typescript`.

TypeScript Mantente al día con un habitualnpm `update -g typescript`.

> **Note**
>
> Degradación de idiomas de terceros: la versión en otros idiomas solo se admite hasta que el proveedor o la comunidad compartan su fecha de caducidad (EOL) y está sujeta a cambios con previo aviso.

Creación de un proyecto

Para crear un AWS CDK proyecto nuevo, se invoca `cdk init` en un directorio vacío. Utilice la `--language` opción y especifique`typescript`:

```
mkdir my-project
cd my-project
cdk init app --language typescript
```

Al crear un proyecto, también se instala el `aws-cdk-lib`módulo y sus dependencias.

`cdk init`usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debe tener la forma de un TypeScript identificador; por ejemplo, no debe empezar por un número ni contener espacios.

Utilización de elementos locales **`tsc`** y **`cdk`**

En su mayor parte, en esta guía se supone que se instala TypeScript el kit de herramientas del CDK de forma global (`npm install -g typescript aws-cdk`), y los ejemplos de comandos proporcionados (por ejemplo`cdk synth`) siguen esta suposición. Este enfoque facilita el mantenimiento de ambos componentes actualizados y, dado que ambos adoptan un enfoque estricto en materia de compatibilidad con versiones anteriores, por lo general, se corre poco riesgo al utilizar siempre las versiones más recientes.

Algunos equipos prefieren especificar todas las dependencias de cada proyecto, incluidas herramientas como el TypeScript compilador y el kit de herramientas CDK. Esta práctica le permite vincular estos componentes a versiones específicas y garantizar que todos los desarrolladores de su equipo (y de su entorno de CI/CD) utilicen exactamente esas versiones. Esto elimina una posible fuente de cambio, lo que ayuda a que las compilaciones e implementaciones sean más consistentes y repetibles.

La CDK incluye dependencias tanto para ella como TypeScript para el kit de herramientas de la CDK en la plantilla del TypeScript proyecto`package.json`, por lo que si quieres utilizar este enfoque, no necesitas realizar ningún cambio en el proyecto. Todo lo que necesitas hacer es usar comandos ligeramente diferentes para crear tu aplicación y para emitir comandos. `cdk`

Operación	Usa herramientas globales	Utilice herramientas locales
Inicializa el proyecto	`cdk init --language typescript`	`npx aws-cdk init --language typescript`
Compilación	`tsc`	`npm run build`
Ejecute el comando CDK Toolkit	`cdk ...`	`npm run cdk ...` or `npx aws-cdk ...`

`npx aws-cdk`ejecuta la versión del CDK Toolkit instalada localmente en el proyecto actual, si existe, y recurre a la instalación global, si la hubiera. Si no existe una instalación global, `npx` descarga una copia temporal del kit de herramientas del CDK y la ejecuta. Puede especificar una versión arbitraria del kit de herramientas del CDK mediante la `@` sintaxis: prints. `npx aws-cdk@2.0 --version 2.0.0`

> **Tip**
>
> Configure un alias para poder usar el `cdk` comando en una instalación local del CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Administrar los módulos de AWS Construct Library

Use el Node Package Manager (npm) para instalar y actualizar los módulos de AWS Construct Library para que los usen sus aplicaciones, así como otros paquetes que necesite. (Puede usarlo `yarn` en lugar de `npm` si lo prefiere). `npm`también instala las dependencias de esos módulos automáticamente.

La mayoría de AWS CDK las construcciones se encuentran en el paquete CDK principal, llamado`aws-cdk-lib`, que es una dependencia predeterminada en los nuevos proyectos creados por. cdk init Los módulos de la biblioteca de AWS construcciones «experimentales», en los que aún se están desarrollando construcciones de nivel superior, reciben el mismo nombre. `@aws-cdk/`*`SERVICE-NAME`*`-alpha` El nombre del servicio tiene el prefijo aws-. Si no está seguro del nombre de un módulo, búsquelo en NPM.

> **Note**
>
> La referencia de la API de CDK también muestra los nombres de los paquetes.

Por ejemplo, el siguiente comando instala el módulo experimental para. AWS CodeStar

```
npm install @aws-cdk/aws-codestar-alpha
```

El soporte de Construct Library de algunos servicios está en más de un espacio de nombres. Por ejemplo, además`aws-route53`, hay tres espacios de nombres adicionales de Amazon Route 53, `aws-route53-targets``aws-route53-patterns`, y. `aws-route53resolver`

Las dependencias de su proyecto se mantienen en. `package.json` Puedes editar este archivo para bloquear algunas o todas tus dependencias en una versión específica o para permitir que se actualicen a versiones más recientes según ciertos criterios. Para actualizar las dependencias de NPM de tu proyecto a la última versión permitida de acuerdo con las reglas que especificaste en: `package.json`

```
npm update
```

En TypeScript, importas módulos a tu código con el mismo nombre que utilizaste para instalarlos mediante NPM. Recomendamos las siguientes prácticas al importar AWS CDK clases y módulos de AWS Construct Library a sus aplicaciones. Seguir estas pautas ayudará a que su código sea coherente con el de otras AWS CDK aplicaciones y a que sea más fácil de entender.

- Utilice `import` directivas al estilo de ES6, no. `require()`
- Por lo general, importe clases individuales de. `aws-cdk-lib`

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

- Si necesita muchas clases`aws-cdk-lib`, puede utilizar un alias de espacio de nombres `cdk` en lugar de importar las clases individuales. Evita hacer ambas cosas.

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

- Por lo general, las construcciones AWS de servicios de importación utilizan alias de espacios de nombres cortos.

```
import { aws_s3 as s3 } from 'aws-cdk-lib';
```

Administrar las dependencias en TypeScript

En los proyectos de TypeScript CDK, las dependencias se especifican en el `package.json` archivo del directorio principal del proyecto. Los AWS CDK módulos principales se encuentran en un único NPM paquete llamado. `aws-cdk-lib`

Cuando instala un paquete utilizandonpm install, NPM graba el paquete `package.json` por usted.

Si lo prefieres, puedes usar Yarn en lugar de NPM. Sin embargo, el CDK no admite el plug-and-play modo de Yarn, que es el modo predeterminado en Yarn 2. Agrega lo siguiente al `.yarnrc.yml` archivo de tu proyecto para desactivar esta función.

```
nodeLinker: node-modules
```

Aplicaciones CDK

El siguiente es un ejemplo de `package.json` archivo generado por el `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"
  }
}
```

En el caso de las aplicaciones CDK desplegables, `aws-cdk-lib` debe especificarse en la `dependencies` sección de. `package.json` Puede usar un especificador de número de versión con un signo de intercalación (^) para indicar que aceptará versiones posteriores a la especificada, siempre y cuando estén dentro de la misma versión principal.

En el caso de las construcciones experimentales, especifique las versiones exactas de los módulos de la biblioteca de construcciones alfa, que tienen API que pueden cambiar. No utilices ^ ni ~, ya que las versiones posteriores de estos módulos pueden incluir cambios en la API que pueden dañar tu aplicación.

Especifica las versiones de las bibliotecas y herramientas necesarias para probar tu aplicación (por ejemplo, el marco de `jest` pruebas) en la `devDependencies` sección de`package.json`. Si lo desea, utilice ^ para especificar si se aceptan versiones posteriores compatibles.

Bibliotecas de construcción de terceros

Si está desarrollando una biblioteca de construcción, especifique sus dependencias mediante una combinación de las `devDependencies` secciones `peerDependencies` y, como se muestra en el siguiente `package.json` archivo de ejemplo.

```
{
  "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"
  }
}
```

En`peerDependencies`, utilice un signo de intercalación (^) para especificar la versión más baja con la `aws-cdk-lib` que funciona la biblioteca. Esto maximiza la compatibilidad de la biblioteca con una variedad de versiones de CDK. Especifique las versiones exactas de los módulos de la biblioteca Alpha Construct, que tienen API que pueden cambiar. El uso `peerDependencies` garantiza que solo haya una copia de todas las bibliotecas de CDK en el `node_modules` árbol.

En`devDependencies`, especifique las herramientas y bibliotecas que necesita para realizar las pruebas, si lo desea, con ^ para indicar que se aceptan versiones posteriores compatibles. Especifique exactamente (sin ^ ni ~) las versiones más bajas `aws-cdk-lib` y otros paquetes de CDK con los que anuncia que su biblioteca es compatible. Esta práctica garantiza que las pruebas se ejecuten con esas versiones. De esta forma, si utilizas inadvertidamente una función que solo se encuentra en las versiones más recientes, tus pruebas pueden detectarla.

> **Warning**
>
> `peerDependencies`se instalan automáticamente solo en NPM 7 y versiones posteriores. Si usa NPM 6 o una versión anterior, o si usa Yarn, debe incluir las dependencias de sus dependencias en. `devDependencies` De lo contrario, no se instalarán y recibirás una advertencia sobre las dependencias entre pares no resueltas.

Instalación y actualización de las dependencias

Ejecuta el siguiente comando para instalar las dependencias de tu proyecto.

NPM

```
# Install the latest version of everything that matches the ranges in 'package.json'
npm install

# Install the same exact dependency versions as recorded in 'package-lock.json'
npm ci
```

Yarn

```
# Install the latest version of everything that matches the ranges in 'package.json'
yarn upgrade

# Install the same exact dependency versions as recorded in 'yarn.lock'
yarn install --frozen-lockfile
```

Para actualizar los módulos instalados, se pueden usar yarn upgrade los comandos anteriores npm install y. Cualquiera de los dos comandos actualiza los paquetes `node_modules` a las versiones más recientes que cumplen las reglas de`package.json`. Sin embargo, no `package.json` se actualizan solos, por lo que puede que desee establecer una nueva versión mínima. Si alojas tu paquete GitHub, puedes configurar las actualizaciones de las versiones del Dependabot para que se actualicen automáticamente. `package.json` Para otras opciones, consulte npm-check-updates.

> **Important**
>
> Por diseño, al instalar o actualizar las dependencias, NPM y Yarn eligen la última versión de cada paquete que cumpla con los requisitos especificados en. `package.json` Siempre existe el riesgo de que estas versiones se rompan (de forma accidental o intencionada). Realice pruebas exhaustivas después de actualizar las dependencias de su proyecto.

AWS CDK modismos en TypeScript

Utilería

Todas las clases de AWS Construct Library se instancian mediante tres argumentos: el ámbito en el que se define la construcción (su elemento principal en el árbol de construcciones), un identificador y los accesorios. Argument props es un conjunto de pares clave/valor que la construcción utiliza para configurar los recursos que crea. AWS Otras clases y métodos también utilizan el patrón de «paquete de atributos» para los argumentos.

En TypeScript, la forma de `props` se define mediante una interfaz que indica los argumentos obligatorios y opcionales y sus tipos. Esta interfaz se define para cada tipo de `props` argumento y, por lo general, es específica de una sola construcción o método. Por ejemplo, la construcción Bucket (en`aws-cdk-lib/aws-s3 module`) especifica un `props` argumento que se ajusta a la BucketPropsinterfaz.

Si una propiedad es en sí misma un objeto, por ejemplo, la propiedad WebsiteRedirect de`BucketProps`, ese objeto tendrá su propia interfaz a la que deberá ajustarse su forma, en este caso. RedirectTarget

Si está subclasificando una clase de AWS Construct Library (o anulando un método que utiliza un argumento similar a un objeto), puede heredarlo de la interfaz existente para crear una nueva que especifique los accesorios nuevos que necesite el código. Al llamar a la clase principal o al método base, normalmente puedes pasar todo el argumento props que hayas recibido, ya que se ignorarán todos los atributos proporcionados en el objeto pero no especificados en la interfaz.

Una futura versión del AWS CDK podría añadir casualmente una nueva propiedad con el nombre que utilizó para su propia propiedad. Pasar el valor que reciba a la cadena de herencia puede provocar un comportamiento inesperado. Es más seguro entregar una copia superficial de los accesorios que recibiste con tus bienes retirados o puestos a `undefined` punto. Por ejemplo:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Como alternativa, ponle un nombre a tus propiedades para que quede claro que pertenecen a tu construcción. De esta forma, es poco probable que choquen con propiedades en futuras AWS CDK versiones. Si hay muchos de ellos, utilice un único objeto con el nombre apropiado para sostenerlos.

Valores faltantes

Los valores que faltan en un objeto (como los accesorios) tienen el valor en. `undefined` TypeScript La versión 3.7 del lenguaje introdujo operadores que simplifican el trabajo con estos valores, lo que facilita la especificación de los valores predeterminados y el encadenamiento de «cortocircuitos» cuando se alcanza un valor indefinido. Para obtener más información sobre estas funciones, consulte las notas de la versión TypeScript 3.7, específicamente las dos primeras funciones, el encadenamiento opcional y la fusión nula.

Construir, sintetizar e implementar

Por lo general, debes estar en el directorio raíz del proyecto al crear y ejecutar la aplicación.

Node.js no se puede ejecutar TypeScript directamente; en su lugar, la aplicación se convierte para que JavaScript utilice el TypeScript compilador,`tsc`. A continuación, se ejecuta el JavaScript código resultante.

Lo hace AWS CDK automáticamente siempre que necesite ejecutar tu aplicación. Sin embargo, puede resultar útil compilar manualmente para comprobar si hay errores y ejecutar pruebas. Para compilar TypeScript la aplicación manualmente, ejecutenpm `run build`. También puedes `npm run watch` introducir el modo de visualización, en el que el TypeScript compilador reconstruye automáticamente tu aplicación cada vez que guardas los cambios en un archivo fuente.

Las pilas definidas en tu AWS CDK aplicación se pueden sintetizar e implementar de forma individual o conjunta mediante los siguientes comandos. Por lo general, debes estar en el directorio principal de tu proyecto cuando los emitas.

- `cdk synth`: sintetiza una AWS CloudFormation plantilla a partir de una o más de las pilas de tu AWS CDK aplicación.
- `cdk deploy`: despliega los recursos definidos por una o más de las pilas de tu aplicación en. AWS CDK AWS

Puedes especificar los nombres de varias pilas que se van a sintetizar o implementar en un solo comando. Si tu aplicación define solo una pila, no es necesario que la especifiques.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

También puedes usar los caracteres comodín * (cualquier número de caracteres) y? (cualquier carácter individual) para identificar las pilas por patrón. Cuando utilice caracteres comodín, escriba el patrón entre comillas. De lo contrario, es posible que el shell intente ampliarlo hasta incluir los nombres de los archivos del directorio actual antes de pasarlos al AWS CDK kit de herramientas.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> No necesita sintetizar las pilas de forma explícita antes de desplegarlas; `cdk deploy` realiza este paso para asegurarse de que se implementa el código más reciente.

Para obtener la documentación completa del `cdk` comando, consulte. the section called “AWS CDK Kit de herramientas”

Trabajando con la AWS CDK tinta JavaScript

JavaScriptes un lenguaje de cliente totalmente compatible AWS CDK y se considera estable. Para trabajar con AWS Cloud Development Kit (AWS CDK) él se JavaScript utilizan herramientas conocidas, como Node.js y Node Package Manager (`npm`). También puedes usar Yarn si lo prefieres, aunque en los ejemplos de esta guía se usa NPM. Los módulos que componen la biblioteca AWS Construct se distribuyen a través del repositorio de NPM, npmjs.org.

Puede utilizar cualquier editor o IDE. Muchos AWS CDK desarrolladores utilizan Visual Studio Code (o su equivalente de código abierto, VSCodium), que cuenta con un buen soporte. JavaScript

Temas

- Introducción a JavaScript
- Creación de un proyecto
- Uso de local cdk
- Administrar los módulos de AWS Construct Library
- Administrar las dependencias en JavaScript
- AWS CDK modismos en JavaScript
- Sintetizar y desplegar

- Uso de TypeScript ejemplos con JavaScript
- ¿Migrar a TypeScript

Introducción a JavaScript

Para trabajar con él AWS CDK, debe tener una AWS cuenta y credenciales y tener instalados Node.js y el kit de herramientas. AWS CDK Consulte Cómo empezar con el AWS CDK.

JavaScript AWS CDK las aplicaciones no requieren requisitos previos adicionales más allá de estos.

> **Note**
>
> El uso de idiomas de terceros está en desuso: la versión en otros idiomas solo se admite hasta que el proveedor o la comunidad compartan su fecha de caducidad (EOL), y está sujeta a cambios con previo aviso.

Creación de un proyecto

Para crear un AWS CDK proyecto nuevo, se invoca `cdk init` en un directorio vacío. Utilice la `--language` opción y especifique`javascript`:

```
mkdir my-project
cd my-project
cdk init app --language javascript
```

Al crear un proyecto, también se instala el `aws-cdk-lib`módulo y sus dependencias.

`cdk init`usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debe tener la forma de un JavaScript identificador; por ejemplo, no debe empezar por un número ni contener espacios.

Uso de local **`cdk`**

En su mayor parte, en esta guía se supone que se instala el kit de herramientas CDK de forma global (`npm install -g aws-cdk`), y los ejemplos de comandos proporcionados (como`cdk synth`) siguen esta suposición. Este enfoque facilita mantener actualizado el kit de herramientas del CDK y,

dado que el CDK adopta un enfoque estricto en cuanto a la compatibilidad con versiones anteriores, por lo general, se corre poco riesgo al utilizar siempre la última versión.

Algunos equipos prefieren especificar todas las dependencias de cada proyecto, incluidas herramientas como el kit de herramientas del CDK. Esta práctica le permite fijar dichos componentes a versiones específicas y garantizar que todos los desarrolladores de su equipo (y de su entorno de CI/CD) utilicen exactamente esas versiones. Esto elimina una posible fuente de cambio, lo que ayuda a que las compilaciones y las implementaciones sean más consistentes y repetibles.

El CDK incluye una dependencia para el kit de herramientas del CDK en las plantillas del JavaScript proyecto`package.json`, por lo que si quieres utilizar este enfoque, no necesitas realizar ningún cambio en el proyecto. Todo lo que necesitas hacer es usar comandos ligeramente diferentes para crear tu aplicación y para emitir comandos. `cdk`

Operación	Usa el kit de herramientas CDK global	Utilice el kit de herramientas CDK local
Inicialice el proyecto	`cdk init --idioma javascript`	`npx aws-cdk init --idioma javascript`
Ejecute el comando CDK Toolkit	`cdk...`	`npm run cdk...` or `npx aws-cdk...`

`npx aws-cdk`ejecuta la versión del kit de herramientas CDK instalada localmente en el proyecto actual, si existe, y recurre a la instalación global, si la hubiera. Si no existe una instalación global, `npx` descarga una copia temporal del kit de herramientas del CDK y la ejecuta. Puede especificar una versión arbitraria del kit de herramientas del CDK mediante la `@` sintaxis: prints. `npx aws-cdk@1.120 --version` `1.120.0`

> **Tip**
>
> Configure un alias para poder usar el `cdk` comando en una instalación local del CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```

Windows

```
doskey cdk=npx aws-cdk $*
```

Administrar los módulos de AWS Construct Library

Use el Node Package Manager (npm) para instalar y actualizar los módulos de AWS Construct Library para que los usen sus aplicaciones, así como otros paquetes que necesite. (Puede usarlo `yarn` en lugar de `npm` si lo prefiere). `npm`también instala las dependencias de esos módulos automáticamente.

La mayoría de AWS CDK las construcciones se encuentran en el paquete CDK principal, llamado`aws-cdk-lib`, que es una dependencia predeterminada en los nuevos proyectos creados por. cdk init Los módulos de la biblioteca de AWS construcciones «experimentales», en los que aún se están desarrollando construcciones de nivel superior, reciben el mismo nombre. `aws-cdk-lib/`*`SERVICE-NAME`*`-alpha` El nombre del servicio tiene el prefijo aws-. Si no está seguro del nombre de un módulo, búsquelo en NPM.

> **Note**
>
> La referencia de la API de CDK también muestra los nombres de los paquetes.

Por ejemplo, el siguiente comando instala el módulo experimental para. AWS CodeStar

```
npm install @aws-cdk/aws-codestar-alpha
```

El soporte de Construct Library de algunos servicios está en más de un espacio de nombres. Por ejemplo, además`aws-route53`, hay tres espacios de nombres adicionales de Amazon Route 53, `aws-route53-targets`aws-route53-patterns, y. `aws-route53resolver`

Las dependencias de su proyecto se mantienen en. `package.json` Puedes editar este archivo para bloquear algunas o todas tus dependencias en una versión específica o para permitir que se actualicen a versiones más recientes según ciertos criterios. Para actualizar las dependencias de NPM de tu proyecto a la última versión permitida de acuerdo con las reglas que especificaste en: `package.json`

```
npm update
```

En JavaScript, importas módulos a tu código con el mismo nombre que utilizaste para instalarlos mediante NPM. Recomendamos las siguientes prácticas al importar AWS CDK clases y módulos de AWS Construct Library a sus aplicaciones. Seguir estas pautas ayudará a que su código sea coherente con el de otras AWS CDK aplicaciones y a que sea más fácil de entender.

- Utilice`require()`, no directivas al estilo de ES6. `import` Las versiones anteriores de Node.js no admiten las importaciones de ES6, por lo que el uso de la sintaxis anterior es más compatible. (Si realmente desea utilizar las importaciones de ES6, utilice esm para asegurarse de que su proyecto es compatible con todas las versiones compatibles de Node.js).
- Por lo general, importe clases individuales desde. `aws-cdk-lib`

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

- Si necesita muchas clases`aws-cdk-lib`, puede utilizar un alias de espacio de nombres `cdk` en lugar de importar las clases individuales. Evita hacer ambas cosas.

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

- Por lo general, importe las bibliotecas de AWS Construct utilizando alias de espacios de nombres cortos.

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

Administrar las dependencias en JavaScript

En los proyectos de JavaScript CDK, las dependencias se especifican en el `package.json` archivo del directorio principal del proyecto. Los AWS CDK módulos principales se encuentran en un único NPM paquete llamado. `aws-cdk-lib`

Cuando instala un paquete utilizandonpm install, NPM graba el paquete `package.json` por usted.

Si lo prefieres, puedes usar Yarn en lugar de NPM. Sin embargo, el CDK no admite el plug-and-play modo de Yarn, que es el modo predeterminado en Yarn 2. Agrega lo siguiente al `.yarnrc.yml` archivo de tu proyecto para desactivar esta función.

```
nodeLinker: node-modules
```

Aplicaciones CDK

El siguiente es un ejemplo de `package.json` archivo generado por el `cdk init --language typescript` comando. El archivo generado para JavaScript es similar, solo que sin las entradas TypeScript relacionadas.

```
{
  "name": "my-package",
  "version": "0.1.0",
  "bin": {
    "my-package": "bin/my-package.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^26.0.10",
    "@types/node": "10.17.27",
    "jest": "^26.4.2",
    "ts-jest": "^26.2.0",
    "aws-cdk": "2.16.0",
    "ts-node": "^9.0.0",
    "typescript": "~3.9.7"
  },
  "dependencies": {
    "aws-cdk-lib": "2.16.0",
    "constructs": "^10.0.0",
    "source-map-support": "^0.5.16"
  }
}
```

En el caso de las aplicaciones de CDK desplegables, `aws-cdk-lib` debe especificarse en la `dependencies` sección de. `package.json` Puede usar un especificador de número de versión con un signo de intercalación (^) para indicar que aceptará versiones posteriores a la especificada, siempre y cuando estén dentro de la misma versión principal.

En el caso de las construcciones experimentales, especifique las versiones exactas de los módulos de la biblioteca de construcciones alfa, que tienen API que pueden cambiar. No utilices ^ ni ~, ya que las versiones posteriores de estos módulos pueden incluir cambios en la API que pueden dañar tu aplicación.

Especifica las versiones de las bibliotecas y herramientas necesarias para probar tu aplicación (por ejemplo, el marco de `jest` pruebas) en la `devDependencies` sección de`package.json`. Si lo desea, utilice ^ para especificar si se aceptan versiones posteriores compatibles.

Bibliotecas de construcción de terceros

Si está desarrollando una biblioteca de construcción, especifique sus dependencias mediante una combinación de las `devDependencies` secciones `peerDependencies` y, como se muestra en el siguiente `package.json` archivo de ejemplo.

```
{
  "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"
  }
}
```

En`peerDependencies`, utilice un signo de intercalación (^) para especificar la versión más baja con la `aws-cdk-lib` que funciona la biblioteca. Esto maximiza la compatibilidad de la biblioteca con una variedad de versiones de CDK. Especifique las versiones exactas de los módulos de la biblioteca Alpha Construct, que tienen API que pueden cambiar. El uso `peerDependencies` garantiza que solo haya una copia de todas las bibliotecas de CDK en el `node_modules` árbol.

En`devDependencies`, especifique las herramientas y bibliotecas que necesita para realizar las pruebas, si lo desea, con ^ para indicar que se aceptan versiones posteriores compatibles.

Especifique exactamente (sin ^ ni ~) las versiones más bajas `aws-cdk-lib` y otros paquetes de CDK con los que anuncia que su biblioteca es compatible. Esta práctica garantiza que las pruebas se ejecuten con esas versiones. De esta forma, si utilizas inadvertidamente una función que solo se encuentra en las versiones más recientes, tus pruebas pueden detectarla.

> **Warning**
>
> `peerDependencies`se instalan automáticamente solo en NPM 7 y versiones posteriores. Si usa NPM 6 o una versión anterior, o si usa Yarn, debe incluir las dependencias de sus dependencias en. `devDependencies` De lo contrario, no se instalarán y recibirás una advertencia sobre las dependencias entre pares no resueltas.

Instalación y actualización de las dependencias

Ejecuta el siguiente comando para instalar las dependencias de tu proyecto.

NPM

```
# Install the latest version of everything that matches the ranges in 'package.json'
npm install

# Install the same exact dependency versions as recorded in 'package-lock.json'
npm ci
```

Yarn

```
# Install the latest version of everything that matches the ranges in 'package.json'
yarn upgrade

# Install the same exact dependency versions as recorded in 'yarn.lock'
yarn install --frozen-lockfile
```

Para actualizar los módulos instalados, se pueden usar yarn upgrade los comandos anteriores npm install y. Cualquiera de los dos comandos actualiza los paquetes `node_modules` a las versiones más recientes que cumplen las reglas de`package.json`. Sin embargo, no `package.json` se actualizan solos, por lo que puede que desee establecer una nueva versión mínima. Si alojas tu paquete GitHub, puedes configurar las actualizaciones de las versiones del Dependabot para que se actualicen automáticamente. `package.json` Para otras opciones, consulte npm-check-updates.

> **Important**
>
> Por diseño, al instalar o actualizar las dependencias, NPM y Yarn eligen la última versión de cada paquete que cumpla con los requisitos especificados en. `package.json` Siempre existe el riesgo de que estas versiones se rompan (de forma accidental o intencionada). Realice pruebas exhaustivas después de actualizar las dependencias de su proyecto.

AWS CDK modismos en JavaScript

Accesorios

Todas las clases de AWS Construct Library se instancian mediante tres argumentos: el ámbito en el que se define la construcción (su elemento principal en el árbol de construcciones), un identificador y props, un conjunto de pares clave/valor que la construcción utiliza para configurar los recursos que crea. AWS Otras clases y métodos también utilizan el patrón de «conjunto de atributos» como argumento.

El uso de un IDE o un editor que tenga una buena JavaScript función de autocompletar ayudará a evitar errores ortográficos en los nombres de las propiedades. Si un componente fijo está esperando una `encryptionKeys` propiedad y la escribes al crear una instancia del componente fijo, no le has dado el valor deseado. `encryptionkeys` Esto puede provocar un error en el momento de la síntesis si la propiedad es necesaria o hacer que la propiedad se ignore silenciosamente si es opcional. En este último caso, es posible que obtenga un comportamiento predeterminado que pretendía anular. Tenga especial cuidado aquí.

Cuando subclasifique una clase de AWS Construct Library (o sustituya un método que utilice un argumento similar a un objeto), puede que desee aceptar propiedades adicionales para su propio uso. La clase principal o el método anulado ignorarán estos valores, ya que en ese código nunca se accede a ellos, por lo que, en general, puedes transferir todos los accesorios que hayas recibido.

Una futura versión del AWS CDK podría añadir casualmente una nueva propiedad con el nombre que utilizó para su propia propiedad. Pasar el valor que reciba a la cadena de herencia puede provocar un comportamiento inesperado. Es más seguro entregar una copia superficial de los accesorios que recibiste con tus bienes retirados o puestos a `undefined` punto. Por ejemplo:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Como alternativa, ponle un nombre a tus propiedades para que quede claro que pertenecen a tu construcción. De esta forma, es poco probable que choquen con propiedades en futuras AWS CDK versiones. Si hay muchos de ellos, utilice un único objeto con el nombre apropiado para sostenerlos.

Valores faltantes

Los valores que faltan en un objeto (por ejemplo`props`) tienen el valor en. `undefined` JavaScript Para tratarlos se utilizan las técnicas habituales. Por ejemplo, un modismo común para acceder a una propiedad de un valor que puede no estar definido es el siguiente:

```
// 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;
```

Sin embargo, si además a pudiera tener algún otro valor «falso»`undefined`, es mejor hacer la prueba más explícita. Aquí, aprovecharemos el hecho de que `null` `undefined` somos iguales para probar ambos a la vez:

```
let c = a == null ? a : a.b;
```

> **Tip**
>
> Node.js 14.0 y versiones posteriores admiten nuevos operadores que pueden simplificar el manejo de valores indefinidos. Para obtener más información, consulte las propuestas opcionales de encadenamiento y anulación de coalescentes.

Sintetizar y desplegar

Las pilas definidas en tu AWS CDK aplicación se pueden sintetizar e implementar de forma individual o conjunta mediante los siguientes comandos. Por lo general, debes estar en el directorio principal de tu proyecto cuando los emitas.

- `cdk synth`: sintetiza una AWS CloudFormation plantilla a partir de una o más de las pilas de tu AWS CDK aplicación.
- `cdk deploy`: despliega los recursos definidos por una o más de las pilas de tu aplicación en. AWS CDK AWS

Puedes especificar los nombres de varias pilas que se van a sintetizar o implementar en un solo comando. Si tu aplicación define solo una pila, no es necesario que la especifiques.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

También puedes usar los comodines * (cualquier número de caracteres) y? (cualquier carácter individual) para identificar las pilas por patrón. Cuando utilice caracteres comodín, escriba el patrón entre comillas. De lo contrario, es posible que el shell intente ampliarlo hasta incluir los nombres de los archivos del directorio actual antes de pasarlos al AWS CDK kit de herramientas.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> No necesita sintetizar las pilas de forma explícita antes de desplegarlas; `cdk deploy` realiza este paso para asegurarse de que se implementa el código más reciente.

Para obtener la documentación completa del `cdk` comando, consulte. the section called “AWS CDK Kit de herramientas”

Uso de TypeScript ejemplos con JavaScript

TypeScriptes el lenguaje que utilizamos para desarrollar y fue el AWS CDK primer lenguaje compatible para el desarrollo de aplicaciones, por lo que están escritos muchos ejemplos de AWS CDK código disponibles TypeScript. Estos ejemplos de código pueden ser un buen recurso para JavaScript los desarrolladores; basta con eliminar las partes TypeScript específicas del código.

TypeScript Los fragmentos suelen utilizar el ECMAScript más reciente `import` y `export` palabras clave para importar objetos de otros módulos y declarar que los objetos estarán disponibles fuera del módulo actual. Node.js acaba de empezar a admitir estas palabras clave en sus últimas versiones. En función de la versión de Node.js que utilice (o que desee admitir), puede reescribir las importaciones y exportaciones para utilizar la sintaxis anterior.

Las importaciones se pueden sustituir por llamadas a la `require()` función.

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');
```

Las exportaciones se pueden asignar al `module.exports` objeto.

TypeScript

```
export class Stack1 extends cdk.Stack {
  // ...
}

export class Stack2 extends cdk.Stack {
  // ...
}
```

JavaScript

```
class Stack1 extends cdk.Stack {
  // ...
}

class Stack2 extends cdk.Stack {
  // ...
}

module.exports = { Stack1, Stack2 }
```

> **Note**
>
> Una alternativa al uso de las importaciones y exportaciones al estilo antiguo es utilizar el `esm`módulo.

Una vez que hayas ordenado las importaciones y exportaciones, puedes profundizar en el código real. Es posible que te encuentres con estas funciones de uso común TypeScript :

- Escriba anotaciones
- Definiciones de interfaz
- Conversiones/conversiones de tipos
- Modificadores de acceso

Se pueden proporcionar anotaciones de tipo para las variables, los miembros de la clase, los parámetros de las funciones y los tipos de retorno de las funciones. En el caso de las variables, los parámetros y los miembros, los tipos se especifican siguiendo el identificador con dos puntos y el tipo. Los valores devueltos por las funciones siguen la firma de la función y se componen de dos puntos y el tipo.

Para convertir el código anotado en texto JavaScript, elimine los dos puntos y el tipo. Los miembros de la clase deben tener algún valor JavaScript; configúrelo en `undefined` si solo tienen una anotación de texto. 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) {
    // ...
}
```

En TypeScript, las interfaces se utilizan para asignar un nombre a los paquetes de propiedades obligatorias y opcionales y a sus tipos. A continuación, puede utilizar el nombre de la interfaz como anotación de tipo. TypeScript se asegurará de que el objeto que utilice como argumento para una función, por ejemplo, tenga las propiedades requeridas de los tipos correctos.

```
interface myFuncProps {
    code: lambda.Code,
    handler?: string
}
```

JavaScript no tiene una función de interfaz, por lo que una vez que haya eliminado las anotaciones de tipo, elimine por completo las declaraciones de la interfaz.

Cuando una función o un método devuelve un tipo de uso general (por ejemplo*`object`*), pero desea tratar ese valor como un tipo secundario más específico para acceder a propiedades o métodos que no forman parte de la interfaz del tipo más general, TypeScript le permite convertir el valor *`as`* seguido de un nombre de tipo o interfaz. JavaScript no lo admite (o no lo necesita), así que basta con quitar `as` y el siguiente identificador. Una sintaxis de conversión menos común es usar el nombre de un tipo entre paréntesis`<LikeThis>`; estas conversiones también deben eliminarse.

Por último, TypeScript es compatible con los modificadores `public` de acceso y `private` para los miembros de las clases. `protected` Todos los miembros de la clase JavaScript son públicos. Simplemente elimine estos modificadores dondequiera que los vea.

Saber cómo identificar y eliminar estas TypeScript funciones contribuye en gran medida a adaptar los TypeScript fragmentos cortos a ellas. JavaScript Sin embargo, puede resultar poco práctico convertir TypeScript ejemplos más largos de esta manera, ya que es más probable que utilicen otras funciones. TypeScript Para estas situaciones, recomendamos Sucrase. Sucrase no se quejará si el código usa una variable indefinida, por ejemplo, como lo haría. `tsc` Si es sintácticamente válido, con pocas excepciones, Sucrase puede traducirlo a. JavaScript Esto lo hace particularmente valioso para convertir fragmentos que tal vez no se puedan ejecutar por sí solos.

¿Migrar a TypeScript

Muchos JavaScript desarrolladores se mudan a ella a TypeScriptmedida que sus proyectos se hacen más grandes y complejos. TypeScript es un superconjunto de JavaScript (todo el JavaScript código es TypeScript código válido, por lo que no es necesario realizar cambios en el código) y también es un lenguaje compatible. AWS CDK Las anotaciones de texto y otras TypeScript funciones son opcionales y puedes añadirlas a tu AWS CDK aplicación a medida que encuentres valor en ellas. TypeScript también te da acceso anticipado a nuevas JavaScript funciones, como el encadenamiento opcional y la fusión de valores nulos, antes de que estén listas y sin necesidad de actualizar Node.js.

TypeScriptSus interfaces «basadas en formas», que definen paquetes de propiedades obligatorias y opcionales (y sus tipos) dentro de un objeto, permiten detectar los errores más comunes al escribir el código y facilitan que el IDE pueda proporcionar consejos sólidos de autocompletado y otros consejos de codificación en tiempo real.

La codificación TypeScript implica un paso adicional: compilar la aplicación con el compilador,. TypeScript `tsc` Para AWS CDK las aplicaciones típicas, la compilación requiere unos segundos como máximo.

La forma más sencilla de migrar una JavaScript AWS CDK aplicación existente TypeScript es crear un nuevo TypeScript proyecto utilizando `cdk init app --language typescript` los archivos fuente (y cualquier otro archivo necesario, como recursos como el código fuente de una AWS Lambda función) y copiarlos al nuevo proyecto. Cambia el nombre de tus JavaScript archivos para que terminen en `.ts` y comiencen a desarrollar en TypeScript.

Trabajando con el AWS CDK en Python

Python es un lenguaje de cliente totalmente compatible AWS Cloud Development Kit (AWS CDK) y se considera estable. Para trabajar con él AWS CDK en Python se utilizan herramientas conocidas, como la implementación estándar de Python (CPython), los entornos virtuales con `virtualenv` y el instalador `pip` de paquetes de Python. Los módulos que componen la biblioteca AWS Construct se distribuyen a través de pypi.org. La versión Python del evento usa AWS CDK identificadores al estilo de Python (por ejemplo, `snake_case` nombres de métodos).

Puedes usar cualquier editor o IDE. Muchos AWS CDK desarrolladores usan Visual Studio Code (o su equivalente de código abierto VSCodium), que tiene un buen soporte para Python a través de una extensión oficial. El editor IDLE incluido con Python será suficiente para empezar. Los módulos de Python AWS CDK tienen sugerencias de tipo, que son útiles para una herramienta de linting o un IDE que admita la validación de tipos.

Temas

Introducción a Python

Para trabajar con ellos AWS CDK, debe tener una AWS cuenta y credenciales y tener instalados Node.js y el AWS CDK kit de herramientas. Consulte Cómo empezar con el AWS CDK.

AWS CDK Las aplicaciones de Python requieren Python 3.6 o posterior. Si aún no lo tienes instalado, descarga una versión compatible para tu sistema operativo en python.org. Si utilizas Linux, es posible que tu sistema tenga una versión compatible o puedes instalarlo mediante el administrador de paquetes de tu distribución (`yum`,`apt`, etc.). Los usuarios de Mac pueden estar interesados en Homebrew, un administrador de paquetes estilo Linux para macOS.

> **Note**
>
> Algunos idiomas de terceros están en desuso: la versión en otros idiomas solo se admite hasta que el proveedor o la comunidad compartan su fecha de caducidad (EOL) y está sujeta a cambios con previo aviso.

También se requieren el instalador del paquete Python y el `virtualenv` administrador del entorno virtual. `pip` Las instalaciones de Windows de las versiones de Python compatibles incluyen estas herramientas. En Linux, `pip` y se `virtualenv` pueden proporcionar como paquetes separados en el administrador de paquetes. Como alternativa, puede instalarlos con los siguientes comandos:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

Si se produce un error de permisos, ejecute los comandos anteriores con la `--user` marca para que los módulos se instalen en su directorio de usuarios o utilícelos `sudo` para obtener los permisos necesarios para instalar los módulos en todo el sistema.

> **Note**
>
> Es común que las distribuciones de Linux usen el nombre ejecutable `python3` para Python 3.x y hagan `python` referencia a una instalación de Python 2.x. Algunas distribuciones tienen un paquete opcional que puedes instalar y que hace que el `python` comando haga referencia a Python 3. De lo contrario, puedes ajustar el comando utilizado para ejecutar tu aplicación editándolo `cdk.json` en el directorio principal del proyecto.

> **Note**
>
> En Windows, es posible que desee invocar Python (ypip) mediante el py ejecutable, el lanzador >Python para Windows. Entre otras cosas, el lanzador le permite especificar fácilmente qué versión instalada de Python desea usar.
> Si python al escribir en la línea de comandos aparece un mensaje sobre la instalación de Python desde la Tienda Windows, incluso después de instalar una versión de Python para Windows, abra el panel de configuración Administrar alias de ejecución de aplicaciones de Windows y desactive las dos entradas del instalador de aplicaciones para Python.

Creación de un proyecto

Para crear un AWS CDK proyecto nuevo, se invoca `cdk init` en un directorio vacío. Utilice la `--language` opción y especifiquepython:

```
mkdir my-project
cd my-project
cdk init app --language python
```

`cdk init`usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debería seguir la forma de un identificador de Python; por ejemplo, no debería empezar por un número ni contener espacios.

Para trabajar con el nuevo proyecto, active su entorno virtual. Esto permite que las dependencias del proyecto se instalen localmente en la carpeta del proyecto, en lugar de hacerlo globalmente.

```
source .venv/bin/activate
```

> **Note**
>
> Es posible que reconozca esto como el comando de Mac/Linux para activar un entorno virtual. Las plantillas de Python incluyen un archivo por lotes, `source.bat`, que permite utilizar el mismo comando en Windows. El comando tradicional de Windows `.venv\Scripts\activate.bat`,, también funciona.
> Si inicializó el AWS CDK proyecto con CDK Toolkit v1.70.0 o una versión anterior, su entorno virtual estará en el directorio en lugar de hacerlo. `.env .venv`

> **Important**
>
> Activa el entorno virtual del proyecto cada vez que empieces a trabajar en él. De lo contrario, no tendrá acceso a los módulos instalados allí y los módulos que instale irán al directorio global de módulos de Python (o generarán un error de permiso).

Tras activar tu entorno virtual por primera vez, instala las dependencias estándar de la aplicación:

```
python -m pip install -r requirements.txt
```

Gestión de los módulos de AWS Construct Library

Usa el instalador de paquetes de Python para instalar y actualizar los módulos de AWS Construct Library para que los usen tus aplicaciones, así como otros paquetes que necesites. pip piptambién instala las dependencias de esos módulos automáticamente. Si su sistema no lo reconoce pip como un comando independiente, invoque pip como un módulo de Python, de la siguiente manera:

```
python -m pip PIP-COMMAND
```

La mayoría de las AWS CDK construcciones están incluidas. `aws-cdk-lib` Los módulos experimentales están en módulos separados llamados así`aws-cdk.SERVICE-NAME.alpha`. El

nombre del servicio incluye un prefijo aws. Si no estás seguro del nombre de un módulo, búscalo en PyPI. Por ejemplo, el siguiente comando instala la biblioteca. AWS CodeStar

```
python -m pip install aws-cdk.aws-codestar-alpha
```

Las estructuras de algunos servicios se encuentran en más de un espacio de nombres. Por ejemplo, además`aws-cdk.aws-route53`, hay tres espacios de nombres adicionales de Amazon Route 53: named `aws-route53-targets`, `aws-route53-patterns`, y. `aws-route53resolver`

> **Note**
>
> La edición de Python de la referencia de la API de CDK también muestra los nombres de los paquetes.

Los nombres utilizados para importar los módulos de AWS Construct Library al código de Python son los siguientes.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

Recomendamos las siguientes prácticas al importar AWS CDK clases y módulos de AWS Construct Library a sus aplicaciones. Seguir estas pautas ayudará a que su código sea coherente con el de otras AWS CDK aplicaciones y a que sea más fácil de entender.

- Por lo general, importe clases individuales del nivel superior. `aws_cdk`

  ```
  from aws_cdk import App, Construct
  ```

- Si necesita muchas clases de`aws_cdk`, puede utilizar un alias de espacio de nombres `cdk` en lugar de importar clases individuales. Evita hacer ambas cosas.

  ```
  import aws_cdk as cdk
  ```

- Por lo general, importe las bibliotecas de AWS Construct utilizando alias de espacios de nombres cortos.

  ```
  import aws_cdk.aws_s3 as s3
  ```

Después de instalar un módulo, actualiza el `requirements.txt` archivo del proyecto, que contiene una lista de las dependencias del proyecto. Es mejor hacerlo manualmente en lugar de usarlo`pip freeze`. `pip freeze`captura las versiones actuales de todos los módulos instalados en su entorno virtual de Python, lo que puede resultar útil al empaquetar un proyecto para ejecutarlo en otro lugar.

Sin embargo, normalmente `requirements.txt` debes enumerar solo las dependencias de nivel superior (módulos de los que depende directamente tu aplicación) y no las dependencias de esas bibliotecas. Esta estrategia simplifica la actualización de las dependencias.

Puede editar `requirements.txt` para permitir las actualizaciones; basta con ~= sustituir el número de versión == anterior por uno que permita actualizar a una versión compatible superior o eliminar por completo el requisito de versión para especificar la última versión disponible del módulo.

Si se ha `requirements.txt` editado adecuadamente para permitir las actualizaciones, ejecute este comando para actualizar los módulos instalados del proyecto en cualquier momento:

```
pip install --upgrade -r requirements.txt
```

Administrar las dependencias en Python

En Python, las dependencias se especifican colocándolas en `requirements.txt` aplicaciones o bibliotecas `setup.py` de construcción. Luego, las dependencias se administran con la herramienta PIP. El PIP se invoca de una de las siguientes maneras:

```
pip command options
python -m pip command options
```

La python -m pip invocación funciona en la mayoría de los sistemas; pip requiere que el ejecutable de PIP esté en la ruta del sistema. Si pip no funciona, intente sustituirlo por. python -m pip

El cdk init --language python comando crea un entorno virtual para su nuevo proyecto. Esto permite que cada proyecto tenga sus propias versiones de las dependencias y también un `requirements.txt` archivo básico. Debe activar este entorno virtual ejecutándolo source .venv/bin/activate cada vez que comience a trabajar con el proyecto.

Aplicaciones CDK

A continuación se muestra un ejemplo de un archivo `requirements.txt`. Como PIP no tiene una función de bloqueo de dependencias, le recomendamos que utilice el operador == para especificar las versiones exactas de todas las dependencias, como se muestra aquí.

```
aws-cdk-lib==2.14.0
aws-cdk.aws-appsync-alpha==2.10.0a0
```

Al instalar un módulo con, pip install no se añade automáticamente a. `requirements.txt` Debe hacerlo usted mismo. Si desea actualizar a una versión posterior de una dependencia, edite su número de versión en`requirements.txt`.

Para instalar o actualizar las dependencias de tu proyecto después de crearlo o editarlo`requirements.txt`, ejecuta lo siguiente:

```
python -m pip install -r requirements.txt
```

> **Tip**
>
> El pip freeze comando muestra las versiones de todas las dependencias instaladas en un formato que se puede escribir en un archivo de texto. Se puede utilizar como un archivo de requisitos con`pip install -r`. Este archivo es práctico para fijar todas las dependencias (incluidas las transitivas) a las versiones exactas con las que ha realizado las pruebas. Para evitar problemas al actualizar los paquetes más adelante, utilice un archivo independiente para ello, como `freeze.txt` (no). `requirements.txt` A continuación, regenéralo cuando actualices las dependencias de tu proyecto.

Bibliotecas de construcción de terceros

En las bibliotecas, las dependencias se especifican en`setup.py`, de modo que las dependencias transitivas se descargan automáticamente cuando una aplicación consume el paquete. De lo contrario, todas las aplicaciones que quieran usar tu paquete deberán copiar tus dependencias en las suyas. `requirements.txt` Aquí `setup.py` se muestra un ejemplo.

```
from setuptools import setup

setup(
  name='my-package',
  version='0.0.1',
  install_requires=[
    'aws-cdk-lib==2.14.0',
  ],
  ...
```

```
)
```

Para trabajar en el paquete para el desarrollo, cree o active un entorno virtual y, a continuación, ejecute el siguiente comando.

```
python -m pip install -e .
```

Aunque PIP instala automáticamente las dependencias transitivas, solo puede haber una copia instalada de cada paquete. Se selecciona la versión que se especifique más arriba en el árbol de dependencias; las aplicaciones siempre tienen la última palabra en qué versión de los paquetes se instalan.

AWS CDK modismos en Python

Conflictos lingüísticos

En Python, `lambda` es una palabra clave del lenguaje, por lo que no se puede utilizar como nombre para el módulo de biblioteca de AWS Lambda construcciones o las funciones de Lambda. La convención de Python para este tipo de conflictos es usar un guión bajo al final, como en el `lambda_` nombre de la variable.

Por convención, se nombra el segundo argumento de los AWS CDK constructos. `id` Al escribir tus propias pilas y construcciones, llamar a un parámetro `id` «sombrea» la función integrada de Python`id()`, que devuelve el identificador único de un objeto. Esta función no se usa con mucha frecuencia, pero si la necesitas en tu construcción, cambia el nombre del argumento, por ejemplo. `construct_id`

Argumentos y propiedades

Todas las clases de AWS Construct Library se instancian mediante tres argumentos: el ámbito en el que se define la construcción (su elemento principal en el árbol de construcciones), un identificador y props, un conjunto de pares clave/valor que la construcción utiliza para configurar los recursos que crea. Otras clases y métodos también utilizan el patrón de «conjunto de atributos» como argumento.

scope e id siempre deben pasarse como argumentos posicionales, no como argumentos de palabras clave, ya que sus nombres cambian si la construcción acepta una propiedad denominada scope o id.

En Python, los props se expresan como argumentos de palabras clave. Si un argumento contiene estructuras de datos anidadas, estas se expresan mediante una clase que toma sus propios

argumentos de palabras clave en la instanciación. El mismo patrón se aplica a otras llamadas a métodos que utilizan un argumento estructurado.

Por ejemplo, en el `add_lifecycle_rule` método de un bucket de Amazon S3, la `transitions` propiedad es una lista de `Transition` instancias.

```
bucket.add_lifecycle_rule(
  transitions=[
    Transition(
      storage_class=StorageClass.GLACIER,
      transition_after=Duration.days(10)
    )
  ]
)
```

Al ampliar una clase o anular un método, es posible que desee aceptar argumentos adicionales para sus propios fines que la clase principal no comprenda. En este caso, debes aceptar los argumentos que no te interese usar en la `**kwargs` expresión idiomática y usar argumentos que solo contengan palabras clave para aceptar los argumentos que te interesen. Cuando llames al constructor principal o al método anulado, pasa solo los argumentos que esperas (normalmente solo). `**kwargs` Si se pasan argumentos que la clase o el método principal no esperan, se produce un error.

```
class MyConstruct(Construct):
    def __init__(self, id, *, MyProperty=42, **kwargs):
        super().__init__(self, id, **kwargs)
        # ...
```

Una futura versión del AWS CDK podría añadir casualmente una nueva propiedad con el nombre que utilizó para su propia propiedad. Esto no provocará ningún problema técnico a los usuarios de su construcción o método (dado que su propiedad no pasa a un nivel superior de la cadena, la clase principal o el método anulado simplemente utilizarán un valor predeterminado), pero puede provocar confusión. Puedes evitar este posible problema asignando un nombre a tus propiedades de forma que pertenezcan claramente a tu construcción. Si hay muchas propiedades nuevas, agrúpalas en una clase con el nombre adecuado y pásala como un único argumento de palabra clave.

Valores faltantes

Se AWS CDK utiliza None para representar valores faltantes o indefinidos. Al trabajar con `**kwargs` ellos, utilice el `get()` método del diccionario para proporcionar un valor predeterminado si no se

proporciona una propiedad. Evite su usokwargs[...], ya que esto aumenta el KeyError número de valores faltantes.

```
encrypted = kwargs.get("encrypted")         # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)  # specify default of False if property is
 missing
```

Es posible que algunos AWS CDK métodos (como tryGetContext() obtener un valor de contexto en tiempo de ejecución) devuelvanNone, lo que deberá comprobar de forma explícita.

Uso de interfaces

Python no tiene una función de interfaz como la tienen otros lenguajes, aunque sí tiene clases base abstractas, que son similares. (Si no estás familiarizado con las interfaces, Wikipedia tiene una buena introducción). TypeScript, el lenguaje en el que AWS CDK se implementa, proporciona interfaces, y las construcciones y otros AWS CDK objetos suelen requerir un objeto que se adhiera a una interfaz en particular, en lugar de heredarlo de una clase en particular. Por lo tanto, AWS CDK proporciona su propia función de interfaz como parte de la capa JSII.

Para indicar que una clase implementa una interfaz en particular, puedes usar el @jsii.implements decorador:

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Escriba los escollos

Python usa la escritura dinámica, donde todas las variables pueden hacer referencia a un valor de cualquier tipo. Los parámetros y los valores devueltos se pueden anotar con tipos, pero se trata de «sugerencias» y no se aplican. Esto significa que en Python, es fácil pasar el tipo de valor incorrecto a una AWS CDK construcción. En lugar de recibir un error de tipo durante la compilación, como ocurriría en un lenguaje de tipado estático, es posible que se produzca un error de tiempo de ejecución cuando la capa JSII (que traduce entre Python y su TypeScript núcleo) no pueda procesar el AWS CDK tipo inesperado.

Según nuestra experiencia, los errores tipográficos que cometen los programadores de Python suelen clasificarse en estas categorías.

- Pasar un único valor donde una construcción espera un contenedor (lista o diccionario de Python) o viceversa.
- Pasar un valor de un tipo asociado a una construcción de capa 1 (`CfnXxxxxx`) a una construcción de L2 o L3, o viceversa.

Los módulos de AWS CDK Python incluyen anotaciones de tipos, por lo que puede usar herramientas que los admitan para ayudar con los tipos. Si, por ejemplo, no utilizas un IDE compatible con estas funciones PyCharm, puedes utilizar el validador de MyPytipos como parte del proceso de creación. También hay comprobadores de tipos en tiempo de ejecución que pueden mejorar los mensajes de error relacionados con los tipos.

Sintetizar y desplegar

Las pilas definidas en tu AWS CDK aplicación se pueden sintetizar e implementar de forma individual o conjunta mediante los siguientes comandos. Por lo general, debes estar en el directorio principal de tu proyecto cuando los emitas.

- `cdk synth`: sintetiza una AWS CloudFormation plantilla a partir de una o más de las pilas de tu AWS CDK aplicación.
- `cdk deploy`: despliega los recursos definidos por una o más de las pilas de tu aplicación en. AWS CDK AWS

Puedes especificar los nombres de varias pilas que se van a sintetizar o implementar en un solo comando. Si tu aplicación define solo una pila, no es necesario que la especifiques.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

También puedes usar los comodines * (cualquier número de caracteres) y? (cualquier carácter individual) para identificar las pilas por patrón. Cuando utilice caracteres comodín, escriba el patrón entre comillas. De lo contrario, es posible que el shell intente ampliarlo hasta incluir los nombres de los archivos del directorio actual antes de pasarlos al AWS CDK kit de herramientas.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
```

```
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> No necesita sintetizar las pilas de forma explícita antes de desplegarlas; `cdk deploy` realiza este paso para asegurarse de que se implementa el código más reciente.

Para obtener la documentación completa del `cdk` comando, consulte. the section called “AWS CDK Kit de herramientas”

Trabajando con el AWS CDK en Java

Java es un lenguaje de cliente totalmente compatible AWS CDK y se considera estable. Puede desarrollar AWS CDK aplicaciones en Java con herramientas conocidas, como el JDK (de Oracle o una distribución de OpenJDK como Amazon Corretto) y Apache Maven.

AWS CDK Es compatible con Java 8 y versiones posteriores. Sin embargo, le recomendamos que utilice la última versión posible, ya que las versiones posteriores del lenguaje incluyen mejoras que son especialmente prácticas para el desarrollo de AWS CDK aplicaciones. Por ejemplo, Java 9 introduce el `Map.of()` método (una forma práctica de declarar mapas hash que se escribirían como objetos literales). TypeScript Java 10 introduce la inferencia de tipos locales mediante la palabra clave. `var`

> **Note**
>
> La mayoría de los ejemplos de código de esta guía para desarrolladores funcionan con Java 8. Algunos ejemplos utilizan`Map.of()`; estos ejemplos incluyen comentarios en los que se indica que requieren Java 9.

Puedes usar cualquier editor de texto o un IDE de Java que pueda leer proyectos de Maven para trabajar en tus AWS CDK aplicaciones. En esta guía ofrecemos consejos sobre Eclipse, pero IntelliJ IDEA NetBeans y otros IDE pueden importar proyectos de Maven y pueden usarse para AWS CDK desarrollar aplicaciones en Java.

Es posible escribir AWS CDK aplicaciones en lenguajes alojados en JVM distintos de Java (por ejemplo, Kotlin, Groovy, Clojure o Scala), pero es posible que la experiencia no sea particularmente idiomática y no podamos ofrecer ningún soporte para estos lenguajes.

Temas

Introducción a Java

Para trabajar con ellos, debe tener una cuenta AWS CDK y credenciales y tener instalados Node.js y el kit de herramientas AWS . AWS CDK Consulte Cómo empezar con el AWS CDK.

AWS CDK Las aplicaciones Java requieren Java 8 (v1.8) o posterior. Recomendamos Amazon Corretto, pero puede usar cualquier distribución de OpenJDK o el JDK de Oracle. También necesitará Apache Maven 3.5 o posterior. También puede utilizar herramientas como Gradle, pero los esqueletos de aplicaciones generados por el kit de AWS CDK herramientas son proyectos de Maven.

> **Note**
>
> Degradación de idiomas de terceros: la versión lingüística solo se admite hasta el final de su vida útil (EOL), compartida por el proveedor o la comunidad, y está sujeta a cambios con previo aviso.

Creación de un proyecto

Para crear un AWS CDK proyecto nuevo, se invoca `cdk init` en un directorio vacío. Utilice la `--language` opción y especifique`java`:

```
mkdir my-project
```

```
cd my-project
cdk init app --language java
```

`cdk init`usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debe seguir la forma de un identificador de Java; por ejemplo, no debe empezar por un número ni contener espacios.

El proyecto resultante incluye una referencia al paquete `software.amazon.awscdk` Maven. Maven lo instala automáticamente y sus dependencias.

Si está utilizando un IDE, ahora puede abrir o importar el proyecto. En Eclipse, por ejemplo, elija Archivo > Importar > Maven > Proyectos Maven existentes. Asegúrese de que los ajustes del proyecto estén configurados para utilizar Java 8 (1.8).

Gestión de los módulos de AWS Construct Library

Use Maven para instalar los paquetes de AWS Construct Library, que están en el grupo`software.amazon.awscdk`. La mayoría de las construcciones se encuentran en el artefacto`aws-cdk-lib`, que se añade a los nuevos proyectos de Java de forma predeterminada. Los módulos para los servicios cuyo soporte de CDK de nivel superior aún se está desarrollando se encuentran en paquetes «experimentales» separados, denominados con una versión abreviada (no o con el prefijo de AWS Amazon) del nombre de su servicio. Busca en el repositorio central de Maven los nombres de todas las bibliotecas AWS CDK y AWS de Construct Module.

> **Note**
>
> La edición Java de la referencia de la API de CDK también muestra los nombres de los paquetes.

La compatibilidad con AWS Construct Library de algunos servicios está en más de un espacio de nombres. Por ejemplo, Amazon Route 53 tiene su funcionalidad dividida en `software.amazon.awscdk.route53`route53-patterns,`route53resolver`, yroute53-targets.

El AWS CDK paquete principal se importa en código Java como`software.amazon.awscdk`. Los módulos de los distintos servicios de la biblioteca AWS Construct se encuentran bajo el nombre de su paquete de Maven `software.amazon.awscdk.services` y reciben un

nombre similar al de su paquete. Por ejemplo, el espacio de nombres del módulo Amazon S3 es. `software.amazon.awscdk.services.s3`

Le recomendamos escribir una `import` sentencia Java independiente para cada clase de AWS Construct Library que utilice en cada uno de sus archivos fuente de Java y evitar la importación de caracteres comodín. Siempre puedes usar el nombre completo de un tipo (incluido su espacio de nombres) sin ninguna instrucción. `import`

Si tu aplicación depende de un paquete experimental, edita la de tu proyecto `pom.xml` y añade un nuevo `<dependency>` elemento al contenedor. `<dependencies>` Por ejemplo, el siguiente `<dependency>` elemento especifica el módulo de biblioteca de construcciones CodeStar experimentales:

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>codestar-alpha</artifactId>
    <version>2.0.0-alpha.10</version>
</dependency>
```

> **Tip**
>
> Si usa un IDE de Java, probablemente tenga funciones para administrar las dependencias de Maven. Sin embargo, te recomendamos editarlo `pom.xml` directamente, a menos que estés absolutamente seguro de que la funcionalidad del IDE coincide con lo que harías a mano.

Administrar las dependencias en Java

En Java, las dependencias se especifican `pom.xml` e instalan mediante Maven. El `<dependencies>` contenedor incluye un `<dependency>` elemento para cada paquete. La siguiente es una sección `pom.xml` de una aplicación Java de CDK típica.

```
<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**
>
> Muchos IDE de Java cuentan con soporte para Maven y `pom.xml` editores visuales integrados, lo que puede resultarle práctico para gestionar las dependencias.

Maven no admite el bloqueo de dependencias. Aunque es posible especificar rangos de versiones`pom.xml`, te recomendamos que utilices siempre versiones exactas para que tus compilaciones se puedan repetir.

Maven instala automáticamente las dependencias transitivas, pero solo puede haber una copia instalada de cada paquete. Se selecciona la versión que se especifique más arriba en el árbol POM; las aplicaciones siempre tienen la última palabra en qué versión de los paquetes se instalan.

Maven instala o actualiza automáticamente tus dependencias cada vez que compilas (mvn compile) o empaquetas () tu proyectomvn package. El kit de herramientas CDK lo hace automáticamente cada vez que lo ejecutas, por lo que, por lo general, no es necesario invocar Maven manualmente.

AWS CDK modismos en Java

Utilería

Todas las clases de AWS Construct Library se instancian mediante tres argumentos: el ámbito en el que se define la construcción (su elemento principal en el árbol de construcciones), un identificador y props, un conjunto de pares clave/valor que la construcción utiliza para configurar los recursos que crea. Otras clases y métodos también utilizan el patrón de «conjunto de atributos» como argumento.

En Java, los accesorios se expresan mediante el patrón Builder. Cada tipo de construcción tiene un tipo de accesorios correspondiente; por ejemplo, la `Bucket` construcción (que representa un bucket de Amazon S3) toma como accesorios una instancia de. `BucketProps`

La `BucketProps` clase (como todas las clases de props de AWS Construct Library) tiene una clase interna llamada. `Builder` El `BucketProps.Builder` tipo ofrece métodos para establecer las

distintas propiedades de una `BucketProps` instancia. Cada método devuelve la `Builder` instancia, por lo que las llamadas a los métodos se pueden encadenar para establecer varias propiedades. Al final de la cadena, se llama `build()` para producir realmente el `BucketProps` objeto.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build());
```

Los constructos y otras clases que utilizan como argumento final un objeto similar a un accesorio ofrecen un atajo. La clase tiene una propia que crea una instancia `Builder` de ella y de su objeto props en un solo paso. De esta forma, no necesitas crear instancias explícitas (por ejemplo) tanto como un`Bucket`, `BucketProps` y no necesitas importar el tipo props.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build();
```

Al derivar tu propia construcción a partir de una construcción existente, es posible que desees aceptar propiedades adicionales. Le recomendamos que siga estos patrones de construcción. Sin embargo, esto no es tan simple como subclasificar una clase de construcción. Debes proporcionar tú mismo las partes móviles de las dos nuevas `Builder` clases. Es posible que prefiera que su construcción simplemente acepte uno o más argumentos adicionales. Debe proporcionar constructores adicionales cuando un argumento sea opcional.

Estructuras genéricas

En algunas API, AWS CDK utiliza JavaScript matrices u objetos sin tipo como entrada a un método. (Consulte, por ejemplo, el método AWS CodeBuild de `BuildSpec.fromObject()`.) En Java, estos objetos se representan como`java.util.Map<String, Object>`. En los casos en que los valores son todos cadenas, puede utilizar`Map<String, String>`.

Java no proporciona una forma de escribir literales para dichos contenedores como lo hacen otros lenguajes. En Java 9 y versiones posteriores, se pueden utilizar `java.util.Map.of()`para definir cómodamente mapas de hasta diez entradas en línea con una de estas llamadas.

```
java.util.Map.of(
    "base-directory", "dist",
```

```
    "files", "LambdaStack.template.json"
)
```

Para crear mapas con más de diez entradas, utilice `java.util.Map.ofEntries()`.

Si utiliza Java 8, puede proporcionar sus propios métodos similares a estos.

JavaScript las matrices se representan como `List<Object>` o `List<String>` en Java. El método `java.util.Arrays.asList` es conveniente para definir `List` s. cortas.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

Valores faltantes

En Java, los valores que faltan en AWS CDK objetos como los accesorios se representan mediante`null`. Debe probar explícitamente cualquier valor que pueda existir `null` para asegurarse de que contiene un valor antes de hacer cualquier cosa con él. Java no tiene un «azúcar sintáctico» que ayude a gestionar los valores nulos, como ocurre en otros lenguajes. Puede que el uso ObjectUtil de defaultIfNullApache le firstNonNullresulte útil en algunas situaciones. Como alternativa, escribe tus propios métodos auxiliares estáticos para facilitar el manejo de valores potencialmente nulos y hacer que tu código sea más legible.

Construir, sintetizar e implementar

Compila AWS CDK automáticamente la aplicación antes de ejecutarla. Sin embargo, puede resultar útil compilar la aplicación manualmente para comprobar si hay errores y realizar pruebas. Puedes hacerlo en tu IDE (por ejemplo, presionando Control-B en Eclipse) o ejecutándolo `mvn compile` en una línea de comandos desde el directorio raíz de tu proyecto.

Ejecuta cualquier prueba que hayas escrito desde una línea `mvn test` de comandos.

Las pilas definidas en tu AWS CDK aplicación se pueden sintetizar e implementar de forma individual o conjunta mediante los siguientes comandos. Por lo general, debes estar en el directorio principal de tu proyecto cuando los emitas.

- `cdk synth`: sintetiza una AWS CloudFormation plantilla a partir de una o más de las pilas de tu AWS CDK aplicación.
- `cdk deploy`: despliega los recursos definidos por una o más de las pilas de tu aplicación en. AWS CDK AWS

Puedes especificar los nombres de varias pilas que se van a sintetizar o implementar en un solo comando. Si tu aplicación define solo una pila, no es necesario que la especifiques.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

También puedes usar los comodines * (cualquier número de caracteres) y? (cualquier carácter individual) para identificar las pilas por patrón. Cuando utilice caracteres comodín, escriba el patrón entre comillas. De lo contrario, es posible que el shell intente ampliarlo hasta incluir los nombres de los archivos del directorio actual antes de pasarlos al AWS CDK kit de herramientas.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> No necesita sintetizar las pilas de forma explícita antes de desplegarlas; `cdk deploy` realiza este paso para asegurarse de que se implementa el código más reciente.

Para obtener la documentación completa del `cdk` comando, consulte. the section called “AWS CDK Kit de herramientas”

Trabajando con el AWS CDK en C#

.NET es un lenguaje de cliente totalmente compatible AWS CDK y se considera estable. C# es el lenguaje principal de.NET para el que ofrecemos ejemplos y soporte. Puede escribir AWS CDK aplicaciones en otros lenguajes.NET, como Visual Basic o F#, pero AWS ofrece una compatibilidad limitada para el uso de estos lenguajes con el CDK.

Puede desarrollar AWS CDK aplicaciones en C# con herramientas conocidas, como Visual Studio, Visual Studio Code, el `dotnet` comando y el NuGet administrador de paquetes. Los módulos que componen la biblioteca AWS Construct se distribuyen a través de nuget.org.

Sugerimos usar Visual Studio 2019 (cualquier edición) en Windows para desarrollar AWS CDK aplicaciones en C#.

Temas

Introducción a C#

Para trabajar con él AWS CDK, debe tener una AWS cuenta y credenciales y tener instalados Node.js y el AWS CDK kit de herramientas. Consulte Cómo empezar con el AWS CDK.

AWS CDK Las aplicaciones de C# requieren .NET Core v3.1 o una versión posterior, disponible aquí.

La cadena de herramientas de.NET incluye `dotnet` una herramienta de línea de comandos para crear y ejecutar aplicaciones.NET y administrar paquetes. NuGet Aunque trabaje principalmente en Visual Studio, este comando puede resultar útil para operaciones por lotes y para instalar paquetes de AWS Construct Library.

Creación de un proyecto

Para crear un AWS CDK proyecto nuevo, se invoca `cdk init` en un directorio vacío. Utilice la `--language` opción y especifique`csharp`:

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init`usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debe seguir la forma de un identificador de C#; por ejemplo, no debe empezar por un número ni contener espacios.

El proyecto resultante incluye una referencia al `Amazon.CDK.Lib` NuGet paquete. Tanto él como sus dependencias se instalan automáticamente mediante NuGet.

Gestión de los módulos de AWS Construct Library

El ecosistema.NET usa el administrador de NuGet paquetes. El paquete CDK principal, que contiene las clases principales y todas las construcciones de servicio estables, es. `Amazon.CDK.Lib` Los

módulos experimentales, en los que se está desarrollando una nueva funcionalidad, reciben un nombre similar`Amazon.CDK.AWS.SERVICE-NAME.Alpha`, donde el nombre del servicio es un nombre abreviado sin el prefijo AWS o de Amazon. Por ejemplo, el nombre NuGet del paquete del AWS IoT módulo es`Amazon.CDK.AWS.IoT.Alpha`. Si no encuentra el paquete que busca, busque en NuGet.org.

> **Note**
>
> La edición.NET de la referencia de la API de CDK también muestra los nombres de los paquetes.

El soporte de AWS Construct Library de algunos servicios se encuentra en más de un módulo. Por ejemplo, AWS IoT tiene un segundo módulo llamado`Amazon.CDK.AWS.IoT.Actions.Alpha`.

El AWS CDK módulo principal, que necesitarás en la mayoría de AWS CDK las aplicaciones, se importa en código C# como`Amazon.CDK`. Los módulos de los distintos servicios de la biblioteca AWS Construct se encuentran debajo`Amazon.CDK.AWS`. Por ejemplo, el espacio de nombres del módulo Amazon S3 es. `Amazon.CDK.AWS.S3`

Recomendamos escribir `using` directivas de C# para las construcciones principales de CDK y para cada AWS servicio que utilice en cada uno de sus archivos fuente de C#. Puede que te resulte práctico usar un alias para un espacio de nombres o un tipo para resolver conflictos de nombres. Siempre puedes usar el nombre completo de un tipo (incluido su espacio de nombres) sin una instrucción. `using`

Administrar las dependencias en C#

En las AWS CDK aplicaciones de C#, las dependencias se administran mediante. NuGet NuGet tiene cuatro interfaces estándar, en su mayoría equivalentes. Utilice la que mejor se adapte a sus necesidades y estilo de trabajo. También puedes usar herramientas compatibles, como Paket, MyGeto incluso editar el `.csproj` archivo directamente.

NuGet no permite especificar rangos de versiones para las dependencias. Cada dependencia está anclada a una versión específica.

Tras actualizar las dependencias, Visual Studio utilizará Visual Studio NuGet para recuperar las versiones especificadas de cada paquete la próxima vez que compile. Si no usa Visual Studio, use el dotnet restore comando para actualizar las dependencias.

Editar el archivo del proyecto directamente

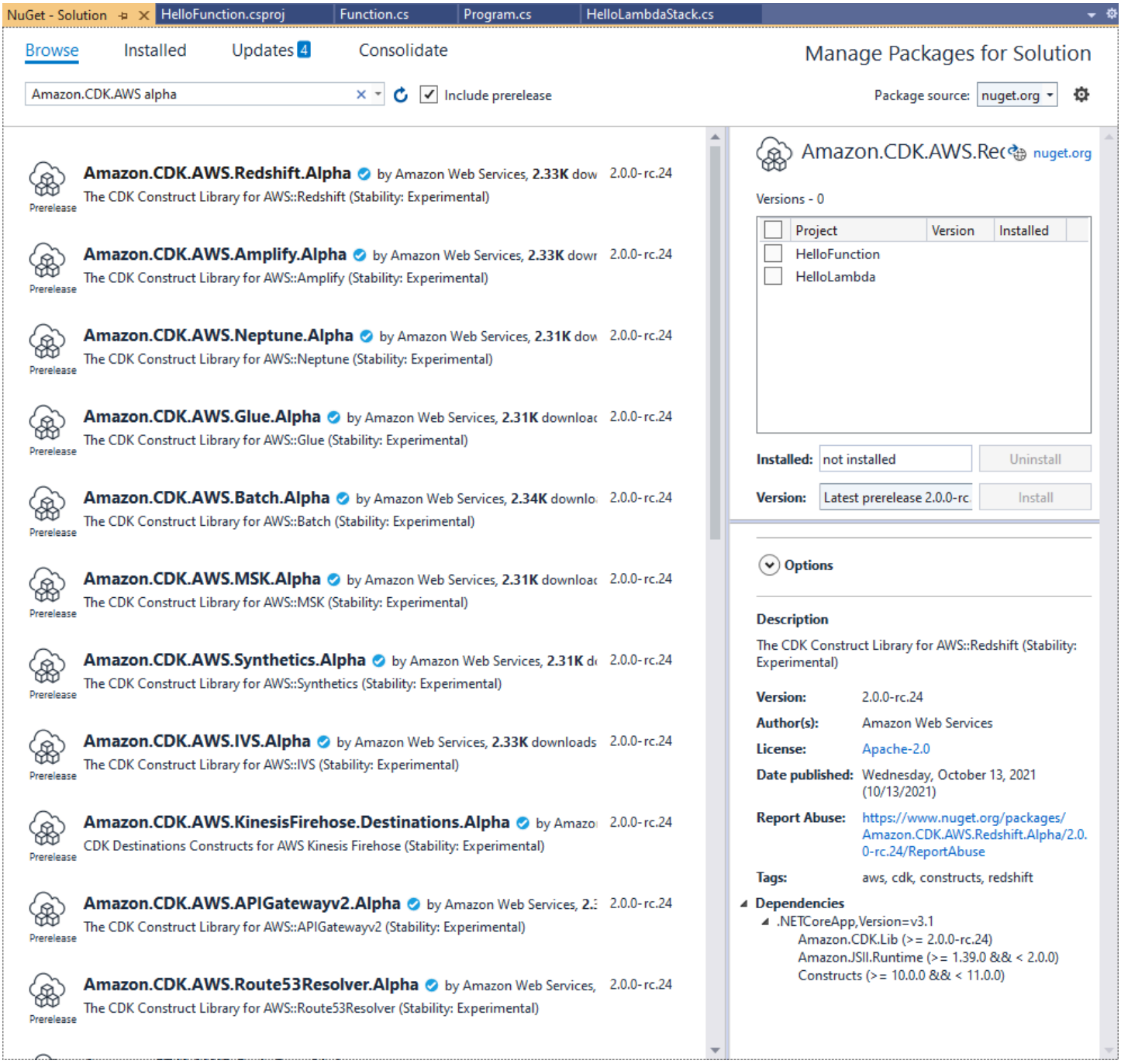
El `.csproj` archivo del proyecto contiene un `<ItemGroup>` contenedor en el que se enumeran las dependencias como `<PackageReference` elementos.

```
<ItemGroup>
    <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
    <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

La GUI de Visual Studio NuGet

Se puede acceder a NuGet las herramientas de Visual Studio desde Herramientas > Administrador de NuGet paquetes > Administrar NuGet paquetes para la solución. Utilice la pestaña Examinar para buscar los paquetes de AWS Construct Library que desee instalar. Puede elegir la versión que desee, incluidas las versiones preliminares de sus módulos, y añadirlas a cualquiera de los proyectos abiertos.

> **Note**
>
> Todos los módulos de AWS Construct Library que se consideran «experimentales» (consultethe section called “Control de versiones”) están marcados como versión preliminar NuGet y tienen un sufijo de nombre. `alpha`



Consulte la página de actualizaciones para instalar nuevas versiones de sus paquetes.

La NuGet consola

La NuGet consola es una interfaz PowerShell basada NuGet que funciona en el contexto de un proyecto de Visual Studio. Puede abrirlo en Visual Studio seleccionando Herramientas > Gestor de NuGet paquetes > Consola de Package Manager. Para obtener más información sobre el uso

de esta herramienta, consulte Instalar y administrar paquetes con la consola Package Manager en Visual Studio.

El **`dotnet`** comando

El `dotnet` comando es la principal herramienta de línea de comandos para trabajar con proyectos de Visual Studio en C#. Puede invocarlo desde cualquier línea de comandos de Windows. Entre sus múltiples capacidades, `dotnet` puede añadir NuGet dependencias a un proyecto de Visual Studio.

Suponiendo que se encuentra en el mismo directorio que el archivo de proyecto (`.csproj`) de Visual Studio, ejecute un comando como el siguiente para instalar un paquete. Como la biblioteca principal de CDK se incluye al crear un proyecto, solo necesita instalar de forma explícita los módulos experimentales. Los módulos experimentales requieren que especifique un número de versión explícito.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Puede ejecutar el comando desde otro directorio. Para ello, incluya la ruta al archivo del proyecto o al directorio que lo contiene después de la `add` palabra clave. En el siguiente ejemplo se supone que se encuentra en el directorio principal del AWS CDK proyecto.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Para instalar una versión específica de un paquete, incluye la `-v` marca y la versión deseada.

Para actualizar un paquete, ejecute el mismo `dotnet add` comando que utilizó para instalarlo. En el caso de los módulos experimentales, de nuevo, debe especificar un número de versión explícito.

Para obtener más información sobre la administración de paquetes mediante el `dotnet` comando, consulte Instalación y administración de paquetes mediante la CLI de dotnet.

El comando **`nuget`**

La herramienta de línea de `nuget` comandos puede instalar y actualizar NuGet paquetes. Sin embargo, requiere que el proyecto de Visual Studio se configure de forma diferente a como `cdk init` se configuran los proyectos. (Detalles técnicos: `nuget` funciona con `Packages.config` proyectos, mientras que `cdk init` crea un `PackageReference` proyecto de estilo más nuevo).

No recomendamos el uso de la `nuget` herramienta con AWS CDK proyectos creados por. `cdk init` Si está utilizando otro tipo de proyecto y quiere usarlo`nuget`, consulte la referencia de NuGet CLI.

AWS CDK modismos en C#

Accesorios

Todas las clases de AWS Construct Library se instancian mediante tres argumentos: el ámbito en el que se define la construcción (su elemento principal en el árbol de construcciones), un identificador y props, un conjunto de pares clave/valor que la construcción utiliza para configurar los recursos que crea. Otras clases y métodos también utilizan el patrón de «conjunto de atributos» como argumento.

En C#, los accesorios se expresan mediante un tipo de accesorios. Al estilo idiomático de C#, podemos usar un inicializador de objetos para establecer las distintas propiedades. Aquí estamos creando un bucket de Amazon S3 con la `Bucket` construcción; su tipo de accesorio correspondiente es`BucketProps`.

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
    Versioned = true
});
```

> **Tip**
>
> Añada el paquete `Amazon.JSII.Analyzers` a su proyecto para obtener los valores necesarios que comprueben las definiciones de sus accesorios en Visual Studio.

Al ampliar una clase o anular un método, es posible que desee aceptar accesorios adicionales para sus propios fines que la clase principal no comprenda. Para ello, subclasifique el tipo de accesorio apropiado y añada los nuevos atributos.

```
// extend BucketProps for use with MimeBucket
class MimeBucketProps : BucketProps {
    public string MimeType { get; set; }
}

// hypothetical bucket that enforces MIME type of objects inside it
class MimeBucket : Bucket {
```

```
     public MimeBucket( readonly Construct scope, readonly string id, readonly
 MimeBucketProps props=null) : base(scope, id, props) {
         // ...
     }
}

// instantiate our MimeBucket class
var bucket = new MimeBucket(this, "MyBucket", new MimeBucketProps {
    Versioned = true,
    MimeType = "image/jpeg"
});
```

Al llamar al inicializador o al método anulado de la clase principal, normalmente puedes pasar los accesorios que has recibido. El nuevo tipo es compatible con su principal y los accesorios adicionales que agregues se ignoran.

Una futura versión del AWS CDK podría añadir casualmente una nueva propiedad con el nombre que utilizó para su propia propiedad. Esto no provocará ningún problema técnico al utilizar la construcción o el método (dado que la propiedad no pasa a un nivel superior de la cadena, la clase principal o el método anulado se limitará a utilizar un valor predeterminado), pero podría causar confusión a los usuarios de la construcción. Puedes evitar este posible problema asignando un nombre a tus propiedades de forma que pertenezcan claramente a tu construcción. Si hay muchas propiedades nuevas, agrúpalas en una clase con el nombre adecuado y pásalas como una sola propiedad.

Estructuras genéricas

En algunas API, AWS CDK utiliza JavaScript matrices u objetos sin tipo como entrada a un método. (Consulte, por ejemplo, el método AWS CodeBuild de `BuildSpec.fromObject()`.) En C#, estos objetos se representan como`System.Collections.Generic.Dictionary<String, Object>`. En los casos en que los valores son todos cadenas, puede utilizar`Dictionary<String, String>`. JavaScript las matrices se representan como `object[]` tipos de `string[]` matrices en C#.

> **Tip**
>
> Puede definir alias cortos para facilitar el trabajo con estos tipos de diccionarios específicos.
>
> ```
> using StringDict = System.Collections.Generic.Dictionary<string, string>;
> ```

```
using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
```

Valores faltantes

En C#, los valores que faltan en AWS CDK objetos como los accesorios se representan mediante. `null` El operador de acceso a miembros con condiciones nulas `?.` y el operador coalescente nulo son prácticos para trabajar con estos valores. `??`

```
// mimeType is null if props is null or if props.MimeType is null
string mimeType = props?.MimeType;

// mimeType defaults to text/plain. either props or props.MimeType can be null
string MimeType = props?.MimeType ?? "text/plain";
```

Construir, sintetizar e implementar

Compila AWS CDK automáticamente la aplicación antes de ejecutarla. Sin embargo, puede resultar útil compilar la aplicación manualmente para comprobar si hay errores y realizar pruebas. Para ello, presione F6 en Visual Studio o ejecute `dotnet build src` desde la línea de comandos el directorio del proyecto que contiene el archivo de la solución de Visual Studio (`.sln`). `src`

Las pilas definidas en la AWS CDK aplicación se pueden sintetizar e implementar de forma individual o conjunta mediante los siguientes comandos. Por lo general, debes estar en el directorio principal de tu proyecto cuando los emitas.

- `cdk synth`: Sintetiza una AWS CloudFormation plantilla a partir de una o más de las pilas de tu AWS CDK aplicación.
- `cdk deploy`: despliega los recursos definidos por una o más de las pilas de tu aplicación en. AWS CDK AWS

Puedes especificar los nombres de varias pilas que se van a sintetizar o implementar en un solo comando. Si tu aplicación define solo una pila, no es necesario que la especifiques.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

También puedes usar los caracteres comodín * (cualquier número de caracteres) y? (cualquier carácter individual) para identificar las pilas por patrón. Cuando utilice caracteres comodín, escriba el patrón entre comillas. De lo contrario, es posible que el shell intente ampliarlo hasta incluir los nombres de los archivos del directorio actual antes de pasarlos al AWS CDK kit de herramientas.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> No necesita sintetizar las pilas de forma explícita antes de desplegarlas; `cdk deploy` realiza este paso para asegurarse de que se implementa el código más reciente.

Para obtener la documentación completa del `cdk` comando, consulte. the section called “AWS CDK Kit de herramientas”

Trabajando con el AWS CDK in Go

Go es un lenguaje de cliente totalmente compatible AWS Cloud Development Kit (AWS CDK) y se considera estable. Para trabajar con el AWS CDK in Go se utilizan herramientas conocidas. La versión Go AWS CDK incluso usa identificadores tipo Go.

A diferencia de los otros lenguajes compatibles con el CDK, Go no es un lenguaje de programación tradicional orientado a objetos. Go utiliza la composición, mientras que otros lenguajes suelen aprovechar la herencia. Hemos intentado emplear enfoques idiomáticos de Go en la medida de lo posible, pero hay aspectos en los que la CDK puede diferir.

En este tema se proporciona orientación para trabajar con el AWS CDK in Go. Consulte la publicación del blog sobre el anuncio para ver un tutorial de un proyecto sencillo de Go para. AWS CDK

Temas

- Introducción a Go
- Creación de un proyecto
- Gestión de los módulos de AWS Construct Library
- Administrar las dependencias en Go

- AWS CDK modismos en Go
- Construir, sintetizar e implementar

Introducción a Go

Para trabajar con el AWS CDK, debe tener una AWS cuenta y credenciales y haber instalado Node.js y el AWS CDK kit de herramientas. Consulte Cómo empezar con el AWS CDK.

Los enlaces Go para el AWS CDK uso de la cadena de herramientas Go estándar, versión 1.18 o posterior. Puedes usar el editor que prefieras.

> **Note**
>
> El uso de idiomas de terceros está en desuso: la versión en otros idiomas solo se admite hasta que el proveedor o la comunidad compartan su fecha de caducidad (EOL), y está sujeta a cambios con previo aviso.

Creación de un proyecto

Para crear un AWS CDK proyecto nuevo, se invoca `cdk init` en un directorio vacío. Utilice la `--language` opción y especifiquego:

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init`usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debería seguir la forma de un identificador de Go; por ejemplo, no debería empezar por un número ni contener espacios.

El proyecto resultante incluye una referencia al módulo principal de AWS CDK Go`github.com/aws/aws-cdk-go/awscdk/v2`, en`go.mod`. Problema go get para instalar este y otros módulos necesarios.

Gestión de los módulos de AWS Construct Library

En la mayoría de la AWS CDK documentación y los ejemplos, la palabra «módulo» suele usarse para referirse a los módulos de AWS Construct Library, uno o más por AWS servicio, lo que difiere del uso idiomático del término en Go. La biblioteca Construct de CDK se proporciona en un módulo Go junto con los módulos individuales de la biblioteca Construct, que admiten los distintos AWS servicios y que se proporcionan como paquetes Go dentro de ese módulo.

La compatibilidad con AWS Construct Library de algunos servicios se encuentra en más de un módulo de Construct Library (paquete Go). Por ejemplo, Amazon Route 53 tiene tres módulos Construct Library además `awsroute53` del paquete principal `awsroute53patternsawsroute53resolver`, denominados y`awsroute53targets`.

El AWS CDK paquete principal, que necesitarás en la mayoría de AWS CDK las aplicaciones, se importa en código Go como`github.com/aws/aws-cdk-go/awscdk/v2`. Los paquetes de los distintos servicios de la biblioteca AWS Construct se encuentran en`github.com/aws/aws-cdk-go/awscdk/v2`. Por ejemplo, el espacio de nombres del módulo Amazon S3 es. `github.com/aws/aws-cdk-go/awscdk/v2/awss3`

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

Una vez que haya importado los módulos de Construct Library (paquetes Go) para los servicios que desee usar en su aplicación, podrá acceder a las construcciones de ese módulo utilizando, por ejemplo,. `awss3.Bucket`

Administrar las dependencias en Go

En Go, las versiones de las dependencias se definen en. `go.mod` El valor predeterminado `go.mod` es similar al que se muestra aquí.

```
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
)
```

Los nombres de los paquetes (módulos, en el lenguaje de Go) se especifican mediante una URL con el número de versión requerido adjunto. El sistema de módulos de Go no admite rangos de versiones.

Ejecute el go get comando para instalar todos los módulos necesarios y actualizarlos`go.mod`. Para ver una lista de las actualizaciones disponibles para sus dependencias, ejecutego list -m -u all.

AWS CDK modismos en Go

Nombres de campos y métodos

Los nombres de campos y métodos utilizan camel casey (`likeThis`) TypeScript, el idioma de origen del CDK. En Go, estos siguen las convenciones de Go, al igual que Pascal-cased (). `LikeThis`

Limpieza

En tu `main` método, úsalo `defer jsii.Close()` para asegurarte de que la aplicación CDK se limpie sola.

Valores faltantes y conversión de punteros

En Go, los valores que faltan en AWS CDK objetos como los paquetes de propiedades se representan mediante`nil`. Go no tiene tipos que acepten valores nulos; el único tipo que puede contener `nil` es un puntero. Por lo tanto, para permitir que los valores sean opcionales, todas las propiedades, argumentos y valores de retorno de CDK son punteros, incluso en el caso de los tipos primitivos. Esto se aplica tanto a los valores obligatorios como a los opcionales, por lo que si un valor obligatorio pasa a ser opcional más adelante, no será necesario realizar ningún cambio radical en el tipo.

Al pasar valores o expresiones literales, utilice las siguientes funciones auxiliares para crear punteros hacia los valores.

- `jsii.String`
- `jsii.Number`
- `jsii.Bool`
- `jsii.Time`

Para mantener la coherencia, le recomendamos que utilice los punteros de forma similar cuando defina sus propias construcciones, aunque pueda parecer más conveniente, por ejemplo, recibir la construcción como una cadena en lugar de `id` como un puntero a una cadena.

Cuando se trate de AWS CDK valores opcionales, incluidos valores primitivos y tipos complejos, deberías probar los punteros de forma explícita para asegurarte de que no lo están `nil` antes de hacer cualquier cosa con ellos. Go no tiene un «azúcar sintáctico» que ayude a gestionar los valores vacíos o faltantes, como ocurre en otros lenguajes. Sin embargo, se garantiza la existencia de los valores obligatorios en los paquetes de propiedades y estructuras similares (de lo contrario, la construcción no funciona), por lo que no es necesario `nil` comprobar estos valores.

Construcciones y accesorios

Las construcciones, que representan uno o más AWS recursos y sus atributos asociados, se representan en Go como interfaces. Por ejemplo, `awss3.Bucket` es una interfaz. Cada construcción tiene una función de fábrica, por ejemplo`awss3.NewBucket`, para devolver una estructura que implementa la interfaz correspondiente.

Todas las funciones de fábrica utilizan tres argumentos: el argumento `scope` en el que se define la construcción (su elemento principal en el árbol de construcciones)`id`, un conjunto de pares clave/valor y `props` un conjunto de pares clave/valor que la construcción utiliza para configurar los recursos que crea. El patrón del «conjunto de atributos» también se utiliza en otras partes del. AWS CDK

En Go, los accesorios se representan mediante un tipo de estructura específico para cada construcción. Por ejemplo, an `awss3.Bucket` toma un argumento de tipo props. `awss3.BucketProps` Usa una estructura literal para escribir argumentos de utilería.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

Estructuras genéricas

En algunos lugares, AWS CDK utiliza JavaScript matrices u objetos sin tipo como entrada a un método. (Consulte, por ejemplo, el método AWS CodeBuild de `BuildSpec.fromObject()`.) En Go, estos objetos se representan como sectores y una interfaz vacía, respectivamente.

El CDK proporciona diversas funciones auxiliares, por ejemplo, `jsii.Strings` para crear segmentos que contienen tipos primitivos.

```
jsii.Strings("One", "Two", "Three")
```

Desarrollo de construcciones personalizadas

En Go, suele ser más sencillo escribir una nueva construcción que extender una existente. En primer lugar, defina un nuevo tipo de estructura e incorpore de forma anónima uno o más tipos existentes si se desea una semántica similar a la de una extensión. Escribe métodos para cualquier funcionalidad nueva que vayas a añadir y los campos necesarios para almacenar los datos que necesites. Defina una interfaz de accesorios si su construcción la necesita. Por último, escribe una función de fábrica `NewMyConstruct()` para devolver una instancia de tu construcción.

Si simplemente va a cambiar algunos valores por defecto de una construcción existente o a añadir un comportamiento sencillo al instanciar, no necesitará toda esa combinación. En su lugar, escribe una función de fábrica que llame a la función de fábrica de la construcción que estás «extendiendo». En otros lenguajes de CDK, por ejemplo, puedes crear una `TypedBucket` construcción que aplique el tipo de objetos de un bucket de Amazon S3 anulando el `s3.Bucket` tipo y, en el inicializador del nuevo tipo, añadiendo una política de bucket que permita añadir al bucket únicamente extensiones de nombre de archivo especificadas. En Go, es más fácil escribir simplemente una `NewTypedBucket` que devuelva una `s3.Bucket` (uso instanciados`s3.NewBucket`) a la que se haya agregado una política de bucket adecuada. No es necesario ningún tipo de construcción nueva porque la funcionalidad ya está disponible en la construcción de bucket estándar; la nueva «construcción» simplemente proporciona una forma más sencilla de configurarla.

Construir, sintetizar e implementar

Compila AWS CDK automáticamente la aplicación antes de ejecutarla. Sin embargo, puede resultar útil compilar la aplicación manualmente para comprobar si hay errores y realizar pruebas. Puedes hacerlo ejecutándolo `go build` en una línea de comandos desde el directorio raíz de tu proyecto.

Ejecute cualquier prueba que haya escrito ejecutándola `go test` en una línea de comandos.

Las pilas definidas en tu AWS CDK aplicación se pueden sintetizar e implementar de forma individual o conjunta mediante los siguientes comandos. Por lo general, debes estar en el directorio principal de tu proyecto cuando los emitas.

- `cdk synth`: sintetiza una AWS CloudFormation plantilla a partir de una o más de las pilas de tu AWS CDK aplicación.
- `cdk deploy`: despliega los recursos definidos por una o más de las pilas de tu aplicación en. AWS CDK AWS

Puedes especificar los nombres de varias pilas que se van a sintetizar o implementar en un solo comando. Si tu aplicación define solo una pila, no es necesario que la especifiques.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy  # app defines two or more stacks; two are deployed
```

También puedes usar los caracteres comodín * (cualquier número de caracteres) y? (cualquier carácter individual) para identificar las pilas por patrón. Cuando utilice caracteres comodín, escriba el patrón entre comillas. De lo contrario, es posible que el shell intente ampliarlo hasta incluir los nombres de los archivos del directorio actual antes de pasarlos al AWS CDK kit de herramientas.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> No necesita sintetizar las pilas de forma explícita antes de desplegarlas; `cdk deploy` realiza este paso para asegurarse de que se implementa el código más reciente.

Para obtener la documentación completa del `cdk` comando, consulte. the section called “AWS CDK Kit de herramientas”

Desarrollo de AWS CDK aplicaciones

Desarrolle AWS Cloud Development Kit (AWS CDK) aplicaciones.

Temas

Personalización de componentes fijos de la AWS biblioteca de componentes

Personalice las construcciones de la biblioteca de AWS construcciones mediante trampillas de escape, anulaciones sin procesar y recursos personalizados.

Temas

Uso de trampillas de escape

La biblioteca AWS Construct proporciona construcciones con distintos niveles de abstracción.

En el nivel más alto, la AWS CDK aplicación y las pilas que contiene son en sí mismas abstracciones de toda la infraestructura de nube o de partes importantes de la misma. Se pueden parametrizar para implementarlos en diferentes entornos o para diferentes necesidades.

Las abstracciones son herramientas poderosas para diseñar e implementar aplicaciones en la nube. AWS CDK Esto le da el poder no solo de construir con sus abstracciones, sino también de crear nuevas abstracciones. Utilizando como guía los constructos L2 y L3 de código abierto existentes, puede crear sus propios constructos L2 y L3 para reflejar las mejores prácticas y opiniones de su propia organización.

Ninguna abstracción es perfecta, e incluso las buenas abstracciones no pueden cubrir todos los casos de uso posibles. Durante el desarrollo, es posible que encuentres una construcción que casi se adapte a tus necesidades y que requiera una personalización pequeña o grande.

Por esta razón, AWS CDK proporciona formas de romper con el modelo de construcción. Esto incluye pasar a una abstracción de nivel inferior o a un modelo completamente diferente. Las trampillas de escape te permiten escapar del AWS CDK paradigma y personalizarlo de forma que se adapte a tus necesidades. Luego, puedes agrupar los cambios en una nueva estructura para eliminar la complejidad subyacente y proporcionar una API limpia para otros desarrolladores.

A continuación, se muestran ejemplos de situaciones en las que puede utilizar trampillas de escape:

- Hay una función de AWS servicio disponible a través de AWS CloudFormation ella, pero no tiene ninguna estructura de nivel 2.
- Una función AWS de servicio está disponible a través del servicio AWS CloudFormation, y existen estructuras de nivel 2 para el servicio, pero estas aún no exponen la función. Como las construcciones de nivel 2 son seleccionadas por el equipo de CDK, es posible que no estén disponibles de forma inmediata para las nuevas funciones.
- La función aún no está disponible en absoluto. AWS CloudFormation

  Para determinar si una función está disponible a través de ella AWS CloudFormation, consulte la Referencia de tipos de AWS recursos y propiedades.

Desarrolle trampillas de escape para las construcciones L1

Si las construcciones L2 no están disponibles para el servicio, puede usar las construcciones L1 generadas automáticamente. Estos recursos se pueden reconocer por su nombre que comience por, por ejemplo`Cfn`, o. `CfnBucket` `CfnRole` Los instancias exactamente como usarías el recurso equivalente AWS CloudFormation .

Por ejemplo, para crear una instancia de un bucket L1 de Amazon S3 de bajo nivel con la analítica habilitada, escribiría algo como lo siguiente.

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",
        // ...
    }
});
```

En raras ocasiones, puede que desee definir un recurso que no tenga una clase correspondiente. `CfnXxx` Podría tratarse de un tipo de recurso nuevo que aún no se haya publicado en la especificación del AWS CloudFormation recurso. En estos casos, puede crear una instancia `cdk.CfnResource` directamente y especificar el tipo y las propiedades del recurso. Esto se muestra en el siguiente ejemplo.

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"
            }
```

```
        }
    }
});
```

Desarrolle trampillas de escape para construcciones L2

Si a una construcción L2 le falta una característica o si está intentando solucionar un problema, puede modificar la construcción L1 encapsulada por la construcción L2.

Todas las construcciones L2 contienen en su interior la construcción L1 correspondiente. Por ejemplo, la construcción de alto nivel envuelve la `Bucket` construcción de nivel inferior. `CfnBucket` Como `CfnBucket` corresponde directamente al AWS CloudFormation recurso, expone todas las funciones que están disponibles a través de él. AWS CloudFormation

El enfoque básico para acceder a la construcción L1 es usar `construct.node.defaultChild` (Python:`default_child`), convertirla en el tipo correcto (si es necesario) y modificar sus propiedades. De nuevo, tomemos el ejemplo de un`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",
        // ...
    }
};
```

También puedes usar este objeto para cambiar AWS CloudFormation opciones como `Metadata` y`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"
};
```

Escotillas de escape

AWS CDK También proporciona la capacidad de subir un nivel de abstracción, al que podríamos denominar trampilla de «escape». Si tiene una construcción L1, por ejemplo`CfnBucket`, puede crear una nueva construcción L2 (`Bucket`en este caso) para incluir la construcción L1.

Esto resulta práctico cuando se crea un recurso de L1 pero se quiere utilizar con una construcción que requiere un recurso de L2. También resulta útil cuando se quieren utilizar métodos prácticos como los `.grantXxxx()` que no están disponibles en la construcción L1.

Se pasa al nivel de abstracción superior mediante un método estático en la clase L2 denominado, `.fromCfnXxxxx()` por ejemplo, para los buckets de Amazon `Bucket.fromCfnBucket()` S3. El recurso L1 es el único parámetro.

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);
```

Las construcciones de L2 creadas a partir de las construcciones de L1 son objetos proxy que hacen referencia al recurso de L1, similares a los que se crean a partir de nombres de recursos, ARN o búsquedas. Las modificaciones de estas construcciones no afectan a la AWS CloudFormation plantilla sintetizada final (sin embargo, dado que tiene el recurso L1, puede modificarlo en su lugar). Para obtener más información sobre los objetos proxy, consulte. the section called “Hacer referencia a los recursos de tu cuenta AWS”

Para evitar confusiones, no cree varias construcciones de L2 que hagan referencia a la misma construcción de L1. Por ejemplo, si extraes el `CfnBucket` de un `Bucket` mediante la técnica de la sección anterior, no deberías crear una segunda `Bucket` instancia llamando `Bucket.fromCfnBucket()` con eso. `CfnBucket` De hecho, funciona como cabría esperar (solo `AWS::S3::Bucket` se sintetiza una), pero dificulta el mantenimiento del código.

Anulaciones sin procesar

Si faltan propiedades en la construcción L1, puede omitir todo tipo de escritura mediante anulaciones sin procesar. Esto también permite eliminar las propiedades sintetizadas.

Utilice uno de los `addOverride` métodos (Python:`add_override`), como se muestra en el siguiente ejemplo.

TypeScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild as s3.CfnBucket;

// Use dot notation to address inside the resource template fragment
cfnBucket.addOverride('Properties.VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addDeletionOverride('Properties.VersioningConfiguration.Status');

// use index (0 here) to address an element of a list
cfnBucket.addOverride('Properties.Tags.0.Value', 'NewValue');
cfnBucket.addDeletionOverride('Properties.Tags.0');

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.addPropertyOverride('VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addPropertyDeletionOverride('VersioningConfiguration.Status');
cfnBucket.addPropertyOverride('Tags.0.Value', 'NewValue');
cfnBucket.addPropertyDeletionOverride('Tags.0');
```

JavaScript

```
// Get the CloudFormation resource
const cfnBucket = bucket.node.defaultChild ;

// Use dot notation to address inside the resource template fragment
cfnBucket.addOverride('Properties.VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addDeletionOverride('Properties.VersioningConfiguration.Status');
```

```
// use index (0 here) to address an element of a list
cfnBucket.addOverride('Properties.Tags.0.Value', 'NewValue');
cfnBucket.addDeletionOverride('Properties.Tags.0');

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.addPropertyOverride('VersioningConfiguration.Status', 'NewStatus');
cfnBucket.addPropertyDeletionOverride('VersioningConfiguration.Status');
cfnBucket.addPropertyOverride('Tags.0.Value', 'NewValue');
cfnBucket.addPropertyDeletionOverride('Tags.0');
```

Python

```
# Get the CloudFormation resource
cfn_bucket = bucket.node.default_child

# Use dot notation to address inside the resource template fragment
cfn_bucket.add_override("Properties.VersioningConfiguration.Status", "NewStatus")
cfn_bucket.add_deletion_override("Properties.VersioningConfiguration.Status")

# use index (0 here) to address an element of a list
cfn_bucket.add_override("Properties.Tags.0.Value", "NewValue")
cfn_bucket.add_deletion_override("Properties.Tags.0")

# addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfn_bucket.add_property_override("VersioningConfiguration.Status", "NewStatus")
cfn_bucket.add_property_deletion_override("VersioningConfiguration.Status")
cfn_bucket.add_property_override("Tags.0.Value", "NewValue")
cfn_bucket.add_property_deletion_override("Tags.0")
```

Java

```
// Get the CloudFormation resource
CfnBucket cfnBucket = (CfnBucket)bucket.getNode().getDefaultChild();

// Use dot notation to address inside the resource template fragment
cfnBucket.addOverride("Properties.VersioningConfiguration.Status", "NewStatus");
cfnBucket.addDeletionOverride("Properties.VersioningConfiguration.Status");

// use index (0 here) to address an element of a list
cfnBucket.addOverride("Properties.Tags.0.Value", "NewValue");
```

```
cfnBucket.addDeletionOverride("Properties.Tags.0");

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.addPropertyOverride("VersioningConfiguration.Status", "NewStatus");
cfnBucket.addPropertyDeletionOverride("VersioningConfiguration.Status");
cfnBucket.addPropertyOverride("Tags.0.Value", "NewValue");
cfnBucket.addPropertyDeletionOverride("Tags.0");
```

C#

```
// Get the CloudFormation resource
var cfnBucket = (CfnBucket)bucket.node.defaultChild;

// Use dot notation to address inside the resource template fragment
cfnBucket.AddOverride("Properties.VersioningConfiguration.Status", "NewStatus");
cfnBucket.AddDeletionOverride("Properties.VersioningConfiguration.Status");

// use index (0 here) to address an element of a list
cfnBucket.AddOverride("Properties.Tags.0.Value", "NewValue");
cfnBucket.AddDeletionOverride("Properties.Tags.0");

// addPropertyOverride is a convenience function for paths starting with
 "Properties."
cfnBucket.AddPropertyOverride("VersioningConfiguration.Status", "NewStatus");
cfnBucket.AddPropertyDeletionOverride("VersioningConfiguration.Status");
cfnBucket.AddPropertyOverride("Tags.0.Value", "NewValue");
cfnBucket.AddPropertyDeletionOverride("Tags.0");
```

Recursos personalizados

Si la función no está disponible mediante una llamada directa a la API AWS CloudFormation, sino solo a través de ella, debes escribir un recurso AWS CloudFormation personalizado para realizar la llamada a la API que necesitas. Puedes usarlo AWS CDK para escribir recursos personalizados y agruparlos en una interfaz de construcción normal. Desde la perspectiva de un consumidor de tu estilo, la experiencia parecerá nativa.

La creación de un recurso personalizado implica escribir una función Lambda que responda a los eventos del recurso y del `CREATE` `DELETE` ciclo de vida. `UPDATE` Si su recurso personalizado solo necesita realizar una única llamada a la API, considere la posibilidad de utilizar la AwsCustomResource. Esto permite realizar llamadas arbitrarias al SDK durante una AWS

CloudFormation implementación. De lo contrario, debe escribir su propia función Lambda para realizar el trabajo que necesita realizar.

El tema es demasiado amplio para tratarlo por completo aquí, pero los siguientes enlaces deberían ayudarte a empezar:

- Recursos personalizados
- Ejemplo de recurso personalizado
- Para ver un ejemplo más completo, consulte la DnsValidatedCertificateclase en la biblioteca estándar de CDK. Esto se implementa como un recurso personalizado.

Obtener un valor de una variable de entorno

Para obtener el valor de una variable de entorno, utilice un código como el siguiente. Este código obtiene el valor de la variable de entornoMYBUCKET.

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 valor

Consulte the section called “Parámetros” para obtener información sobre el uso de AWS CloudFormation parámetros con AWS CDK.

Para obtener una referencia a un recurso en una AWS CloudFormation plantilla existente, consultethe section called “Importar una AWS CloudFormation plantilla”.

Importar una AWS CloudFormation plantilla existente

Importe recursos de una AWS CloudFormation plantilla a sus AWS Cloud Development Kit (AWS CDK) aplicaciones mediante la `cloudformation-include.CfnInclude`construcción para convertir los recursos en construcciones de nivel 1.

Tras la importación, puedes trabajar con estos recursos en tu aplicación de la misma manera que lo harías si estuvieran definidos originalmente en AWS CDK el código. También puede usar estas construcciones de L1 dentro de las construcciones de nivel superior AWS CDK . Por ejemplo, esto puede permitirle usar los métodos de concesión de permisos L2 con los recursos que definen.

Básicamente, la `cloudformation-include.CfnInclude` construcción agrega un contenedor de AWS CDK API a cualquier recurso de la plantilla AWS CloudFormation . Usa esta capacidad para importar tus AWS CloudFormation plantillas existentes a AWS CDK una pieza a la vez. De este modo, puede administrar sus recursos existentes mediante AWS CDK componentes constructos para aprovechar las ventajas de las abstracciones de nivel superior. También puedes usar esta función para vender tus AWS CloudFormation plantillas a AWS CDK los desarrolladores proporcionando una API de construcción. AWS CDK

> **Note**
>
> AWS CDK También se incluye la versión 1 `aws-cdk-lib.CfnInclude`, que anteriormente se utilizaba con el mismo propósito general. Sin embargo, carece de gran parte de la funcionalidad de`cloudformation-include.CfnInclude`.

Temas

- Importación de una AWS CloudFormation plantilla
- Acceder a los recursos importados
- Sustituir parámetros
- Otros elementos de la plantilla
- Pilas anidadas

Importación de una AWS CloudFormation plantilla

La siguiente es una AWS CloudFormation plantilla de ejemplo que utilizaremos para proporcionar ejemplos en este tema. Copia y guarda la plantilla como se `my-template.json` indica a continuación. Después de trabajar con estos ejemplos, puede explorar más a fondo utilizando cualquiera de las AWS CloudFormation plantillas desplegadas existentes. Puede obtenerlas desde la AWS CloudFormation consola.

```
{
```

```
  "Resources": {
    "MyBucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

Puedes trabajar con plantillas JSON o YAML. Recomendamos usar JSON si está disponible, ya que los analizadores YAML pueden variar ligeramente en cuanto a lo que aceptan.

A continuación, se muestra un ejemplo de cómo importar la plantilla de ejemplo a tu AWS CDK aplicación utilizando. `cloudformation-include` Las plantillas se importan en el contexto de una pila de CDK.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as cfninc from 'aws-cdk-lib/cloudformation-include';
import { Construct } from 'constructs';

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

    const template = new cfninc.CfnInclude(this, 'Template', {
      templateFile: 'my-template.json',
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const cfninc = require('aws-cdk-lib/cloudformation-include');

class MyStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);
```

```
    const template = new cfninc.CfnInclude(this, 'Template', {
      templateFile: 'my-template.json',
    });
  }
}

module.exports = { MyStack }
```

Python

```
import aws_cdk as cdk
from aws_cdk import cloudformation_include as cfn_inc
from constructs import Construct

class MyStack(cdk.Stack):

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

        template = cfn_inc.CfnInclude(self, "Template",
            template_file="my-template.json")
```

Java

```
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.cloudformation.include.CfnInclude;
import software.constructs.Construct;

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

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

        CfnInclude template = CfnInclude.Builder.create(this, "Template")
         .templateFile("my-template.json")
         .build();
    }
}
```

C#

```
using Amazon.CDK;
using Constructs;
using cfnInc = Amazon.CDK.CloudFormation.Include;

namespace MyApp
{
    public class MyStack : Stack
    {
        internal MyStack(Construct scope, string id, IStackProps props = null) :
 base(scope, id, props)
        {
            var template = new cfnInc.CfnInclude(this, "Template", new
 cfnInc.CfnIncludeProps
            {
                TemplateFile = "my-template.json"
            });
        }
    }
}
```

De forma predeterminada, la importación de un recurso conserva el identificador lógico original del recurso que figura en la plantilla. Este comportamiento es adecuado para importar una AWS CloudFormation plantilla a la AWS CDK, donde se deben conservar los ID lógicos. AWS CloudFormation necesita esta información para reconocer estos recursos importados como los mismos recursos de la AWS CloudFormation plantilla.

Si está desarrollando un contenedor AWS CDK de componentes para la plantilla de forma que puedan utilizarla otros AWS CDK desarrolladores, opte por AWS CDK generar nuevos identificadores de recursos. De este modo, la construcción se puede utilizar varias veces en una pila sin conflictos de nombres. Para ello, defina la `preserveLogicalIds` propiedad en `false` al importar la plantilla. A continuación, se muestra un ejemplo:

TypeScript

```
const template = new cfninc.CfnInclude(this, 'MyConstruct', {
  templateFile: 'my-template.json',
  preserveLogicalIds: false
});
```

JavaScript

```
const template = new cfninc.CfnInclude(this, 'MyConstruct', {
  templateFile: 'my-template.json',
  preserveLogicalIds: false
});
```

Python

```
template = cfn_inc.CfnInclude(self, "Template",
    template_file="my-template.json",
    preserve_logical_ids=False)
```

Java

```
CfnInclude template = CfnInclude.Builder.create(this, "Template")
 .templateFile("my-template.json")
 .preserveLogicalIds(false)
 .build();
```

C#

```
var template = new cfnInc.CfnInclude(this, "Template", new cfn_inc.CfnIncludeProps
{
    TemplateFile = "my-template.json",
    PreserveLogicalIds = false
});
```

Para poner los recursos importados bajo el control de tu AWS CDK aplicación, agrega la pila aApp:

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { MyStack } from '../lib/my-stack';

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

JavaScript

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

```
const { MyStack } = require('../lib/my-stack');

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

Python

```
import aws_cdk as cdk
from mystack.my_stack import MyStack

app = cdk.App()
MyStack(app, "MyStack")
```

Java

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

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

        new MyStack(app, "MyStack");
    }
}
```

C#

```
using Amazon.CDK;

namespace CdkApp
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new MyStack(app, "MyStack");
        }
    }
}
```

Para comprobar que no se produzcan cambios imprevistos en los AWS recursos de la pila, puedes realizar una diferencia. Usa el AWS CDK CLI `cdk diff` comando y omite cualquier metadato específico AWS CDK. A continuación, se muestra un ejemplo:

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

Tras importar una AWS CloudFormation plantilla, la AWS CDK aplicación debería convertirse en la fuente fiable de los recursos importados. Para realizar cambios en los recursos, modifíquelos en la AWS CDK aplicación e impleméntelos con el AWS CDK CLI cdk deploy comando.

Acceder a los recursos importados

El nombre `template` del código de ejemplo representa la AWS CloudFormation plantilla importada. Para acceder a un recurso desde ella, utilice el `getResource()`método del objeto. Para acceder al recurso devuelto como un tipo de recurso específico, asigne el resultado al tipo deseado. Esto no es necesario en Python o JavaScript. A continuación, se muestra un ejemplo:

TypeScript

```
const cfnBucket = template.getResource('MyBucket') as s3.CfnBucket;
```

JavaScript

```
const cfnBucket = template.getResource('MyBucket');
```

Python

```
cfn_bucket = template.get_resource("MyBucket")
```

Java

```
CfnBucket cfnBucket = (CfnBucket)template.getResource("MyBucket");
```

C#

```
var cfnBucket = (CfnBucket)template.GetResource("MyBucket");
```

A partir de este ejemplo, ahora `cfnBucket` es una instancia de la `aws-s3.CfnBucket`clase. Se trata de una construcción L1 que representa el AWS CloudFormation recurso correspondiente.

Puede tratarlo como cualquier otro recurso de su tipo. Por ejemplo, puede obtener su valor de ARN con la `bucket.attrArn` propiedad.

Para empaquetar el `CfnBucket` recurso L1 en una `aws-s3.Bucket`instancia L2, utilice los métodos estáticos `fromBucketArn()`, `fromBucketAttributes()`o. `fromBucketName()` Por lo general, el `fromBucketName()` método es el más conveniente. A continuación, se muestra un ejemplo:

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);
```

Otras construcciones L2 tienen métodos similares para crear la construcción a partir de un recurso existente.

Al incluir una construcción de L1 en una construcción de L2, no se crea un recurso nuevo. A partir de nuestro ejemplo, no vamos a crear un segundo bucket de S3;. En su lugar, la nueva `Bucket` instancia encapsula la existente`CfnBucket`.

A partir del ejemplo, ahora `bucket` es una construcción L2 que se comporta como cualquier otra `Bucket` construcción L2. Por ejemplo, puedes conceder a una AWS Lambda función acceso de escritura al depósito mediante el práctico método del depósito. `grantWrite()` No es necesario

que defina manualmente la política AWS Identity and Access Management (IAM) necesaria. A continuación, se muestra un ejemplo:

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

Sustituir parámetros

Si la AWS CloudFormation plantilla contiene parámetros, puede sustituirlos por valores de tiempo de creación en el momento de la importación mediante la `parameters` propiedad. En el siguiente ejemplo, sustituimos el `UploadBucket` parámetro por el ARN de un bucket definido en otra parte de nuestro AWS CDK código.

TypeScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

JavaScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

Python

```
template = cfn_inc.CfnInclude(self, "Template",
    template_file="my-template.json",
    parameters=dict(UploadBucket=bucket.bucket_arn)
)
```

Java

```
CfnInclude template = CfnInclude.Builder.create(this, "Template")
 .templateFile("my-template.json")
 .parameters(java.util.Map.of(    // Map.of requires Java 9+
   "UploadBucket", bucket.getBucketArn()))
 .build();
```

C#

```
var template = new cfnInc.CfnInclude(this, "Template", new cfnInc.CfnIncludeProps
{
    TemplateFile = "my-template.json",
    Parameters = new Dictionary<string, string>
    {
        { "UploadBucket", bucket.BucketArn }
    }
});
```

Otros elementos de la plantilla

Puede importar cualquier elemento AWS CloudFormation de la plantilla, no solo los recursos. Los elementos importados pasan a formar parte de la AWS CDK pila. Para importar estos elementos, utilice los siguientes métodos del `CfnInclude` objeto:

- `getCondition()`— AWS CloudFormation condiciones.
- `getHook()`— AWS CloudFormation ganchos para despliegues azules/verdes.
- `getMapping()`— mapeos. AWS CloudFormation
- `getOutput()`— salidas AWS CloudFormation .
- `getParameter()`— AWS CloudFormation parámetros.
- `getRule()`— AWS CloudFormation reglas para AWS Service Catalog plantillas.

Cada uno de estos métodos devuelve una instancia de una clase que representa el tipo específico de AWS CloudFormation elemento. Estos objetos son mutables. Los cambios que realices en ellos aparecerán en la plantilla que se genere a partir de la AWS CDK pila. A continuación se muestra un ejemplo en el que se importa un parámetro de la plantilla y se modifica su valor predeterminado:

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";
```

Pilas anidadas

Puede importar pilas anidadas especificándolas al importar su plantilla principal o más adelante. La plantilla anidada debe almacenarse en un archivo local, pero debe hacerse referencia a ella como `NestedStack` recurso en la plantilla principal. Además, el nombre del recurso utilizado en el AWS CDK código debe coincidir con el nombre utilizado para la pila anidada en la plantilla principal.

Dada esta definición de recurso en la plantilla principal, el código siguiente muestra cómo importar la pila anidada a la que se hace referencia en ambos sentidos.

```
"NestedStack": {
  "Type": "AWS::CloudFormation::Stack",
  "Properties": {
    "TemplateURL": "https://my-s3-template-source.s3.amazonaws.com/nested-stack.json"
  }
```

TypeScript

```
// include nested stack when importing main stack
const mainTemplate = new cfninc.CfnInclude(this, 'MainStack', {
  templateFile: 'main-template.json',
  loadNestedStacks: {
    'NestedStack': {
      templateFile: 'nested-template.json',
    },
  },
});

// or add it some time after importing the main stack
const nestedTemplate = mainTemplate.loadNestedStack('NestedTemplate', {
  templateFile: 'nested-template.json',
});
```

JavaScript

```
// include nested stack when importing main stack
const mainTemplate = new cfninc.CfnInclude(this, 'MainStack', {
  templateFile: 'main-template.json',
  loadNestedStacks: {
    'NestedStack': {
      templateFile: 'nested-template.json',
    },
```

```
  },
});

// or add it some time after importing the main stack
const nestedTemplate = mainTemplate.loadNestedStack('NestedStack', {
  templateFile: 'my-nested-template.json',
});
```

Python

```
# include nested stack when importing main stack
main_template = cfn_inc.CfnInclude(self, "MainStack",
    template_file="main-template.json",
    load_nested_stacks=dict(NestedStack=
        cfn_inc.CfnIncludeProps(template_file="nested-template.json")))

# or add it some time after importing the main stack
nested_template = main_template.load_nested_stack("NestedStack",
    template_file="nested-template.json")
```

Java

```
CfnInclude mainTemplate = CfnInclude.Builder.create(this, "MainStack")
 .templateFile("main-template.json")
 .loadNestedStacks(java.util.Map.of(    // Map.of requires Java 9+
    "NestedStack", CfnIncludeProps.builder()
    .templateFile("nested-template.json").build()))
 .build();

// or add it some time after importing the main stack
IncludedNestedStack nestedTemplate = mainTemplate.loadNestedStack("NestedTemplate",
 CfnIncludeProps.builder()
  .templateFile("nested-template.json")
  .build());
```

C#

```
// include nested stack when importing main stack
var mainTemplate = new cfnInc.CfnInclude(this, "MainStack", new
 cfnInc.CfnIncludeProps
{
    TemplateFile = "main-template.json",
    LoadNestedStacks = new Dictionary<string, cfnInc.ICfnIncludeProps>
```

```
    {
        { "NestedStack", new cfnInc.CfnIncludeProps { TemplateFile = "nested-
template.json" } }
    }
});

// or add it some time after importing the main stack
var nestedTemplate = mainTemplate.LoadNestedStack("NestedTemplate", new
 cfnInc.CfnIncludeProps {
    TemplateFile = 'nested-template.json'
});
```

Puede importar varias pilas anidadas con cualquiera de los dos métodos. Al importar la plantilla principal, se proporciona un mapeo entre el nombre del recurso de cada pila anidada y su archivo de plantilla. Este mapeo puede contener cualquier número de entradas. Para hacerlo después de la importación inicial, llama `loadNestedStack()` una vez por cada pila anidada.

Tras importar una pila anidada, puedes acceder a ella mediante el método de la plantilla principal. `getNestedStack()`

TypeScript

```
const nestedStack = mainTemplate.getNestedStack('NestedStack').stack;
```

JavaScript

```
const nestedStack = mainTemplate.getNestedStack('NestedStack').stack;
```

Python

```
nested_stack = main_template.get_nested_stack("NestedStack").stack
```

Java

```
NestedStack nestedStack = mainTemplate.getNestedStack("NestedStack").getStack();
```

C#

```
var nestedStack = mainTemplate.GetNestedStack("NestedStack").Stack;
```

El `getNestedStack()` método devuelve una `IncludedNestedStack`instancia. Desde esta instancia, puede acceder a la AWS CDK `NestedStack`instancia a través de la `stack` propiedad, como se muestra en el ejemplo. También puedes acceder al objeto de AWS CloudFormation plantilla original a través `includedTemplate` del cual puedes cargar recursos y otros AWS CloudFormation elementos.

Obtenga un valor del almacén de parámetros de Systems Manager

AWS Cloud Development Kit (AWS CDK) Puede recuperar el valor de los atributos del almacén de AWS Systems Manager parámetros. Durante la síntesis, AWS CDK produce un token que se resuelve AWS CloudFormation durante el despliegue.

AWS CDK Soporta la recuperación de valores simples y seguros. Puede solicitar una versión específica de cualquier tipo de valor. Para valores simples, puede omitir la versión de su solicitud para recuperar la versión más reciente. Para los valores seguros, debe especificar la versión al solicitar el valor del atributo seguro.

> **Note**
>
> En este tema se muestra cómo leer los atributos del almacén de AWS Systems Manager parámetros. También puede leer los secretos del AWS Secrets Manager (consulteObtenga un valor de AWS Secrets Manager).

Temas

- Lea los valores de Systems Manager en el momento de la implementación
- Lea los valores de Systems Manager en el momento de la síntesis
- Escribir valores en Systems Manager

Lea los valores de Systems Manager en el momento de la implementación

Para leer los valores del almacén de parámetros de Systems Manager, utilice el valueForStringparámetro y valueForSecureStringParameterlos métodos. Elija un método en función de si el atributo que desea es una cadena simple o un valor de cadena seguro. Estos métodos devuelven los símbolos, no el valor real. El valor se resuelve AWS CloudFormation durante la implementación. A continuación, se muestra un ejemplo:

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
```

En la actualidad, hay un número limitado de AWS servicios que admiten esta función.

Lea los valores de Systems Manager en el momento de la síntesis

A veces, resulta útil proporcionar un parámetro en el momento de la síntesis. De este modo, la AWS CloudFormation plantilla siempre utilizará el mismo valor en lugar de resolverlo durante la implementación.

Para leer un valor del almacén de parámetros de Systems Manager en el momento de la síntesis, utilice el valueFromLookupmétodo (Python:`value_from_lookup`). Este método devuelve el valor real del parámetro en forma de the section called “Context” valor. Si el valor aún no está almacenado en caché `cdk.json` o no se ha pasado a través de la línea de comandos, se recupera de la AWS cuenta corriente. Por este motivo, la pila debe sintetizarse con información AWS ambiental explícita.

A continuación, se muestra un ejemplo:

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");
```

Solo se pueden recuperar cadenas simples de Systems Manager. No se pueden recuperar las cadenas seguras. Siempre se devolverá la versión más reciente. No se pueden solicitar versiones específicas.

> **Important**
>
> El valor recuperado acabará en la AWS CloudFormation plantilla sintetizada. Esto puede suponer un riesgo para la seguridad, según quién tenga acceso a tus AWS CloudFormation plantillas y el tipo de valor que tengan. Por lo general, no utilices esta función para contraseñas, claves u otros valores que desees mantener privados.

Escribir valores en Systems Manager

Puede usar la AWS CLI AWS Management Console, el o un AWS SDK para configurar los valores de los parámetros de Systems Manager. Los ejemplos siguientes utilizan el comando CLI ssm put-parameter.

```
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"
```

Al actualizar un valor de SSM que ya existe, incluya también la opción. `--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"
```

Obtenga un valor de AWS Secrets Manager

Para usar los valores AWS Secrets Manager de tu AWS CDK aplicación, usa el método fromSecretAttributes(). Representa un valor que se recupera de Secrets Manager y se utiliza en el momento de la AWS CloudFormation implementación. A continuación, se muestra un ejemplo:

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**
>
> Utilice el comando AWS CLI CLI create-secret para crear un secreto desde la línea de comandos, por ejemplo, al probar:
>
> ```
> aws secretsmanager create-secret --name ImportedSecret --secret-string
>  mygroovybucket
> ```
>
> El comando devuelve un ARN que puede utilizar en el ejemplo anterior.

Una vez que haya creado una `Secret` instancia, puede obtener el valor del secreto a partir del `secretValue` atributo de la instancia. El valor se representa mediante una `SecretValue` instancia, un tipo especial dethe section called “Tokens”. Como es un símbolo, solo tiene significado después de la resolución. Tu aplicación CDK no necesita acceder a su valor real. En su lugar, la aplicación puede pasar la `SecretValue` instancia (o su cadena o representación numérica) al método CDK que necesite el valor.

Configurar una CloudWatch alarma

Usa el paquete aws-cloudwatch para configurar las CloudWatch alarmas de Amazon en las métricas. CloudWatch Puede usar métricas predefinidas o crear las suyas propias.

Temas

- Utilizar una métrica existente
- Crear su propia métrica
- Crear la alarma

Utilizar una métrica existente

Muchos módulos de AWS Construct Library permiten configurar una alarma en una métrica existente pasando el nombre de la métrica a un método práctico de una instancia de un objeto que tiene

métricas. Por ejemplo, dada una cola de Amazon SQS, puede obtener la métrica del método metric () ApproximateNumberOfMessagesVisiblede la cola:

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");
```

Crear su propia métrica

Cree su propia métrica de la siguiente manera, donde el valor del espacio de nombres debe ser algo parecido a AWS/SQS para una cola de Amazon SQS. También debe especificar el nombre y la dimensión de la métrica:

TypeScript

```
const metric = new cloudwatch.Metric({
  namespace: 'MyNamespace',
  metricName: 'MyMetric',
  dimensionsMap: { MyDimension: 'MyDimensionValue' }
});
```

JavaScript

```
const metric = new cloudwatch.Metric({
  namespace: 'MyNamespace',
  metricName: 'MyMetric',
  dimensionsMap: { MyDimension: 'MyDimensionValue' }
});
```

Python

```
metric = cloudwatch.Metric(
    namespace="MyNamespace",
    metric_name="MyMetric",
    dimensionsMap=dict(MyDimension="MyDimensionValue")
)
```

Java

```
Metric metric = Metric.Builder.create()
        .namespace("MyNamespace")
        .metricName("MyMetric")
        .dimensionsMap(java.util.Map.of(    // Java 9 or later
            "MyDimension", "MyDimensionValue"))
        .build();
```

C#

```
var metric = new Metric(this, "Metric", new MetricProps
{
    Namespace = "MyNamespace",
    MetricName = "MyMetric",
    Dimensions = new Dictionary<string, object>
    {
        { "MyDimension", "MyDimensionValue" }
    }
});
```

Crear la alarma

Una vez que tenga una métrica, ya sea una existente o una que haya definido, puede crear una alarma. En este ejemplo, la alarma se activa cuando hay más de 100 unidades de la métrica en dos

de los últimos tres períodos de evaluación. Puede utilizar comparaciones como el valor inferior al de sus alarmas a través de la `comparisonOperator` propiedad. G reater-than-or-equal -to es el AWS CDK valor predeterminado, por lo que no necesitamos especificarlo.

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
});
```

Una forma alternativa de crear una alarma es usar el método createAlarm () de la métrica, que tiene básicamente las mismas propiedades que el constructor. `Alarm` No es necesario que pases la métrica, porque ya la conoces.

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
});
```

Guardar y recuperar valores de variables de contexto

Puede especificar variables de contexto con AWS Cloud Development Kit (AWS CDK) CLI o en el `cdk.json` archivo. A continuación, utilice el `TryGetContext` método para recuperar los valores.

Temas

- Especifique las variables de contexto
- Recupera los valores de las variables de contexto

Especifique las variables de contexto

Puede especificar una variable de contexto como parte de un AWS CDK CLI comando o en`cdk.json`.

Para crear una variable de contexto de línea de comandos, utilice la opción --context (-c), como se muestra en el siguiente ejemplo.

```
cdk synth -c bucket_name=mygroovybucket
```

Para especificar la misma variable de contexto y el mismo valor en el `cdk.json` archivo, utilice el código siguiente.

```
{
  "context": {
```

```
    "bucket_name": "myotherbucket"
  }
}
```

Si especifica una variable de contexto mediante el `cdk.json` archivo AWS CDK CLI y, el AWS CDK CLI valor tiene prioridad.

Recupera los valores de las variables de contexto

Para obtener el valor de una variable de contexto en tu aplicación, usa el `TryGetContext` método en el contexto de una construcción. (Es decir, cuando`this`, o `self` en Python, es una instancia de alguna construcción).

En este ejemplo, recuperamos el valor de la variable de `bucket_name` contexto. Si el valor solicitado no está definido, `TryGetContext` devuelve `undefined` (Noneen Python, `null` Java y C#, `nil` en Go) en lugar de generar una excepción.

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");
```

Fuera del contexto de una construcción, puedes acceder a la variable de contexto desde el objeto de la aplicación, de esta forma.

TypeScript

```
const app = new cdk.App();
const bucket_name = app.node.tryGetContext('bucket_name')
```

JavaScript

```
const app = new cdk.App();
const bucket_name = app.node.tryGetContext('bucket_name');
```

Python

```
app = cdk.App()
bucket_name = app.node.try_get_context("bucket_name")
```

Java

```
App app = App();
String bucketName = (String)app.getNode().tryGetContext("bucket_name");
```

C#

```
app = App();
var bucketName = app.Node.TryGetContext("bucket_name");
```

Para obtener más información sobre cómo trabajar con variables de contexto, consultethe section called “Context”.

Uso de recursos del Registro AWS CloudFormation Público

El registro AWS CloudFormation público le permite administrar las extensiones, tanto públicas como privadas, como los recursos, los módulos y los enlaces que están disponibles para su uso Cuenta de AWS. Puede utilizar extensiones de recursos públicos en sus AWS Cloud Development Kit (AWS CDK) aplicaciones con la `CfnResource`construcción.

Para obtener más información sobre el registro AWS CloudFormation público, consulte Uso del AWS CloudFormation registro en la Guía del AWS CloudFormation usuario.

Todas las extensiones públicas publicadas por AWS están disponibles para todas las cuentas de todas las regiones sin que tengas que hacer nada por tu parte. Sin embargo, debes activar cada extensión de terceros que quieras usar en cada cuenta y región en la que quieras usarla.

> **Note**
>
> Si la utilizas AWS CloudFormation con tipos de recursos de terceros, se te cobrarán cargos. Los cargos se basan en la cantidad de operaciones del manipulador que ejecute al mes y en la duración de las operaciones del manipulador. Consulte los CloudFormation precios para obtener todos los detalles.

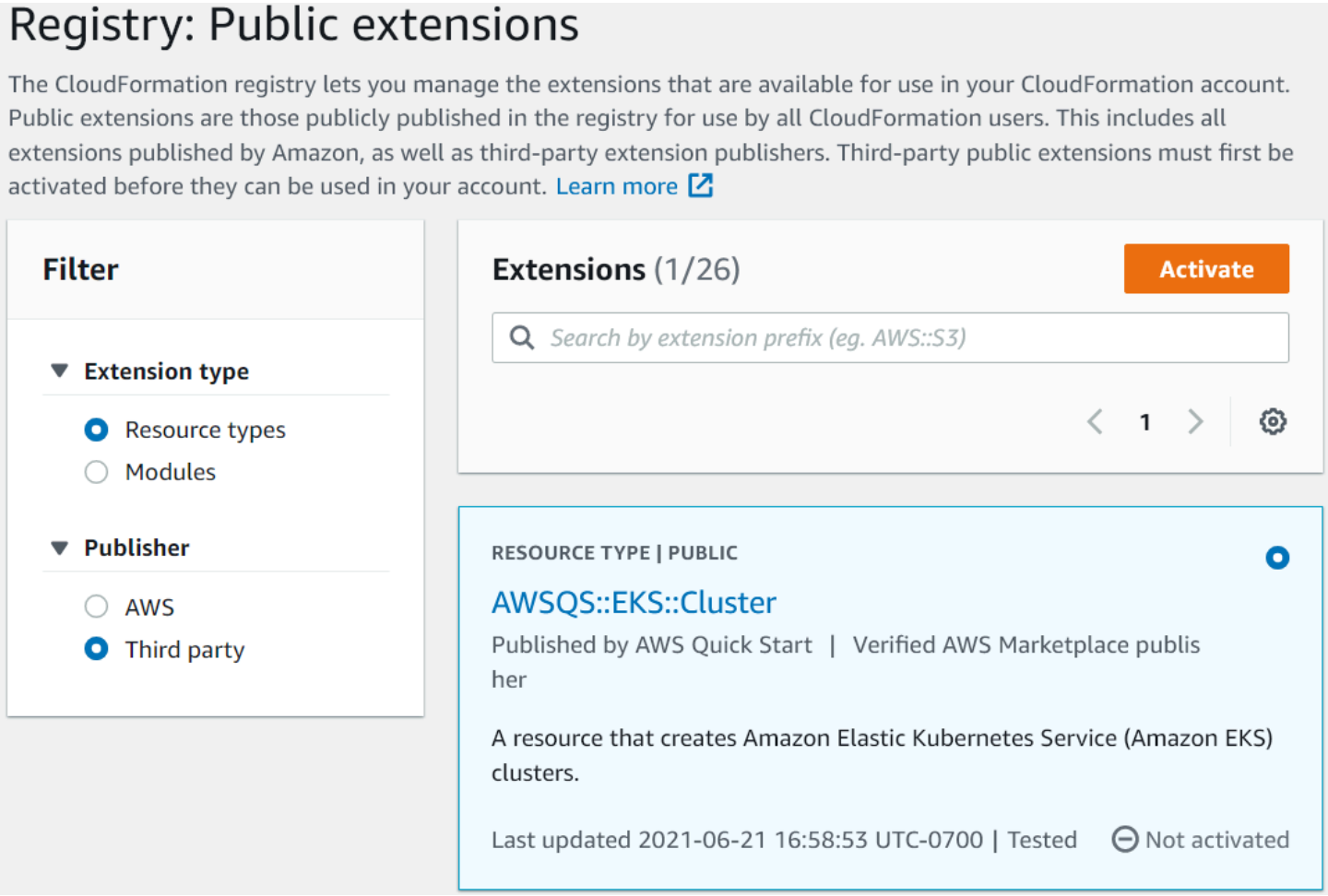
Para obtener más información sobre las extensiones públicas, consulte Uso de extensiones públicas CloudFormation en la Guía del AWS CloudFormation usuario

Temas

Activar un recurso de terceros en tu cuenta y región

Las extensiones publicadas por AWS no requieren activación. Siempre están disponibles en todas las cuentas y regiones. Puedes activar una extensión de terceros a través de AWS Management Console AWS Command Line Interface, mediante o desplegando un AWS CloudFormation recurso especial.

Para activar una extensión de terceros a través del AWS Management Console o ver qué recursos están disponibles



1. Inicia sesión en la AWS cuenta en la que quieres usar la extensión y, a continuación, cambia a la región en la que quieres usarla.
2. Ve a la CloudFormation consola a través del menú Servicios.
3. Selecciona Extensiones públicas en la barra de navegación y, a continuación, activa el botón de opción de terceros en Publisher. Aparece una lista de las extensiones públicas de terceros disponibles. (También puedes elegir AWSver una lista de las extensiones públicas publicadas por AWS, aunque no es necesario que las actives).
4. Navega por la lista y busca la extensión que quieres activar. También puedes buscarla y activar el botón de radio situado en la esquina superior derecha de la tarjeta de la extensión.
5. Pulse el botón Activar en la parte superior de la lista para activar la extensión seleccionada. Aparece la página de activación de la extensión.

6. En la página de activación, puede anular el nombre predeterminado de la extensión y especificar una función de ejecución y una configuración de registro. También puede elegir si desea actualizar automáticamente la extensión cuando se publique una nueva versión. Cuando hayas configurado estas opciones como desees, selecciona Activar la extensión en la parte inferior de la página.

Para activar una extensión de terceros mediante el AWS CLI

- Utilice el comando `activate-type`. Sustituya el ARN del tipo personalizado que desee utilizar donde se indique.

  A continuación, se muestra un ejemplo:

  ```
  aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
  ```

Para activar una extensión de terceros mediante nuestro CloudFormation CDK

- Implemente un tipo de recurso `AWS::CloudFormation::TypeActivation` y especifique las siguientes propiedades:

  a. `TypeName`- El nombre del tipo, por ejemplo`AWSQS::EKS::Cluster`.

  b. `MajorVersion`- El número de versión principal de la extensión que deseas. Omita si desea la versión más reciente.

  c. `AutoUpdate`- Si se debe actualizar automáticamente esta extensión cuando el editor publique una nueva versión secundaria. (Las actualizaciones de las versiones principales requieren cambiar la `MajorVersion` propiedad de forma explícita).

  d. `ExecutionRoleArn`- El ARN del rol de IAM con el que se ejecutará esta extensión.

  e. `LoggingConfig`- La configuración de registro de la extensión.

  La CDK puede implementar el `TypeActivation` recurso mediante la `CfnResource`construcción. Esto se muestra para las extensiones reales en la siguiente sección.

Añadir un recurso del Registro AWS CloudFormation Público a tu aplicación CDK

Use la `CfnResource`construcción para incluir un recurso del Registro AWS CloudFormation Público en su solicitud. Esta construcción se encuentra en el `aws-cdk-lib` módulo del CDK.

Por ejemplo, supongamos que hay un recurso público con el nombre `MY::S5::UltimateBucket` que desea utilizar en su AWS CDK aplicación. Este recurso tiene una propiedad: el nombre del bucket. La `CfnResource` instanciación correspondiente tiene este aspecto.

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"
    }
});
```

Implementación de AWS CDK aplicaciones

Implemente AWS Cloud Development Kit (AWS CDK) aplicaciones.

Temas

AWS CDK validación de políticas en el momento de la síntesis

Temas

Validación de políticas en el momento de la síntesis

Si usted o su organización utilizan alguna herramienta de validación de políticas, como AWS CloudFormation Guardla OPA, para definir las restricciones en la AWS CloudFormation plantilla, pueden integrarla en la plantilla AWS CDK en el momento de la síntesis. Al utilizar el complemento de validación de políticas adecuado, puede hacer que la AWS CDK aplicación compare la AWS CloudFormation plantilla generada con sus políticas inmediatamente después de la síntesis. Si se produce alguna infracción, la síntesis fallará y se imprimirá un informe en la consola.

La validación realizada AWS CDK en el momento de la síntesis valida los controles en un momento del ciclo de vida de la implementación, pero no puede afectar a las acciones que se producen fuera de la síntesis. Entre los ejemplos se incluyen las acciones que se toman directamente en la consola o mediante las API de servicio. No son resistentes a la alteración de las AWS CloudFormation plantillas después de la síntesis. Algún otro mecanismo para validar el mismo conjunto de reglas con más autoridad debería configurarse de forma independiente, como los AWS CloudFormation ganchos o AWS Config. Sin embargo, la capacidad de AWS CDK evaluar el conjunto de reglas durante el desarrollo sigue siendo útil, ya que mejorará la velocidad de detección y la productividad de los desarrolladores.

El objetivo de la validación de AWS CDK políticas es minimizar la cantidad de configuración necesaria durante el desarrollo y hacerlo lo más fácil posible.

> **Note**
>
> Esta función se considera experimental y tanto la API del complemento como el formato del informe de validación están sujetos a cambios en el futuro.

Temas

Para desarrolladores de aplicaciones

Para usar uno o más complementos de validación en su aplicación, use la `policyValidationBeta1` propiedad de`Stage`:

```
import { CfnGuardValidator } from '@cdklabs/cdk-validator-cfnguard';
const app = new App({
  policyValidationBeta1: [
    new CfnGuardValidator()
  ],
});
// only apply to a particular stage
const prodStage = new Stage(app, 'ProdStage', {
  policyValidationBeta1: [...],
});
```

Inmediatamente después de la síntesis, se invocarán todos los complementos registrados de esta manera para validar todas las plantillas generadas en el ámbito que haya definido. En concreto, si registra las plantillas en el App objeto, todas las plantillas estarán sujetas a validación.

> **Warning**
>
> Además de modificar el ensamblaje de la nube, los complementos pueden hacer cualquier cosa que pueda hacer tu AWS CDK aplicación. Pueden leer datos del sistema de archivos,

acceder a la red, etc. Como consumidor de un plugin, es tu responsabilidad comprobar que su uso es seguro.

AWS CloudFormation Guard complemento

El uso del CfnGuardValidatorcomplemento le permite AWS CloudFormation Guardrealizar validaciones de políticas. El `CfnGuardValidator` complemento viene con un conjunto selecto de controles AWS Control Tower proactivos integrados. El conjunto de reglas actual se encuentra en la documentación del proyecto. Como se mencionó enValidación de políticas en el momento de la síntesis, recomendamos que las organizaciones establezcan un método de validación más fiable mediante AWS CloudFormation ganchos.

Para AWS Control Towerlos clientes, estos mismos controles proactivos se pueden implementar en toda la organización. Cuando habilita controles AWS Control Tower proactivos en su AWS Control Tower entorno, los controles pueden detener el despliegue de los recursos no conformes que se desplieguen a través AWS CloudFormation de ellos. Para obtener más información sobre los controles proactivos gestionados y su funcionamiento, consulte la AWS Control Tower documentación.

Es mejor utilizar estos controles AWS CDK agrupados y los controles AWS Control Tower proactivos gestionados juntos. En este escenario, puede configurar este complemento de validación con los mismos controles proactivos que están activos en su entorno de AWS Control Tower nube. De este modo, podrá confiar rápidamente en que su AWS CDK aplicación superará los AWS Control Tower controles ejecutándola de `cdk synth` forma local.

Informe de validación

Al sintetizar la AWS CDK aplicación, se llamarán los complementos de validación y se imprimirán los resultados. A continuación se muestra un ejemplo de informe.

```
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
```

De forma predeterminada, el informe se imprimirá en un formato legible para las personas. Si desea un informe en formato JSON, habilítelo `@aws-cdk/core:validationReportJson` mediante la CLI o pasándolo directamente a la aplicación:

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

Como alternativa, puede establecer este par clave-valor de contexto utilizando los `cdk.context.json` archivos `cdk.json` o del directorio de su proyecto (consulte). Contexto del tiempo de ejecución

Si elige el formato JSON, AWS CDK se imprimirá el informe de validación de políticas en un archivo llamado directorio de ensamblaje `policy-validation-report.json` en la nube. En el formato predeterminado, legible por humanos, el informe se imprimirá en la salida estándar.

Para los autores de complementos

Complementos

El marco AWS CDK principal es responsable de registrar e invocar los complementos y, a continuación, mostrar el informe de validación formateado. La responsabilidad del complemento es actuar como capa de traducción entre el AWS CDK marco y la herramienta de validación de políticas. Se puede crear un complemento en cualquier idioma compatible con AWS CDK. Si vas a crear un plugin que pueda ser utilizado en varios idiomas, te recomendamos que lo crees de `TypeScript` forma que puedas usar JSII para publicarlo en cada AWS CDK idioma.

Crear complementos

La `IPolicyValidationPluginBeta1` interfaz define el protocolo de comunicación entre el módulo AWS CDK principal y la herramienta de políticas. Para crear un nuevo complemento, debe escribir una clase que implemente esta interfaz. Hay dos cosas que debes implementar: el nombre del complemento (anulando la `name` propiedad) y el `validate()` método.

El framework llamará `validate()` y pasará un `IValidationContextBeta1` objeto. La ubicación de las plantillas a validar viene dada por`templatePaths`. El complemento debería devolver una instancia de`ValidationPluginReportBeta1`. Este objeto representa el informe que recibirá el usuario al final de la síntesis.

```
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',
      }],
    }],
  };
}
```

Tenga en cuenta que los complementos no pueden modificar nada en el ensamblaje de la nube. Cualquier intento de hacerlo provocará un fallo en la síntesis.

Si su complemento depende de una herramienta externa, tenga en cuenta que es posible que algunos desarrolladores aún no tengan esa herramienta instalada en sus estaciones de trabajo. Para minimizar la fricción, te recomendamos encarecidamente que incluyas un script de instalación junto con tu paquete de complementos, para automatizar todo el proceso. Mejor aún, ejecuta ese script como parte de la instalación de tu paquete. Con`npm`, por ejemplo, puede añadirlo al `postinstall` script del `package.json` archivo.

Exenciones de manejo

Si su organización tiene un mecanismo para gestionar las exenciones, puede implementarlo como parte del complemento de validación.

Un ejemplo de escenario para ilustrar un posible mecanismo de exención:

- Una organización tiene una norma según la cual no se permiten los depósitos públicos de Amazon S3, excepto en determinadas situaciones.
- Un desarrollador está creando un bucket de Amazon S3 que se encuadra en uno de esos escenarios y solicita una exención (por ejemplo, crear un ticket).
- Las herramientas de seguridad saben cómo leer el sistema interno que registra las exenciones

En este escenario, el desarrollador solicitaría una excepción en el sistema interno y luego necesitaría alguna forma de «registrar» esa excepción. Si añadimos el ejemplo del complemento Guard, podrías crear un complemento que gestione las exenciones filtrando las infracciones que cuenten con una exención equivalente en un sistema interno de venta de entradas.

Consulta los complementos existentes para ver ejemplos de implementaciones.

- @cdklabs/cdk-validator-cfnguard

Integración y entrega continuas (CI/CD) mediante CDK Pipelines

Utilice el módulo CDK Pipelines de la biblioteca Construct para configurar AWS la entrega continua de aplicaciones. AWS CDK Cuando depositas el código fuente de tu aplicación de CDK en AWS CodeCommit, o GitHub AWS CodeStar, CDK Pipelines puede compilar, probar e implementar automáticamente tu nueva versión.

Los CDK Pipelines se actualizan automáticamente. Si agrega etapas o pilas de aplicaciones, la canalización se reconfigura automáticamente para implementar esas nuevas etapas o pilas.

> **Note**
>
> CDK Pipelines admite dos API. Una es la API original que estaba disponible en la versión preliminar para desarrolladores de CDK Pipelines. La otra es una API moderna que incorpora los comentarios de los clientes de CDK recibidos durante la fase de vista previa. Los ejemplos de este tema utilizan la API moderna. Para obtener más información sobre las diferencias entre las dos API compatibles, consulta la API original de CDK Pipelines en el repositorio aws-cdk. GitHub

Temas

- Inicie sus AWS entornos
- Inicialice un proyecto
- Defina una canalización
- Etapas de aplicación
- Probar despliegues
- Notas de seguridad
- Solución de problemas

Inicie sus AWS entornos

Antes de poder usar CDK Pipelines, debe iniciar AWS el entorno en el que desplegará sus stacks.

Una canalización de CDK incluye al menos dos entornos. El primer entorno es donde se aprovisiona la canalización. El segundo entorno es donde desea implementar las pilas o etapas de la aplicación (las etapas son grupos de pilas relacionadas). Estos entornos pueden ser los mismos, pero una práctica recomendada es aislar las etapas unas de otras en entornos diferentes.

> **Note**
>
> Consulte the section called “Bootstrapping (Proceso de arranque)” para obtener más información sobre los tipos de recursos que se crean mediante el arranque y sobre cómo personalizar la pila de arranque.

El despliegue continuo con CDK Pipelines requiere que se incluya lo siguiente en la pila de CDK Toolkit:

- Un bucket de Amazon Simple Storage Service (Amazon S3)
- Un repositorio de Amazon ECR.
- Funciones de IAM para conceder a las distintas partes de una canalización los permisos que necesitan.

El kit de herramientas CDK actualizará tu pila de bootstrap existente o creará una nueva si es necesario.

Para iniciar un entorno que pueda aprovisionar una AWS CDK canalización, invoque `cdk bootstrap` como se muestra en el siguiente ejemplo. Si se invoca el AWS CDK kit de herramientas mediante el `npx` comando, se instala temporalmente si es necesario. También utilizará la versión del kit de herramientas instalada en el proyecto actual, si existe.

--cloudformation-execution-policiesespecifica el ARN de una política según la cual se ejecutarán los despliegues de CDK Pipelines en el futuro. La `AdministratorAccess` política predeterminada garantiza que su canalización pueda implementar todos los tipos de recursos. AWS Si utilizas esta política, asegúrate de confiar en todo el código y las dependencias que componen tu AWS CDK aplicación.

La mayoría de las organizaciones exigen controles más estrictos sobre los tipos de recursos que se pueden implementar mediante la automatización. Consulta con el departamento correspondiente de tu organización para determinar la política que debe utilizar tu canal.

Puede omitir --profile esta opción si su AWS perfil predeterminado contiene la configuración de autenticación necesaria y Región de AWS.

macOS/Linux

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE \
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
```

Windows

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE ^
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
```

Para iniciar entornos adicionales en los que la canalización desplegará AWS CDK las aplicaciones, utilice los siguientes comandos en su lugar. La --trust opción indica qué otra cuenta debe tener permisos para implementar AWS CDK aplicaciones en este entorno. Para esta opción, especifique el ID de AWS cuenta de la canalización.

De nuevo, puedes omitir la --profile opción si tu AWS perfil predeterminado contiene la configuración de autenticación necesaria y Región de 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**
>
> Utilice las credenciales administrativas únicamente para iniciar y aprovisionar la canalización inicial. Después, utilice la propia canalización, no su máquina local, para implementar los cambios.

Si está actualizando un entorno de arranque heredado, el depósito anterior de Amazon S3 quedará huérfano cuando se cree el nuevo depósito. Elimínelo manualmente mediante la consola Amazon S3.

Inicialice un proyecto

Cree un GitHub proyecto nuevo y vacío y clónelo en su estación de trabajo en el `my-pipeline` directorio. (Nuestros ejemplos de código de este tema utilizan GitHub. También puedes usar AWS CodeStar o AWS CodeCommit.)

```
git clone GITHUB-CLONE-URL my-pipeline
cd my-pipeline
```

> **Note**
>
> Puedes usar un nombre que no sea `my-pipeline` el del directorio principal de la aplicación. Sin embargo, si lo hace, tendrá que modificar los nombres de los archivos y las clases más adelante en este tema. Esto se debe a que el AWS CDK kit de herramientas basa algunos nombres de archivos y clases en el nombre del directorio principal.

Tras la clonación, inicialice el proyecto como de costumbre.

TypeScript

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

JavaScript

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

Python

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

Una vez creada la aplicación, introduzca también los dos comandos siguientes. Estos activan el entorno virtual Python de la aplicación e instalan las dependencias AWS CDK principales.

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

Java

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

Si está utilizando un IDE, ahora puede abrir o importar el proyecto. En Eclipse, por ejemplo, elija Archivo > Importar > Maven > Proyectos Maven existentes. Asegúrese de que los ajustes del proyecto estén configurados para utilizar Java 8 (1.8).

C#

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

Si utiliza Visual Studio, abra el archivo de la solución en el `src` directorio.

Go

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

Una vez creada la aplicación, introduzca también el siguiente comando para instalar los módulos de AWS Construct Library que requiere la aplicación.

```
go get
```

Important

Asegúrese de asignar sus archivos `cdk.json` y `cdk.context.json` archivos al control de código fuente. La información contextual (como los indicadores de características y los valores en caché recuperados de tu AWS cuenta) forma parte del estado de tu proyecto. Los

valores pueden ser diferentes en otro entorno, lo que puede provocar cambios inesperados en los resultados. Para obtener más información, consulte the section called “Context”.

Defina una canalización

Su aplicación CDK Pipelines incluirá al menos dos pilas: una que represente la propia canalización y una o más pilas que representen la aplicación desplegada a través de ella. Las pilas también se pueden agrupar en etapas, que puede utilizar para implementar copias de las pilas de infraestructura en diferentes entornos. Por ahora, consideraremos la canalización y, más adelante, profundizaremos en la aplicación que se implementará.

La construcción `CodePipeline`es la construcción que representa una canalización de CDK que se utiliza AWS CodePipeline como motor de despliegue. Al crear una instancia `CodePipeline` en una pila, se define la ubicación de origen de la canalización (por ejemplo, un GitHub repositorio). También defines los comandos para crear la aplicación.

Por ejemplo, lo siguiente define una canalización cuya fuente se almacena en un GitHub repositorio. También incluye un paso de creación de una aplicación de TypeScript CDK. Complete la información sobre su GitHub repositorio donde se indica.

> **Note**
>
> De forma predeterminada, la canalización se autentica GitHub con un token de acceso personal almacenado en Secrets Manager con ese nombre`github-token`.

También tendrás que actualizar la instanciación de la pila de canalizaciones para especificar la AWS cuenta y la región.

TypeScript

En `lib/my-pipeline-stack.ts` (puede variar si no se nombra `my-pipeline` la carpeta de tu proyecto):

```
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']
      })
    });
  }
}
```

En `bin/my-pipeline.ts` (puede variar si la carpeta de tu proyecto no tiene nombre`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

En `lib/my-pipeline-stack.js` (puede variar si la carpeta de tu proyecto no tiene nombre`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 }
```

En `bin/my-pipeline.js` (puede variar si la carpeta de tu proyecto no tiene nombre`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

En `my-pipeline/my-pipeline-stack.py` (puede variar si la carpeta de tu proyecto no tiene nombre`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"]
                        )
                    )
```

En `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

En `src/main/java/com/myorg/MyPipelineStack.java` (puede variar si la carpeta de tu proyecto no tiene nombre`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();
    }
}
```

En `src/main/java/com/myorg/MyPipelineApp.java` (puede variar si la carpeta de tu proyecto no tiene nombre`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#

En `src/MyPipeline/MyPipelineStack.cs` (puede variar si la carpeta de tu proyecto no tiene nombre`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" }
                })
            });
        }
    }
}
```

En `src/MyPipeline/Program.cs` (puede variar si la carpeta de tu proyecto no tiene nombre`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();
```

```
        }
    }
}
```

Debes implementar una canalización manualmente una vez. Después de eso, la canalización se mantiene actualizada desde el repositorio de código fuente. Así que asegúrate de que el código del repositorio sea el código que deseas implementar. Registra los cambios, presiona para GitHub, luego, despliega:

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

> **Tip**
>
> Ahora que ha realizado la implementación inicial, su AWS cuenta local ya no necesita acceso administrativo. Esto se debe a que todos los cambios en tu aplicación se implementarán a través de la canalización. Lo único que tienes que hacer es presionar para GitHub.

Etapas de aplicación

Para definir una AWS aplicación de varias pilas que se pueda añadir a la canalización de una sola vez, defina una subclase de. `Stage` (Esto es diferente al del `CdkStage` módulo CDK Pipelines).

La etapa contiene las pilas que componen la aplicación. Si hay dependencias entre las pilas, las pilas se añaden automáticamente a la canalización en el orden correcto. Las pilas que no dependen unas de otras se despliegan en paralelo. Puede añadir una relación de dependencia entre las pilas mediante una llamada. `stack1.addDependency(stack2)`

Las etapas aceptan un `env` argumento predeterminado, que se convierte en el entorno predeterminado para las pilas que contiene. (Las pilas aún pueden tener su propio entorno especificado).

Para añadir una aplicación a la canalización, se llama `addStage()` con instancias de `Stage`. Se puede crear una instancia de una etapa y añadirla a la canalización varias veces para definir las distintas etapas de la canalización de aplicaciones de DTAP o multirregional.

Crearemos una pila que contenga una función Lambda simple y la colocaremos en una etapa. A continuación, añadiremos la etapa a la canalización para que se pueda implementar.

TypeScript

Cree el nuevo archivo `lib/my-pipeline-lambda-stack.ts` para almacenar nuestra pila de aplicaciones que contiene una función 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 el nuevo archivo `lib/my-pipeline-app-stage.ts` para albergar nuestro escenario.

```
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');
    }
}
```

Edita `lib/my-pipeline-stack.ts` para añadir el escenario a nuestra canalización.

```
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

Cree el nuevo archivo `lib/my-pipeline-lambda-stack.js` para almacenar nuestra pila de aplicaciones que contiene una función 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 el nuevo archivo `lib/my-pipeline-app-stage.js` para albergar nuestro escenario.

```
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 };
```

Edita `lib/my-pipeline-stack.ts` para añadir el escenario a nuestra canalización.

```
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

Cree el nuevo archivo `my_pipeline/my_pipeline_lambda_stack.py` para almacenar nuestra pila de aplicaciones que contiene una función 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 el nuevo archivo `my_pipeline/my_pipeline_app_stage.py` para albergar nuestro escenario.

```
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")
```

Edita `my_pipeline/my-pipeline-stack.py` para añadir el escenario a nuestra canalización.

```
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

Cree el nuevo archivo `src/main/java/com.myorg/MyPipelineLambdaStack.java` para almacenar nuestra pila de aplicaciones que contiene una función 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 el nuevo archivo `src/main/java/com.myorg/MyPipelineAppStage.java` para albergar nuestro escenario.

```
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");
    }

}
```

Edita `src/main/java/com.myorg/MyPipelineStack.java` para añadir el escenario a nuestra canalización.

```
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#

Cree el nuevo archivo `src/MyPipeline/MyPipelineLambdaStack.cs` para almacenar nuestra pila de aplicaciones que contiene una función 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 el nuevo archivo `src/MyPipeline/MyPipelineAppStage.cs` para albergar nuestro escenario.

```
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");
        }
    }
}
```

Edita `src/MyPipeline/MyPipelineStack.cs` para añadir el escenario a nuestra canalización.

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.Pipelines;

namespace MyPipeline
{
    public class MyPipelineStack : Stack
    {
        internal MyPipelineStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
```

```
            var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
            {
                PipelineName = "MyPipeline",
                Synth = new ShellStep("Synth", new ShellStepProps
                {
                    Input = CodePipelineSource.GitHub("OWNER/REPO", "main"),
                    Commands = new string[] { "npm install -g aws-cdk", "cdk
 synth" }
                })
            });

            pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps
            {
                Env = new Environment
                {
                    Account = "111111111111", Region = "eu-west-1"
                }
            }));
        }
    }
}
```

Cada etapa de la aplicación que `addStage()` añada implica la adición de una etapa de canalización correspondiente, representada por una StageDeploymentinstancia devuelta por la `addStage()` llamada. Puede añadir acciones previas o posteriores a la implementación a la etapa llamando a su método `addPre()` o`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"));
```

Puede añadir etapas a una ola para desplegarlas en paralelo, por ejemplo, al implementar una etapa en varias cuentas o regiones.

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"
    }
}));
```

Probar despliegues

Puede añadir pasos a una canalización de CDK para validar las implementaciones que está realizando. Por ejemplo, puede utilizar la biblioteca CDK Pipeline `ShellStep` para realizar tareas como las siguientes:

- Intentando acceder a una Amazon API Gateway recién implementada y respaldada por una función Lambda
- Comprobar la configuración de un recurso desplegado mediante la emisión de un comando AWS CLI

En su forma más simple, añadir acciones de validación tiene el siguiente aspecto:

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'" }
}));
```

Muchas AWS CloudFormation implementaciones dan como resultado la generación de recursos con nombres impredecibles. Por ello, los CDK Pipelines proporcionan una forma de AWS

CloudFormation leer los resultados después de una implementación. Esto permite pasar (por ejemplo) la URL generada de un balanceador de cargas a una acción de prueba.

Para usar los resultados, muestra el `CfnOutput` objeto que te interesa. A continuación, páselo a la `envFromCfnOutputs` propiedad de un paso para que esté disponible como variable de entorno dentro de ese paso.

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" }
}));
```

Puede escribir pruebas de validación sencillas directamente en el`ShellStep`, pero este enfoque se vuelve difícil de manejar cuando la prueba consta de más de unas pocas líneas. Para pruebas más complejas, puede incorporar archivos adicionales (como scripts de shell completos o programas en otros lenguajes) a `ShellStep` través de la propiedad. `inputs` Las entradas pueden ser cualquier paso que tenga una salida, incluida una fuente (como un GitHub repositorio) u otra`ShellStep`.

Incorporar archivos del repositorio fuente es adecuado si los archivos se pueden utilizar directamente en la prueba (por ejemplo, si ellos mismos son ejecutables). En este ejemplo, declaramos nuestro

GitHub repositorio como `source` (en lugar de instanciarlo en línea como parte del). `CodePipeline` Luego, pasamos este conjunto de archivos tanto a la canalización como a la prueba de validación.

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" }
}));
```

Obtener los archivos adicionales del paso de sintetizador es adecuado si es necesario compilar las pruebas, lo cual se hace como parte de la síntesis.

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" }
}));
```

Notas de seguridad

Cualquier forma de entrega continua conlleva riesgos de seguridad inherentes. Según el modelo de responsabilidad AWS compartida, usted es responsable de la seguridad de su información en la AWS nube. La biblioteca CDK Pipelines le ofrece una ventaja inicial al incorporar valores predeterminados seguros y mejores prácticas de modelado.

Sin embargo, por su propia naturaleza, una biblioteca que necesita un alto nivel de acceso para cumplir con el propósito previsto no puede garantizar una seguridad completa. Existen muchos vectores de ataque fuera AWS de su organización.

En particular, tenga en cuenta lo siguiente:

- Tenga en cuenta el software del que depende. Examina todo el software de terceros que utilices en proceso, ya que puede cambiar la infraestructura que se implementa.

- Usa el bloqueo de dependencias para evitar actualizaciones accidentales. CDK Pipelines `package-lock.json` respeta `yarn.lock` y se asegura de que sus dependencias sean las que usted espera.
- CDK Pipelines se ejecuta con recursos creados en tu propia cuenta, y la configuración de esos recursos la controlan los desarrolladores que envían el código a través de la canalización. Por lo tanto, CDK Pipelines por sí solo no puede proteger a los desarrolladores malintencionados que intentan eludir las comprobaciones de conformidad. Si su modelo de amenazas incluye desarrolladores que escriben código CDK, debe contar con mecanismos de cumplimiento externos, como AWS CloudFormation Hooks (preventivo) o AWS Config (reactivo), que el rol de AWS CloudFormation ejecución no tenga permiso para deshabilitar.
- Las credenciales para los entornos de producción deben ser efímeras. Tras el arranque y el aprovisionamiento inicial, no es necesario que los desarrolladores tengan credenciales de cuenta en absoluto. Los cambios se pueden implementar a través de la canalización. Reduzca la posibilidad de que las credenciales se filtren al no necesitarlas en primer lugar.

Solución de problemas

Los siguientes problemas suelen surgir al empezar a usar CDK Pipelines.

Tubería: fallo interno

```
CREATE_FAILED  | AWS::CodePipeline::Pipeline | Pipeline/Pipeline
Internal Failure
```

Comprueba tu token de GitHub acceso. Puede que falte o que no tenga los permisos para acceder al repositorio.

Clave: la política contiene una declaración con uno o más principios no válidos

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey
Policy contains a statement with one or more invalid principals.
```

Uno de los entornos de destino no se ha iniciado con la nueva pila de arranque. Asegúrese de que todos los entornos de destino estén iniciados.

La pila está en estado ROLLBACK_COMPLETE y no se puede actualizar.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
```

```
ID: ...)
```

La pila falló en su implementación anterior y no se puede volver a intentar. Elimine la pila de la AWS CloudFormation consola y vuelva a intentar la implementación.

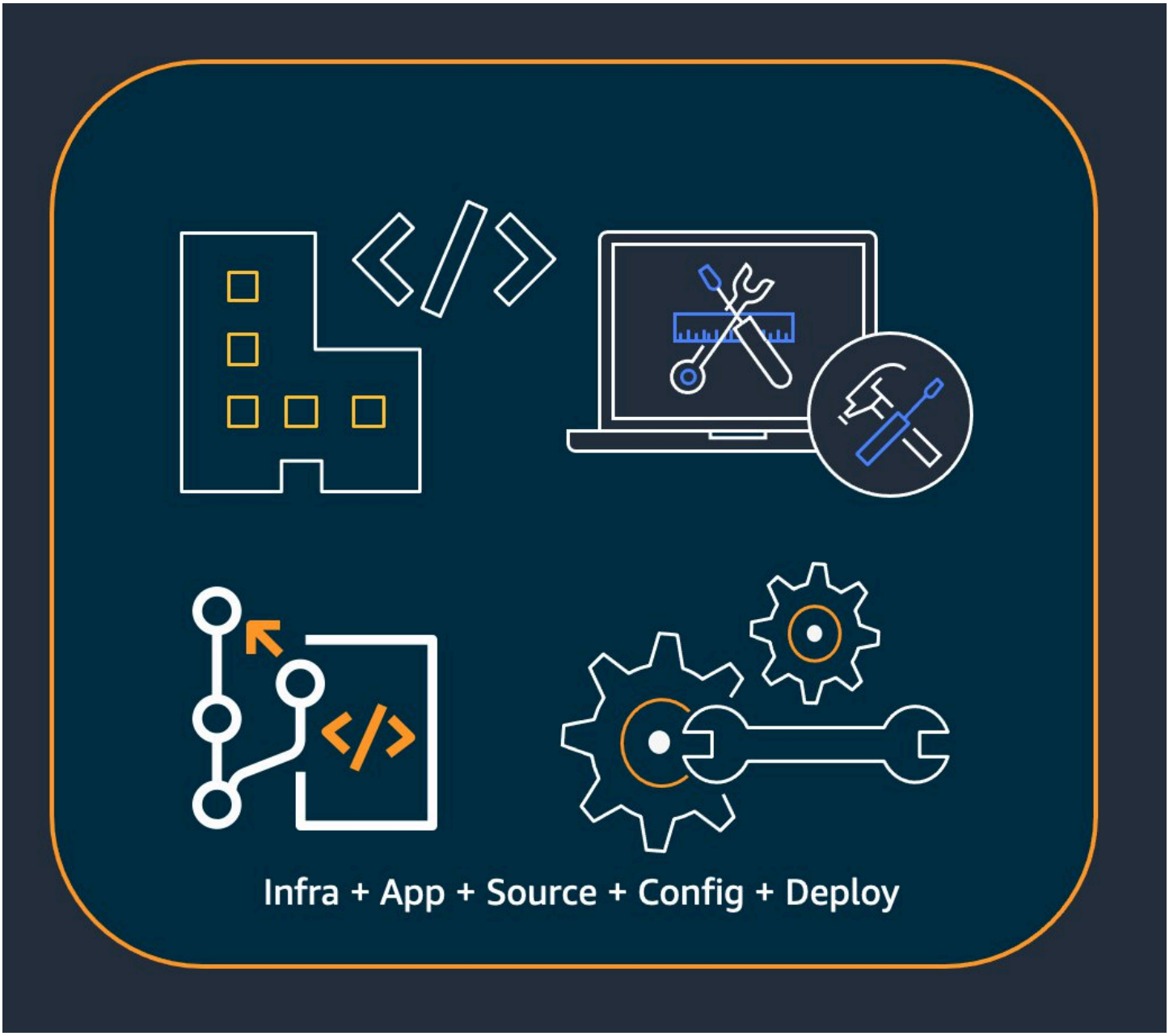
Mejores prácticas para desarrollar e implementar una infraestructura de nube con AWS CDK

Con él AWS CDK, los desarrolladores o administradores pueden definir su infraestructura de nube mediante un lenguaje de programación compatible. Las aplicaciones de CDK deben organizarse en unidades lógicas, como la API, la base de datos y los recursos de monitoreo, y, opcionalmente, tener una canalización para las implementaciones automatizadas. Las unidades lógicas deben implementarse como construcciones que incluyan las siguientes:

- Infraestructura (como buckets de Amazon S3, bases de datos de Amazon RDS o una red de Amazon VPC)
- Código de tiempo de ejecución (como funciones) AWS Lambda
- Código de configuración

Las pilas definen el modelo de despliegue de estas unidades lógicas. Para obtener una introducción más detallada a los conceptos en los que se basa el CDK, consulte. Introducción

Esto AWS CDK refleja una cuidadosa consideración de las necesidades de nuestros clientes y equipos internos y de los patrones de falla que suelen surgir durante el despliegue y el mantenimiento continuo de aplicaciones en la nube complejas. Descubrimos que los fallos suelen estar relacionados con «out-of-band» cambios en una aplicación que no se han probado completamente, como los cambios de configuración. Por lo tanto, lo desarrollamos en AWS CDK torno a un modelo en el que toda la aplicación se define en código, no solo la lógica empresarial, sino también la infraestructura y la configuración. De esta forma, los cambios propuestos pueden revisarse detenidamente, probarse exhaustivamente en entornos que se asemejen en mayor o menor medida a los de producción y revertirse por completo si algo sale mal.



En el momento de la implementación, AWS CDK sintetiza un conjunto de nubes que contiene lo siguiente:

- AWS CloudFormation plantillas que describen su infraestructura en todos los entornos de destino
- Activos de archivos que contienen su código de ejecución y sus archivos auxiliares

Con la CDK, cada confirmación realizada en la rama principal de control de versiones de la aplicación puede representar una versión completa, coherente y desplegable de la aplicación. De este modo, la aplicación se puede implementar automáticamente cada vez que se realice un cambio.

La filosofía en la que se basa nos AWS CDK lleva a nuestras mejores prácticas recomendadas, que hemos dividido en cuatro amplias categorías.

- the section called “Prácticas recomendadas de organizaciones”
- the section called “Mejores prácticas de codificación”
- the section called “Diseñe las mejores prácticas”
- the section called “Mejores prácticas de aplicación”

> **Tip**
>
> Tenga en cuenta también las mejores prácticas AWS CloudFormation y los AWS servicios individuales que utilice, cuando proceda a la infraestructura definida por el CDK.

Prácticas recomendadas de organizaciones

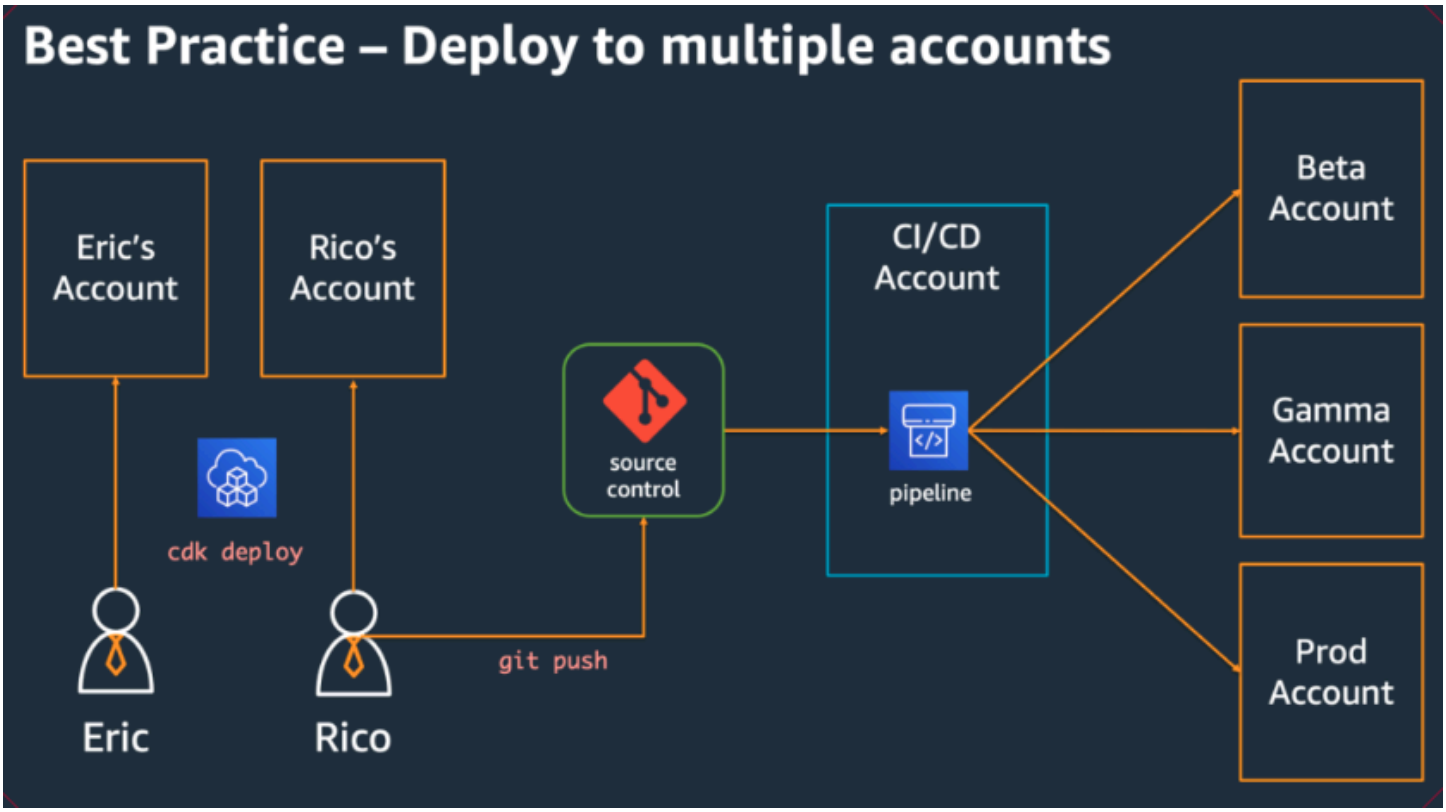
En las etapas iniciales de AWS CDK la adopción, es importante considerar cómo configurar su organización para el éxito. Se recomienda contar con un equipo de expertos que se encargue de formar y guiar al resto de la empresa a medida que adopte la CDK. El tamaño de este equipo puede variar, desde una o dos personas en una empresa pequeña hasta un Cloud Center of Excellence (CCoE) completo en una empresa más grande. Este equipo es responsable de establecer normas y políticas para la infraestructura de nube de su empresa, y también de formar y asesorar a los desarrolladores.

El CCoE podría proporcionar orientación sobre los lenguajes de programación que se deben usar para la infraestructura de nube. Los detalles varían de una organización a otra, pero una buena política ayuda a garantizar que los desarrolladores puedan entender y mantener la infraestructura de nube de la empresa.

El CCoE también crea una «landing zone» que define las unidades organizativas en las que se encuentran AWS. Una landing zone es un AWS entorno multicuenta preconfigurado, seguro, escalable y basado en planes de mejores prácticas. Para unir los servicios que componen tu landing zone, puedes usar AWS Control Tower, que configura y administra todo tu sistema de cuentas múltiples desde una única interfaz de usuario.

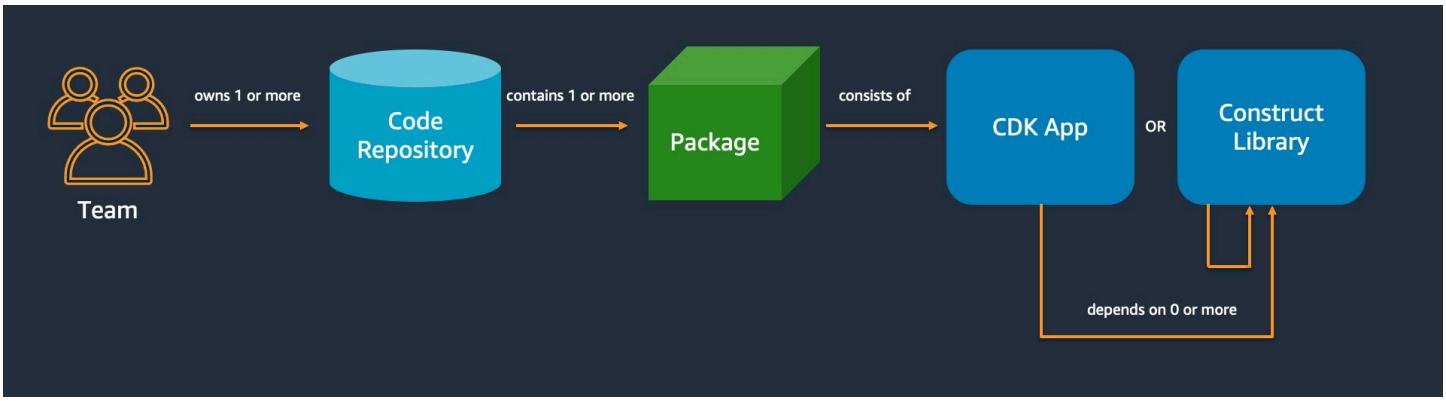
Los equipos de desarrollo deberían poder usar sus propias cuentas para realizar pruebas e implementar nuevos recursos en estas cuentas según sea necesario. Los desarrolladores

individuales pueden tratar estos recursos como extensiones de su propia estación de trabajo de desarrollo. Con CDK Pipelines, AWS CDK las aplicaciones se pueden implementar a través de una cuenta de CI/CD en entornos de prueba, integración y producción (cada uno aislado en su AWS propia región o cuenta). Para ello, se fusiona el código de los desarrolladores en el repositorio canónico de la organización.



Mejores prácticas de codificación

En esta sección se presentan las prácticas recomendadas para organizar el AWS CDK código. El siguiente diagrama muestra la relación entre un equipo y los repositorios de código, paquetes, aplicaciones y bibliotecas de construcción de ese equipo.



Comience de forma sencilla y añada complejidad solo cuando la necesite

El principio rector de la mayoría de nuestras mejores prácticas es mantener las cosas lo más sencillas posible, pero no más sencillas. Añada complejidad solo cuando sus requisitos exijan una solución más complicada. Con ella AWS CDK, puede refactorizar el código según sea necesario para adaptarlo a los nuevos requisitos. No es necesario que diseñe por adelantado todos los escenarios posibles.

Alinéese con el marco de AWS Well-Architected

El AWS Well-Architected Framework define un componente como el código, la configuración AWS y los recursos que, en conjunto, cumplen con un requisito. Un componente suele ser la unidad de propiedad técnica y está disociado de otros componentes. El término carga de trabajo se utiliza para identificar un conjunto de componentes que, en conjunto, aportan valor empresarial. Una carga de trabajo suele ser el nivel de detalle sobre el que se comunican los líderes empresariales y tecnológicos.

Una AWS CDK aplicación se asigna a un componente según lo definido por el AWS Well-Architected Framework. AWS CDK las aplicaciones son un mecanismo para codificar y ofrecer las mejores prácticas de aplicaciones en la nube de Well-Architected. También puede crear y compartir componentes como bibliotecas de códigos reutilizables a través de repositorios de artefactos, como. AWS CodeArtifact

Cada aplicación comienza con un único paquete en un único repositorio

Un solo paquete es el punto de entrada de tu AWS CDK aplicación. Aquí, usted define cómo y dónde implementar las diferentes unidades lógicas de su aplicación. También debe definir la canalización de CI/CD para implementar la aplicación. Las estructuras de la aplicación definen las unidades lógicas de la solución.

Utilice paquetes adicionales para las construcciones que utilice en más de una aplicación. (Las construcciones compartidas también deben tener su propio ciclo de vida y estrategia de pruebas). Las dependencias entre los paquetes del mismo repositorio se gestionan mediante las herramientas de compilación de su repositorio.

Si bien es posible, no recomendamos colocar varias aplicaciones en el mismo repositorio, especialmente cuando se utilizan canalizaciones de despliegue automatizadas. De este modo, se aumenta el «radio de expansión» de los cambios durante el despliegue. Cuando hay varias

aplicaciones en un repositorio, los cambios en una aplicación desencadenan el despliegue de las demás (incluso si las demás no han cambiado). Además, una interrupción en una aplicación impide que se desplieguen las demás.

Mueva el código a los repositorios en función del ciclo de vida del código o de la propiedad del equipo

Cuando los paquetes comiencen a usarse en varias aplicaciones, muévalos a su propio repositorio. De esta forma, los sistemas de creación de aplicaciones que los utilizan pueden hacer referencia a los paquetes y también se pueden actualizar en cadencias independientes de los ciclos de vida de las aplicaciones. Sin embargo, al principio podría tener sentido colocar todas las construcciones compartidas en un repositorio.

Además, mueva los paquetes a su propio repositorio cuando diferentes equipos estén trabajando en ellos. Esto ayuda a reforzar el control de acceso.

Para consumir paquetes más allá de los límites del repositorio, necesitas un repositorio de paquetes privado, similar a NPM o Maven Central PyPi, pero interno en tu organización. También necesitas un proceso de publicación que compile, pruebe y publique el paquete en el repositorio de paquetes privado. CodeArtifactpuede alojar paquetes para los lenguajes de programación más populares.

Las dependencias de los paquetes del repositorio de paquetes las gestiona el administrador de paquetes de su idioma, como NPM for TypeScript or JavaScript applications. El administrador de paquetes ayuda a garantizar que las compilaciones sean repetibles. Para ello, graba las versiones específicas de cada paquete del que depende tu aplicación. También le permite actualizar esas dependencias de forma controlada.

Los paquetes compartidos necesitan una estrategia de prueba diferente. En el caso de una sola aplicación, podría bastar con desplegarla en un entorno de pruebas y confirmar que sigue funcionando. Sin embargo, los paquetes compartidos deben probarse independientemente de la aplicación que los consuma, como si se estuvieran lanzando al público. (Es posible que su organización opte por lanzar al público algunos paquetes compartidos).

Tenga en cuenta que una construcción puede ser arbitrariamente simple o compleja. A `Bucket` es una construcción, pero también `CameraShopWebsite` podría ser una construcción.

La infraestructura y el código de ejecución se encuentran en el mismo paquete

Además de generar AWS CloudFormation plantillas para implementar la infraestructura, AWS CDK también incluye activos de tiempo de ejecución, como funciones Lambda e imágenes de Docker, y los implementa junto con su infraestructura. Esto permite combinar el código que define la infraestructura y el código que implementa la lógica de tiempo de ejecución en una sola construcción. Hacer esto es una buena práctica. No es necesario que estos dos tipos de código se encuentren en repositorios separados o incluso en paquetes separados.

Para desarrollar los dos tipos de código juntos, puedes usar una construcción independiente que describa por completo una parte de la funcionalidad, incluidas su infraestructura y lógica. Con una construcción independiente, puedes probar los dos tipos de código de forma aislada, compartir y reutilizar el código en todos los proyectos y versionar todo el código de forma sincronizada.

Diseñe las mejores prácticas

Esta sección contiene las mejores prácticas para desarrollar construcciones. Las construcciones son módulos reutilizables y componibles que encapsulan los recursos. Son los componentes básicos de las aplicaciones. AWS CDK

Modele con componentes fijos e impleméntelo con pilas

Las pilas son la unidad de despliegue: todos los elementos de una pila se despliegan juntos. Por lo tanto, cuando cree las unidades lógicas de nivel superior de su aplicación a partir de varios AWS recursos, represente cada unidad lógica como un `Construct`, no como un. `Stack` Utilice las pilas únicamente para describir cómo deben componerse y conectarse las estructuras en los distintos escenarios de implementación.

Por ejemplo, si una de sus unidades lógicas es un sitio web, las construcciones que lo componen (como un bucket de Amazon S3, API Gateway, funciones de Lambda o tablas de Amazon RDS) deben estar compuestas en una única construcción de alto nivel. Luego, esa construcción se debe instanciar en una o más pilas para su implementación.

Al utilizar componentes constructivos para la construcción y pilas para la implementación, usted mejora el potencial de reutilización de su infraestructura y obtiene más flexibilidad a la hora de implementarla.

Configure con propiedades y métodos, no con variables de entorno

Las búsquedas de variables de entorno dentro de las construcciones y las pilas son un antipatrón común. Tanto las construcciones como las pilas deben aceptar un objeto de propiedades para permitir una completa configurabilidad en el código. De lo contrario, se crea una dependencia de la máquina en la que se ejecutará el código, lo que crea aún más información de configuración de la que hay que hacer un seguimiento y administrar.

En general, las búsquedas de variables de entorno deben limitarse al nivel superior de una AWS CDK aplicación. También se deben usar para transmitir la información necesaria para ejecutarse en un entorno de desarrollo. Para obtener más información, consulte the section called “Entornos”.

Realice pruebas unitarias de su infraestructura

Para ejecutar de forma coherente un conjunto completo de pruebas unitarias en el momento de la compilación en todos los entornos, evite las búsquedas en la red durante la síntesis y modele todas las etapas de producción en código. (Estas prácticas recomendadas se describen más adelante). Si una sola confirmación siempre da como resultado la misma plantilla generada, puedes confiar en las pruebas unitarias que escribas para confirmar que las plantillas generadas tienen el aspecto esperado. Para obtener más información, consulte Comprobación de construcciones.

No cambies el ID lógico de los recursos con estado

Si se cambia el identificador lógico de un recurso, el recurso se sustituirá por uno nuevo en la siguiente implementación. En el caso de los recursos con estado, como las bases de datos y los buckets de S3, o de infraestructuras persistentes, como una Amazon VPC, esto rara vez es lo que se busca. Tenga cuidado con cualquier refactorización del AWS CDK código que pueda provocar un cambio en la ID. Escribe pruebas unitarias que confirmen que los ID lógicos de tus recursos con estado permanecen estáticos. El identificador lógico se deriva del `id` que especifiques al crear una instancia del componente fijo y de la posición del componente fijo en el árbol de componentes fijos. Para obtener más información, consulte the section called “Identificadores lógicos”.

Los componentes fijos no son suficientes para garantizar la conformidad

Muchos clientes empresariales diseñan sus propios contenedores para las construcciones de nivel 2 (las construcciones «seleccionadas» que representan AWS recursos individuales e incorporan prácticas recomendadas y predeterminadas). Estos contenedores aplican las mejores prácticas de seguridad, como el cifrado estático y políticas de IAM específicas. Por ejemplo, puede crear

una `MyCompanyBucket` que luego utilice en sus aplicaciones en lugar de la `Bucket` construcción habitual de Amazon S3. Este patrón es útil para dar a conocer directrices de seguridad en una fase temprana del ciclo de vida del desarrollo del software, pero no debe basarse en él como único medio de aplicación.

En su lugar, utilice AWS funciones como las políticas de control de servicios y los límites de permisos para reforzar las barreras de seguridad a nivel de la organización. Utilice the section called “Aspectos” herramientas como CloudFormation Guard para hacer afirmaciones sobre las propiedades de seguridad de los elementos de la infraestructura antes del despliegue. Úselo AWS CDK para lo que mejor sabe hacer.

Por último, tenga en cuenta que escribir sus propias construcciones «L2+» puede impedir que sus desarrolladores aprovechen AWS CDK paquetes como AWS Solutions Constructs o construcciones de terceros de Construct Hub. Por lo general, estos paquetes se basan en AWS CDK construcciones estándar y no podrán usar las construcciones de su contenedor.

Mejores prácticas de aplicación

En esta sección, analizamos cómo escribir sus AWS CDK aplicaciones, combinando construcciones para definir cómo están conectados sus AWS recursos.

Tome decisiones en el momento de la síntesis

Si bien AWS CloudFormation le permite tomar decisiones en el momento de la implementación (utilizando `Conditions{ Fn::If }`, `yParameters`) y AWS CDK le da cierto acceso a estos mecanismos, le recomendamos que no los utilice. Los tipos de valores que puede utilizar y los tipos de operaciones que puede realizar con ellos son limitados en comparación con lo que está disponible en un lenguaje de programación de uso general.

En su lugar, intente tomar todas las decisiones, como qué construcción crear una instancia, en la AWS CDK aplicación utilizando las `if` instrucciones del lenguaje de programación y otras funciones. Por ejemplo, una expresión común de CDK, que repite una lista y crea instancias de una construcción con valores de cada elemento de la lista, simplemente no es posible utilizar expresiones. AWS CloudFormation

AWS CloudFormation Tómatelo como un detalle de implementación que se AWS CDK utiliza para despliegues sólidos en la nube, no como un objetivo lingüístico. No estás escribiendo AWS CloudFormation plantillas en TypeScript Python, estás escribiendo código CDK que resulta que se usa CloudFormation para la implementación.

Usa nombres de recursos generados, no nombres físicos

Los nombres son un recurso precioso. Cada nombre solo se puede usar una vez. Por lo tanto, si codificas el nombre de una tabla o un nombre de bucket en tu infraestructura y aplicación, no podrás implementar esa parte de la infraestructura dos veces en la misma cuenta. (El nombre del que hablamos aquí es el nombre especificado, por ejemplo, por la `bucketName` propiedad de una construcción de bucket de Amazon S3).

Lo que es peor, no puede realizar cambios en el recurso que requieran su reemplazo. Si una propiedad solo se puede establecer en el momento de la creación del recurso, como una tabla `KeySchema` de Amazon DynamoDB, entonces esa propiedad es inmutable. Para cambiar esta propiedad se requiere un recurso nuevo. Sin embargo, el nuevo recurso debe tener el mismo nombre para que sea un verdadero sustituto. Sin embargo, no puede tener el mismo nombre mientras el recurso existente siga usando ese nombre.

Un mejor enfoque es especificar el menor número de nombres posible. Si omite los nombres de los recursos, se los AWS CDK generará automáticamente de forma que no le causen problemas. Supongamos que tiene una tabla como recurso. A continuación, puede pasar el nombre de la tabla generada como una variable de entorno a su AWS Lambda función. En su AWS CDK aplicación, puede hacer referencia al nombre de la tabla como`table.tableName`. Como alternativa, puede generar un archivo de configuración en la instancia de Amazon EC2 al iniciarse o escribir el nombre real de la tabla en el almacén de AWS Systems Manager parámetros para que la aplicación pueda leerlo desde allí.

Si el lugar donde lo necesitas es en otra AWS CDK pila, es aún más sencillo. Suponiendo que una pila define el recurso y otra pila necesita usarlo, se aplica lo siguiente:

- Si las dos pilas están en la misma AWS CDK aplicación, pasa una referencia entre las dos pilas. Por ejemplo, guarda una referencia a la construcción del recurso como atributo de la pila que la define ()`this.stack.uploadBucket = myBucket`. A continuación, pasa ese atributo al constructor de la pila que necesita el recurso.
- Cuando las dos pilas estén en AWS CDK aplicaciones diferentes, usa un `from` método estático para usar un recurso definido externamente en función de su ARN, nombre u otros atributos. (Por ejemplo, úselo `Table.fromArn()` para una tabla de DynamoDB). Utilice la `CfnOutput` construcción para imprimir el ARN u otro valor requerido en la salida `decdk deploy`, o busque en el. AWS Management Console Como alternativa, la segunda aplicación puede leer la CloudFormation plantilla generada por la primera aplicación y recuperar ese valor de la `Outputs` sección.

Defina las políticas de eliminación y la retención de registros

Los AWS CDK intentos de evitar que pierda datos mediante políticas que conservan todo lo que usted crea de forma predeterminada. Por ejemplo, la política de eliminación predeterminada de los recursos que contienen datos (como los depósitos de Amazon S3 y las tablas de bases de datos) consiste en no eliminar el recurso cuando se elimina de la pila. En su lugar, el recurso queda huérfano de la pila. Del mismo modo, el valor predeterminado de la CDK es conservar todos los registros para siempre. En los entornos de producción, estos valores predeterminados pueden traducirse rápidamente en el almacenamiento de grandes cantidades de datos que en realidad no se necesitan y en la correspondiente AWS factura.

Considere detenidamente cuáles desea que sean estas políticas para cada recurso de producción y especifíquelas en consecuencia. Utilícelas the section called “Aspectos” para validar las políticas de eliminación y registro de su pila.

Separe la aplicación en varias pilas según lo exijan los requisitos de implementación

No existe una regla estricta sobre el número de pilas que necesita su aplicación. Por lo general, terminarás basando la decisión en tus patrones de implementación. Ten en cuenta las siguientes pautas:

- Por lo general, es más sencillo mantener tantos recursos en la misma pila como sea posible, así que manténgalos juntos a menos que sepa que quiere separarlos.
- Considera la posibilidad de mantener los recursos con estado (como las bases de datos) en una pila separada de los recursos sin estado. A continuación, puede activar la protección de terminación en la pila con estado. De esta forma, puede destruir libremente o crear varias copias de la pila sin estado sin riesgo de perder datos.
- Los recursos con estado son más sensibles a la hora de construir el cambio de nombre: el cambio de nombre implica la sustitución de recursos. Por lo tanto, no coloques los recursos con estado dentro de construcciones que puedan moverse o cambiarse de nombre (a menos que el estado pueda reconstruirse si se pierde, como una memoria caché). Esta es otra buena razón para colocar los recursos con estado en su propia pila.

Comprométete **cdk.context.json** a evitar un comportamiento no determinista

El determinismo es clave para el éxito de las implementaciones. AWS CDK Básicamente, una AWS CDK aplicación debería tener el mismo resultado siempre que se implemente en un entorno determinado.

Como tu AWS CDK aplicación está escrita en un lenguaje de programación de uso general, puede ejecutar código arbitrario, usar bibliotecas arbitrarias y realizar llamadas de red arbitrarias. Por ejemplo, puedes usar un AWS SDK para recuperar cierta información de tu AWS cuenta mientras sintetizas tu aplicación. Ten en cuenta que, al hacerlo, se requerirán requisitos de configuración de credenciales adicionales, se incrementará la latencia y se correrá la posibilidad, por pequeña que sea, de que se produzca un error cada vez que la ejecutes. `cdk synth`

No modifique nunca su AWS cuenta ni sus recursos durante la síntesis. Sintetizar una aplicación no debería tener efectos secundarios. Los cambios en la infraestructura solo deberían producirse en la fase de implementación, una vez que se haya AWS CloudFormation generado la plantilla. De esta forma, si hay algún problema, se AWS CloudFormation puede revertir automáticamente el cambio. Para realizar cambios que no se puedan realizar fácilmente dentro del AWS CDK marco, usa recursos personalizados para ejecutar código arbitrario en el momento de la implementación.

Incluso las llamadas estrictamente de solo lectura no son necesariamente seguras. Tenga en cuenta lo que ocurre si cambia el valor devuelto por una llamada de red. ¿En qué parte de su infraestructura afectará eso? ¿Qué pasará con los recursos ya desplegados? Los siguientes son dos ejemplos de situaciones en las que un cambio repentino en los valores podría provocar un problema.

- Si aprovisiona una Amazon VPC a todas las zonas de disponibilidad disponibles en una región específica y el número de zonas de disponibilidad es de dos el día de la implementación, su espacio IP se divide por la mitad. Si AWS lanza una nueva zona de disponibilidad al día siguiente, la siguiente implementación intentará dividir el espacio IP en tercios, lo que requerirá que se vuelvan a crear todas las subredes. Probablemente esto no sea posible porque sus instancias de Amazon EC2 siguen ejecutándose y tendrá que limpiarlas manualmente.
- Si busca la imagen de máquina Amazon Linux más reciente e implementa una instancia de Amazon EC2 y, al día siguiente, se publica una nueva imagen, una implementación posterior recoge la nueva AMI y reemplaza todas las instancias. Puede que esto no sea lo que esperaba que sucediera.

Estas situaciones pueden ser perniciosas porque el AWS cambio radical puede producirse después de meses o años de implementaciones satisfactorias. De repente, sus despliegues están fallando «sin motivo alguno» y hace tiempo que olvidó lo que hizo y por qué.

Afortunadamente, AWS CDK incluye un mecanismo denominado proveedores de contexto para registrar una instantánea de valores no deterministas. Esto permite que las futuras operaciones de síntesis produzcan exactamente la misma plantilla que cuando se desplegaron por primera vez. Los únicos cambios en la nueva plantilla son los cambios que haya realizado en el código. Cuando utilizas el `.fromLookup()` método de una construcción, el resultado de la llamada se almacena en `cdk.context.json` caché. Deberías confirmarlo con el control de versiones junto con el resto del código para asegurarte de que en las futuras ejecuciones de tu aplicación CDK se utilice el mismo valor. El kit de herramientas CDK incluye comandos para administrar la caché de contexto, de modo que puedas actualizar entradas específicas cuando lo necesites. Para obtener más información, consulte the section called “Context”.

Si necesita algún valor (de AWS o de otro lugar) para el que no haya un proveedor de contexto de CDK nativo, le recomendamos que escriba un script independiente. El script debe recuperar el valor y escribirlo en un archivo y, a continuación, leerlo en la aplicación de CDK. Ejecute el script solo cuando desee actualizar el valor almacenado, no como parte de su proceso de compilación habitual.

Deje que AWS CDK administren las funciones y los grupos de seguridad

Con los `grant()` prácticos métodos de la biblioteca de construcción de CDK de AWS, puede crear AWS Identity and Access Management roles que concedan acceso a un recurso a otro mediante permisos con un alcance mínimo. Por ejemplo, considere una línea como la siguiente:

```
myBucket.grantRead(myLambda)
```

Esta línea única añade una política al rol de la función Lambda (que también se crea para usted). Esa función y sus políticas son más de una docena de líneas CloudFormation que no es necesario escribir. Solo AWS CDK concede los permisos mínimos necesarios para que la función lea el contenido del bucket.

Si se requiere que los desarrolladores utilicen siempre funciones predefinidas creadas por un equipo de seguridad, la AWS CDK codificación se vuelve mucho más complicada. Sus equipos podrían perder mucha flexibilidad a la hora de diseñar sus aplicaciones. Una mejor alternativa es utilizar políticas de control de servicios y límites de permisos para garantizar que los desarrolladores se mantengan dentro de las barreras.

Modele todas las etapas de producción en código

En AWS CloudFormation los escenarios tradicionales, el objetivo es producir un único artefacto parametrizado para que pueda implementarse en varios entornos de destino después de aplicar los valores de configuración específicos de esos entornos. En el CDK, puede y debe crear esa configuración en su código fuente. Cree una pila para su entorno de producción y cree una pila independiente para cada una de las demás etapas. A continuación, coloque los valores de configuración de cada pila en el código. Utilice servicios como Secrets Manager y Systems Manager Parameter Store para los valores confidenciales que no desee registrar en el control de código fuente, utilizando los nombres o los ARN de esos recursos.

Al sintetizar la aplicación, el ensamblaje de nube creado en la `cdk.out` carpeta contiene una plantilla independiente para cada entorno. Toda la compilación es determinista. No hay out-of-band cambios en tu solicitud, y cualquier confirmación determinada siempre arroja exactamente la misma AWS CloudFormation plantilla y los activos correspondientes. Esto hace que las pruebas unitarias sean mucho más fiables.

Mídelo todo

Lograr el objetivo de un despliegue continuo y completo, sin intervención humana, requiere un alto nivel de automatización. Esa automatización solo es posible con una gran cantidad de monitoreo. Para medir todos los aspectos de los recursos desplegados, cree métricas, alarmas y paneles de control. No se limite a medir aspectos como el uso de la CPU y el espacio en disco. Registre también las métricas de su empresa y utilícelas para automatizar las decisiones de implementación, como las reversiones. La mayoría de las construcciones de nivel 2 incluyen métodos AWS CDK prácticos que ayudan a crear métricas, como el `metricUserErrors()` método de la clase Dynamodb.Table.

AWS CDK referencia

Esta sección contiene información de referencia para AWS Cloud Development Kit (AWS CDK).

Temas

Referencia de la API

La referencia de la API contiene información sobre la biblioteca AWS Construct y otras API proporcionadas por AWS Cloud Development Kit (AWS CDK). La mayor parte de la biblioteca AWS Construct está contenida en un solo paquete llamado por su TypeScript nombre:`aws-cdk-lib`. El nombre real del paquete varía según el idioma. Se proporcionan versiones independientes de la referencia de la API para cada lenguaje de programación compatible.

La referencia de la API CDK está organizada en submódulos. Hay uno o más submódulos para cada uno. Servicio de AWS

Cada submódulo tiene una descripción general que incluye información sobre cómo utilizar sus API. Por ejemplo, en la descripción general de S3 se muestra cómo configurar el cifrado predeterminado en un bucket de Amazon Simple Storage Service (Amazon S3).

AWS CDK control de versiones

En este tema se proporciona información de referencia sobre cómo AWS Cloud Development Kit (AWS CDK) gestiona el control de versiones.

Los números de versión constan de tres partes numéricas de la versión: la principal. menor. parche y adhiérase estrictamente al modelo de control de versiones semántico. Esto significa que los cambios más importantes en las API estables se limitan a las versiones principales.

Las versiones menores y los parches son compatibles con versiones anteriores. El código escrito en una versión anterior con la misma versión principal se puede actualizar a una versión más reciente dentro de la misma versión principal. También seguirá compilándose y ejecutándose, produciendo el mismo resultado.

Temas

AWS CDKCLIcompatibilidad

Siempre AWS CDK CLI es compatible con bibliotecas de construcción con un número de versión semánticamente inferior o igual. Por lo tanto, siempre es seguro actualizar la AWS CDK CLI misma versión principal.

No siempre AWS CDK CLI es compatible con bibliotecas de construcción de una versión semánticamente superior. La compatibilidad depende de si los dos componentes utilizan la misma versión del esquema de ensamblaje en la nube. El AWS CDK marco genera un ensamblaje de nube durante la síntesis y lo AWS CDK CLI consume para su implementación. El esquema que define el formato del ensamblaje en la nube está estrictamente especificado y versionado.

AWS Las bibliotecas de construcción que utilizan una versión determinada del esquema de ensamblaje de nubes son compatibles con AWS CDK CLI las versiones que utilizan esa versión del esquema o una versión posterior. Esto podría incluir versiones anteriores a AWS CDK CLI una versión de una biblioteca de construcción determinada.

Cuando la versión de ensamblaje en la nube requerida por la biblioteca de construcción no es compatible con la versión compatible con la AWS CDK CLI, recibirá un mensaje de error como el siguiente:

```
Cloud assembly schema version mismatch: Maximum schema version supported is 3.0.0, but
 found 4.0.0.
    Please upgrade your CLI in order to interact with this app.
```

Para resolver este error, AWS CDK CLI actualícela a una versión compatible con la versión de ensamblaje en nube requerida o a la última versión disponible. Por lo general, no se recomienda la alternativa (degradar los módulos de la biblioteca de construcción que usa tu aplicación).

> **Note**
>
> Para obtener más información sobre el esquema de ensamblaje en la nube, consulta Cloud Assembly Versioning.

AWS Construye el control de versiones de la biblioteca

Los módulos de la biblioteca AWS Construct pasan por varias etapas a medida que se desarrollan desde el concepto hasta la API madura. Las diferentes etapas ofrecen diversos grados de estabilidad de la API en las versiones posteriores de AWS CDK.

Las API de la AWS CDK biblioteca principal son estables y la biblioteca está completamente versionada semánticamente. `aws-cdk-lib` Este paquete incluye construcciones AWS CloudFormation (L1) para todos los AWS servicios y todos los módulos estables de nivel superior (L2 y L3). (También incluye las clases principales de CDK, como y). `App` `Stack` Las API no se eliminarán de este paquete (aunque es posible que estén en desuso) hasta la próxima versión principal de la CDK. Ninguna API individual tendrá nunca cambios importantes. Cuando se requiera un cambio importante, se añadirá una API completamente nueva.

Las nuevas API que se `aws-cdk-lib` estén desarrollando para un servicio ya incorporado se identifican mediante un `Beta`*`N`* sufijo, que `N` comienza en 1 y se incrementa con cada cambio importante que se realice en la nueva API. `Beta`*`N`* Las API nunca se eliminan, solo quedan en desuso, por lo que tu aplicación actual sigue funcionando con las versiones más recientes de`aws-cdk-lib`. Cuando la API se considera estable, se agrega una nueva API sin el `Beta`*`N`* sufijo.

Cuando las API de nivel superior (L2 o L3) comienzan a desarrollarse para un AWS servicio que anteriormente solo tenía API de nivel 1, esas API se distribuyen inicialmente en un paquete independiente. El nombre de dicho paquete tiene el sufijo «Alpha» y su versión coincide con la primera versión con la `aws-cdk-lib` que es compatible, con una subversión. `alpha` Cuando el módulo admite los casos de uso previstos, se añaden sus API. `aws-cdk-lib`

Estabilidad de la vinculación de idiomas

Con el tiempo, podríamos añadir soporte a AWS CDK otros lenguajes de programación. Si bien la API descrita en todos los lenguajes es la misma, la forma en que se expresa varía según el idioma y puede cambiar a medida que evolucione el soporte lingüístico. Por este motivo, los enlaces de idiomas se consideran experimentales durante un tiempo hasta que se considera que están listos para su uso en producción.

Language	Stability
TypeScript	Stable
JavaScript	Stable

Language	Stability
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK tutoriales

Esta sección contiene tutoriales para AWS Cloud Development Kit (AWS CDK).

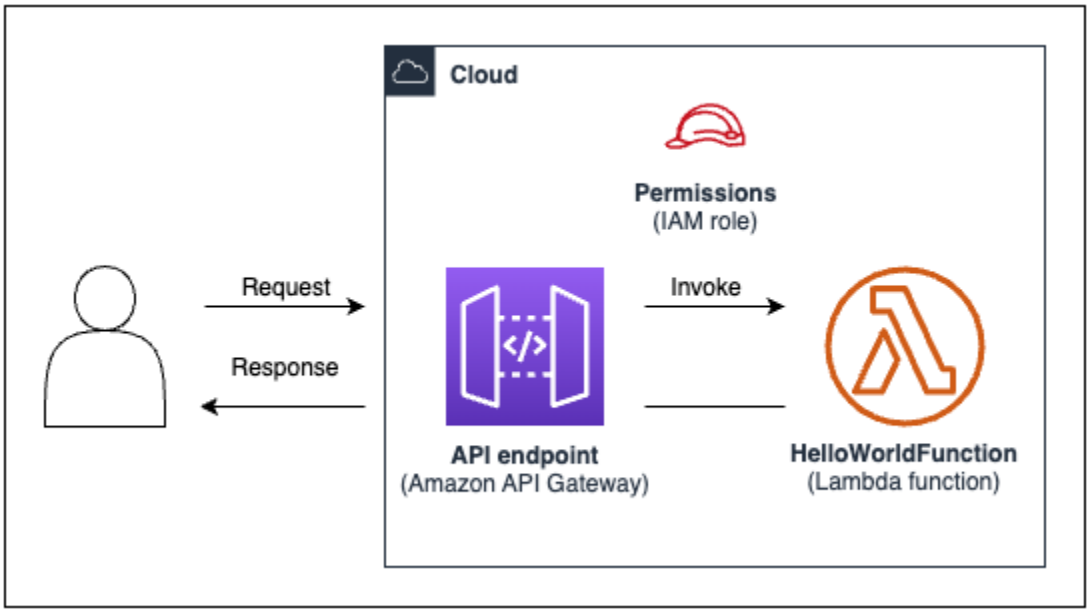
Temas

Cree una aplicación Hello World sin servidor

En este tutorial, se utiliza AWS Cloud Development Kit (AWS CDK) para crear una Hello World aplicación sencilla sin servidor que implemente un backend de API básico mediante la creación de lo siguiente:

- Amazon API Gateway REST API: proporciona un punto de enlace HTTP que se utiliza para invocar la función mediante una HTTP GET solicitud.
- AWS Lambda función: función que devuelve un `Hello World!` mensaje cuando se invoca con el HTTP punto final.
- Integraciones y permisos: detalles de configuración y permisos para que tus recursos interactúen entre sí y realicen acciones, como escribir registros en Amazon CloudWatch.

El siguiente diagrama muestra los componentes de esta aplicación:



Para este tutorial, completará lo siguiente:

1. Cree un AWS CDK proyecto.
2. Defina una función Lambda y una API REST de API Gateway mediante construcciones L2 de la biblioteca Construct. AWS
3. Implemente su aplicación en. Nube de AWS
4. Interactúe con su aplicación en el Nube de AWS.
5. Elimine la aplicación de muestra de la Nube de AWS.

Requisitos previos

Antes de comenzar este tutorial, debe completar lo siguiente:

- Cree un Cuenta de AWS e instale y configure el AWS Command Line Interface (AWS CLI).
- Instale Node.js ynpm.
- Instale el kit de herramientas CDK a nivel mundial, utilizando. `npm install -g aws-cdk`

Para obtener más información, consulte Cómo empezar con el AWS CDK.

También recomendamos tener conocimientos básicos de lo siguiente:

- ¿Qué es el AWS CDK?para una introducción básica al AWS CDK.
- AWS CDK conceptospara obtener una descripción general de los conceptos básicos del AWS CDK.

Paso 1: Crear un proyecto de CDK

En este paso, se crea un nuevo proyecto de CDK mediante el AWS CDK CLI `cdk init` comando.

Para crear un proyecto de CDK

1. Desde el directorio de inicio de su elección, cree y navegue hasta un directorio de proyectos con `cdk-hello-world` el nombre indicado en su máquina:

```
$ mkdir cdk-hello-world && cd cdk-hello-world
```

2. Usa el `cdk init` comando para crear un nuevo proyecto en tu lenguaje de programación preferido:

   TypeScript

   ```
   $ cdk init --language typescript
   ```

   Instale AWS CDK las bibliotecas:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   JavaScript

   ```
   $ cdk init --language javascript
   ```

   Instalar AWS CDK bibliotecas:

   ```
   $ npm install aws-cdk-lib constructs
   ```

   Python

   ```
   $ cdk init --language python
   ```

   Active el entorno virtual:

   ```
   $ source .venv/bin/activate
   ```

   Instale AWS CDK las bibliotecas y las dependencias del proyecto:

   ```
   (.venv)$ python3 -m pip install -r requirements.txt
   ```

   Java

   ```
   $ cdk init --language java
   ```

   Instale AWS CDK las bibliotecas y las dependencias del proyecto:

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

Instale AWS CDK las bibliotecas y las dependencias del proyecto:

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

Instale las dependencias del proyecto:

```
$ go mod tidy
```

El CDK CLI crea un proyecto con la siguiente estructura:

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
```

La CDK crea CLI automáticamente una aplicación de CDK que contiene una sola pila. La instancia de la aplicación CDK se crea a partir de la clase. App La siguiente es una parte del archivo de aplicación de la CDK:

TypeScript

Ubicado en`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

Ubicado en`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

Ubicado en`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

Ubicado en `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#

Ubicado en `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

Ubicado en `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
}
```

Paso 2: Cree su función Lambda

Dentro de su proyecto de CDK, cree un `lambda` directorio que incluya un archivo nuevo`hello.js`. A continuación, se muestra un ejemplo:

TypeScript

Desde la raíz del proyecto, ejecute lo siguiente:

```
$ mkdir lambda && cd lambda
```

```
$ touch hello.js
```

Ahora debería agregarse lo siguiente a su proyecto de CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

JavaScript

Desde la raíz de su proyecto, ejecute lo siguiente:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Ahora debería agregarse lo siguiente a su proyecto de CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

Desde la raíz de su proyecto, ejecute lo siguiente:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Ahora debería agregarse lo siguiente a su proyecto de CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

Desde la raíz de su proyecto, ejecute lo siguiente:

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

Ahora debería agregarse lo siguiente a su proyecto de CDK:

```
cdk-hello-world
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

Desde la raíz de su proyecto, ejecute lo siguiente:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Ahora debería agregarse lo siguiente a su proyecto de CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

Desde la raíz de su proyecto, ejecute lo siguiente:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Ahora debería agregarse lo siguiente a su proyecto de CDK:

```
cdk-hello-world
### lambda
    ### hello.js
```

> **Note**
>
> Para simplificar este tutorial, utilizamos una función JavaScript Lambda para todos los lenguajes de programación CDK.

Defina la función Lambda añadiendo lo siguiente al archivo recién creado:

```
exports.handler = async (event) => {
    return {
        statusCode: 200,
        headers: { "Content-Type": "text/plain" },
        body: JSON.stringify({ message: "Hello, World!" }),
    };
};
```

Paso 3: Defina sus construcciones

En este paso, definirá sus recursos de Lambda y API Gateway mediante construcciones AWS CDK L2.

Abra el archivo de proyecto que define su pila de CDK. Modificará este archivo para definir sus componentes fijos. El siguiente es un ejemplo del archivo de pila inicial:

TypeScript

Ubicado en`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

Ubicado en`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

Ubicado en`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

Ubicado en`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#

Ubicado en `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

Ubicado en `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
}
```

En este archivo, AWS CDK hace lo siguiente:

- Su instancia de pila de CDK se crea a partir de la clase. `Stack`
- La clase `Constructs` base se importa y se proporciona como ámbito o elemento principal de la instancia de la pila.

Defina su recurso de función Lambda

Para definir el recurso de la función Lambda, importe y utilice la construcción `aws-lambda` L2 de la AWS biblioteca Construct.

Modifique el archivo de pila de la siguiente manera:

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
// Import Lambda L2 construct
import * as lambda from 'aws-cdk-lib/aws-lambda';

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

    // Define the Lambda function resource
    const helloWorldFunction = new lambda.Function(this, 'HelloWorldFunction', {
      runtime: lambda.Runtime.NODEJS_20_X, // Choose any supported Node.js runtime
      code: lambda.Code.fromAsset('lambda'), // Points to the lambda directory
      handler: 'hello.handler', // Points to the 'hello' file in the lambda
 directory
```

```
    });
  }
}
```

JavaScript

```
const { Stack, Duration } = require('aws-cdk-lib');
// Import Lambda L2 construct
const lambda = require('aws-cdk-lib/aws-lambda');


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

    // Define the Lambda function resource
    const helloWorldFunction = new lambda.Function(this, 'HelloWorldFunction', {
      runtime: lambda.Runtime.NODEJS_20_X, // Choose any supported Node.js runtime
      code: lambda.Code.fromAsset('lambda'), // Points to the lambda directory
      handler: 'hello.handler', // Points to the 'hello' file in the lambda
 directory
    });
  }
}

module.exports = { CdkHelloWorldStack }
```

Python

```
from aws_cdk import (
    Stack,
    # Import Lambda L2 construct
    aws_lambda as _lambda,
)
# ...

class CdkHelloWorldStack(Stack):

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

        # Define the Lambda function resource
        hello_world_function = _lambda.Function(
```

```
            self,
            "HelloWorldFunction",
            runtime = _lambda.Runtime.NODEJS_20_X, # Choose any supported Node.js
 runtime
            code = _lambda.Code.from_asset("lambda"), # Points to the lambda
 directory
            handler = "hello.handler", # Points to the 'hello' file in the lambda
 directory
        )
```

> **Note**
>
> Importamos el `aws_lambda` módulo `_lambda` porque `lambda` es un identificador incorporado. 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
}

// ...
```

Aquí, se crea un recurso de función Lambda y se definen las siguientes propiedades:

- `runtime`— El entorno en el que se ejecuta la función. En este caso, utilizamos Node.js la versión20.x.
- `code`— La ruta al código de la función en su máquina local.
- `handler`— El nombre del archivo específico que contiene el código de la función.

Defina su REST API recurso de API Gateway

Para definir su REST API recurso de API Gateway, importe y utilice la construcción `aws-apigateway` L2 de la biblioteca AWS Construct.

Modifique el archivo de pila de la siguiente manera:

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
}

// ...
```

Aquí, se crea un REST API recurso de API Gateway, junto con lo siguiente:

- Una integración entre la función Lambda REST API y su función, que permite a la API invocar su función. Esto incluye la creación de un recurso de permisos de Lambda.
- Un nuevo nombre de recurso o ruta `hello` que se agrega a la raíz del punto final de la API. Esto crea un nuevo punto final que se suma `/hello` a tu baseURL.
- Un método GET para el `hello` recurso. Cuando se envía una solicitud GET al `/hello` punto final, se invoca la función Lambda y se devuelve su respuesta.

Paso 4: Prepare la aplicación para su implementación

En este paso, debe preparar la aplicación para su despliegue compilándola, si es necesario, y realizando una validación básica con el AWS CDK CLI `cdk synth` comando.

Si es necesario, cree la aplicación:

TypeScript

Desde la raíz del proyecto, ejecuta lo siguiente:

```
$ npm run build
```

JavaScript

No es necesario construir.

Python

No se requiere construir.

Java

Desde la raíz del proyecto, ejecute lo siguiente:

```
$ mvn package
```

C#

Desde la raíz del proyecto, ejecuta lo siguiente:

```
$ dotnet build src
```

Go

Desde la raíz del proyecto, ejecuta lo siguiente:

```
$ go build
```

Ejecuta `cdk synth` para sintetizar una AWS CloudFormation plantilla a partir de tu código CDK. Al utilizar construcciones de capa 2, muchos de los detalles de configuración necesarios AWS CloudFormation para facilitar la interacción entre la función Lambda y REST API los proporciona el. AWS CDK

Desde la raíz del proyecto, ejecute lo siguiente:

```
$ cdk synth
```

> **Note**
>
> Si recibes un error como el siguiente, comprueba que estás en el `cdk-hello-world` directorio e inténtalo de nuevo:
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

Si se ejecuta correctamente, AWS CDK CLI mostrará la AWS CloudFormation plantilla en YAML formato en la línea de comandos. También se guarda una plantilla JSON formateada en el `cdk.out` directorio.

A continuación se muestra un ejemplo de salida de la AWS CloudFormation plantilla:

AWS CloudFormation plantilla

```
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
  HelloWorldApiDeployment<unique-identifier>:
    Type: AWS::ApiGateway::Deployment
    Properties:
      Description: Automatically created by the RestApi construct
      RestApiId:
        Ref: HelloWorldApi<unique-identifier>
    DependsOn:
      - HelloWorldApihelloGET<unique-identifier>
      - HelloWorldApihello<unique-identifier>
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Deployment/Resource
  HelloWorldApiDeploymentStageprod<012345ABC>:
    Type: AWS::ApiGateway::Stage
    Properties:
      DeploymentId:
        Ref: HelloWorldApiDeployment<unique-identifier>
      RestApiId:
        Ref: HelloWorldApi<unique-identifier>
      StageName: prod
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/DeploymentStage.prod/Resource
  HelloWorldApihello<unique-identifier>:
    Type: AWS::ApiGateway::Resource
    Properties:
      ParentId:
        Fn::GetAtt:
          - HelloWorldApi<unique-identifier>
          - RootResourceId
      PathPart: hello
      RestApiId:
        Ref: HelloWorldApi<unique-identifier>
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/Resource
  HelloWorldApihelloGETApiPermissionCdkHelloWorldStackHelloWorldApi<unique-identifier>:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunction<unique-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.
```

Al utilizar construcciones de nivel 2, se definen algunas propiedades para configurar los recursos y se utilizan métodos auxiliares para integrarlos entre sí. AWS CDK Configura la mayoría de los AWS CloudFormation recursos y propiedades necesarios para aprovisionar la aplicación.

Paso 5: implementar la aplicación

En este paso, utiliza el AWS CDK CLI `cdk deploy` comando para implementar la aplicación. AWS CDK Funciona con el AWS CloudFormation servicio para aprovisionar sus recursos.

> **Important**
>
> Debe realizar un arranque único de su AWS entorno antes de la implementación. Para obtener instrucciones, consulte Bootstrap your environment.

Desde la raíz del proyecto, ejecute lo siguiente. Confirma los cambios si se te solicita:

```
$ cdk deploy

#  Synthesis time: 2.44s

...

Do you wish to deploy these changes (y/n)? y
```

Cuando se complete la implementación, AWS CDK CLI se generará la URL de su punto final. Copie esta URL para el siguiente paso. A continuación, se muestra un ejemplo:

```
...
#  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
...
```

Paso 6: Interactúa con tu aplicación

En este paso, inicia una solicitud GET en el punto final de la API y recibe la respuesta de la función Lambda.

Localice la URL del punto final del paso anterior y añada la `/hello` ruta. A continuación, mediante el navegador o la línea de comandos, envía una solicitud GET a tu punto final. A continuación, se muestra un ejemplo:

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello
{"message":"Hello World!"}%
```

¡Enhorabuena! ¡Ha creado, implementado e interactuado correctamente con su aplicación mediante el AWS CDK!

Paso 7: Elimine su aplicación

En este paso, utiliza el AWS CDK CLI para eliminar su aplicación del Nube de AWS.

Para eliminar la aplicación, ejecute`cdk destroy`. Cuando se le solicite, confirme su solicitud para eliminar la aplicación:

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
 #  CdkHelloWorldStack: destroyed
```

Solución de problemas

Error: {«message»: «Error interno del servidor»}%

Al invocar la función Lambda desplegada, recibe este error. Este error puede producirse por varios motivos.

Para seguir solucionando problemas

Utilice el AWS CLI para invocar la función Lambda.

1. Modifique el archivo de pila para capturar el valor de salida del nombre de la función Lambda implementada. A continuación, se muestra un ejemplo:

```
...

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. Vuelva a implementar la aplicación. El resultado AWS CDK CLI será el valor del nombre de la función Lambda implementada:

```
$ cdk deploy

#  Synthesis time: 0.29s
...
 #  CdkHelloWorldStack

#  Deployment time: 20.36s
```

```
Outputs:
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. Utilice el AWS CLI para invocar la función Lambda en y enviar Nube de AWS la respuesta a un archivo de texto:

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-
identifier output.txt
```

4. Compruebe `output.txt` los resultados.

Causa posible: el recurso API Gateway está definido incorrectamente en el archivo de pila.

Si `output.txt` muestra una respuesta correcta de la función Lambda, el problema podría estar relacionado con la forma en que definió la API REST de API Gateway. AWS CLI Invoca su Lambda directamente, no a través de su punto final. Compruebe su código para asegurarse de que coincide con este tutorial. A continuación, vuelva a implementar.

Causa posible: el recurso Lambda está definido incorrectamente en el archivo de pila.

Si `output.txt` devuelve un error, el problema podría estar relacionado con la forma en que definió la función Lambda. Comprueba tu código para asegurarte de que coincide con este tutorial. A continuación, vuelva a implementar.

Crea una aplicación con múltiples pilas

Puede crear una AWS Cloud Development Kit (AWS CDK) aplicación que contenga varias pilas. Al implementar la AWS CDK aplicación, cada pila se convierte en su propia AWS CloudFormation plantilla. También puedes sintetizar e implementar cada pila de forma individual mediante el AWS CDK CLI `cdk deploy` comando.

En este tutorial se explica lo siguiente:

- Cómo extender la `Stack` clase para que acepte nuevas propiedades o argumentos.
- Cómo usar las propiedades para determinar qué recursos contiene la pila y su configuración.
- Cómo crear instancias de varias pilas de esta clase.

El ejemplo de este tema usa una propiedad booleana llamada (`encryptBucket`Python:`encrypt_bucket`). Indica si un bucket de Amazon S3 debe cifrarse. Si es así, la pila permite el cifrado mediante una clave administrada por AWS Key Management Service (AWS KMS). La aplicación crea dos instancias de esta pila, una con cifrado y otra sin él.

Temas

Antes de empezar

En primer lugar, instale Node.js y las herramientas de línea de AWS CDK comandos, si aún no lo ha hecho. Para obtener más información, consulte Cómo empezar con el AWS CDK.

A continuación, cree un AWS CDK proyecto introduciendo los siguientes comandos en la línea de comandos.

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
```

Puede importar el proyecto Maven resultante a su IDE de Java.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

Puede abrir el archivo `src/Pipeline.sln` en Visual Studio.

Agregue un parámetro opcional

El `props` argumento del `Stack` constructor completa la interfaz`StackProps`. En este ejemplo, queremos que la pila acepte una propiedad adicional que nos indique si debemos cifrar el bucket de Amazon S3. Deberíamos crear una interfaz o clase que incluya la propiedad. Esto permite al compilador asegurarse de que la propiedad tiene un valor booleano y permite completarla automáticamente en su IDE.

Así que abre el archivo fuente indicado en tu IDE o editor y añade la nueva interfaz, clase o argumento. El código debería tener este aspecto después de los cambios. Las líneas que hemos añadido se muestran en negrita.

TypeScript

Archivo: `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

Archivo: `lib/multistack-stack.js`

JavaScript no tiene una función de interfaz; no necesitamos añadir ningún código.

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

class MultistackStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // The code that defines your stack goes here
  }
}

module.exports = { MultistackStack }
```

Python

Archivo: `multistack/multistack_stack.py`

Python no tiene una función de interfaz, por lo que ampliaremos nuestra pila para aceptar la nueva propiedad añadiendo un argumento de palabra clave.

```
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

Archivo: `src/main/java/com/myorg/MultistackStack.java`

Ampliar un tipo de accesorios en Java es más complicado de lo que realmente queremos. En su lugar, escribe el constructor de la pila para aceptar un parámetro booleano opcional. Como `props` es un argumento opcional, escribiremos un constructor adicional que te permita omitirlo. Estará predeterminado en`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#

Archivo: `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 nueva propiedad es opcional. Si `encryptBucket` (Python:`encrypt_bucket`) no está presente, su valor es`undefined`, o el equivalente local. El depósito no estará cifrado de forma predeterminada.

Defina la clase de pila

Ahora definamos nuestra clase de pila, usando nuestra nueva propiedad. Haz que el código tenga el siguiente aspecto. El código que necesita añadir o cambiar se muestra en negrita.

TypeScript

Archivo: `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

Archivo: `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

Archivo: `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

Archivo: `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#

Archivo: `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 dos instancias de pila

Ahora añadiremos el código para crear instancias de dos pilas distintas. Como antes, las líneas de código que se muestran en negrita son las que debes agregar. Elimine la `MultistackStack` definición existente.

TypeScript

Archivo: `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

Archivo: `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

Archivo: `./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

Archivo: `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#

Archivo: 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();
        }
    }
}
```

Este código usa la nueva propiedad `encryptBucket` (Python:`encrypt_bucket`) de la `MultistackStack` clase para instanciar lo siguiente:

- Una pila con un bucket de Amazon S3 cifrado en la `us-east-1` AWS región.
- Una pila con un bucket de Amazon S3 sin cifrar en la `us-west-1` AWS región.

Sintetice e implemente la pila

Ahora puede implementar pilas desde la aplicación. En primer lugar, sintetiza una AWS CloudFormation plantilla para`MyEastCdkStack`: la pila en. `us-east-1` Esta es la pila con el bucket S3 cifrado.

```
$ cdk synth MyEastCdkStack
```

Para implementar esta pila en su AWS cuenta, ejecute uno de los siguientes comandos. El primer comando usa tu AWS perfil predeterminado para obtener las credenciales necesarias para implementar la pila. El segundo usa un perfil que usted especifique. En lugar de *`PROFILE_NAME`*, sustituya el nombre de un AWS CLI perfil que contenga las credenciales adecuadas para el despliegue en la `us-east-1` AWS región.

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

Limpieza

Para evitar que se le cobre por los recursos que haya desplegado, destruya la pila mediante el siguiente comando.

```
cdk destroy MyEastCdkStack
```

La operación de destrucción falla si hay algo almacenado en el depósito de la pila. No debería ser así si solo has seguido las instrucciones de este tema. Pero si pusiste algo en el depósito, debes eliminar el contenido del depósito antes de destruir la pila. (No elimines el cubo en sí). Utilice el AWS Management Console o el AWS CLI para eliminar el contenido del depósito.

Ejemplos

En este tema se incluyen los siguientes ejemplos:

- Cree una aplicación Hello World sin servidorCrea una aplicación sin servidor mediante Lambda, API Gateway y Amazon S3.
- Creación de un servicio AWS Fargate mediante el AWS CDKCrea un servicio Amazon ECS Fargate a partir de una imagen. DockerHub

Creación de un servicio AWS Fargate mediante el AWS CDK

En este ejemplo, se explica cómo crear un servicio AWS Fargate que se ejecute en un clúster de Amazon Elastic Container Service (Amazon ECS) encabezado por un Application Load Balancer con acceso a Internet a partir de una imagen de Amazon ECR.

Amazon ECS es un servicio de administración de contenedores muy escalable y rápido que facilita la tarea de ejecutar, detener y administrar contenedores de Docker en un clúster. Puede alojar su clúster en una infraestructura sin servidor gestionada por Amazon ECS lanzando sus servicios o tareas mediante el tipo de lanzamiento Fargate. Para tener más control, puede alojar sus tareas en un clúster de instancias de Amazon Elastic Compute Cloud (Amazon EC2) que administra mediante el tipo de lanzamiento de Amazon EC2.

En este tutorial se muestra cómo lanzar algunos servicios mediante el tipo de lanzamiento Fargate. Si ha utilizado el AWS Management Console para crear un servicio Fargate, sabrá que hay muchos pasos a seguir para realizar esa tarea. AWS tiene varios tutoriales y temas de documentación que le guiarán a través de la creación de un servicio Fargate, entre los que se incluyen:

- Cómo implementar contenedores Docker - AWS
- Configuración con Amazon ECS
- Cómo empezar a utilizar Amazon ECS mediante Fargate

En este ejemplo, se crea un servicio Fargate similar en el AWS CDK código.

La construcción de Amazon ECS utilizada en este tutorial le ayuda a utilizar AWS los servicios al proporcionarle las siguientes ventajas:

- Configura automáticamente un balanceador de carga.

- Abre automáticamente un grupo de seguridad para los balanceadores de carga. Esto permite que los balanceadores de carga se comuniquen con las instancias sin que tengas que crear un grupo de seguridad de forma explícita.
- Ordena automáticamente la dependencia entre el servicio y el balanceador de cargas asociado a un grupo objetivo, donde se AWS CDK impone el orden correcto de creación del listener antes de que se cree una instancia.
- Configura automáticamente los datos de usuario en grupos que escalan automáticamente. Esto crea la configuración correcta para asociar un clúster a las AMI.
- Valida anticipadamente las combinaciones de parámetros. Esto expone AWS CloudFormation los problemas antes y, por lo tanto, ahorra tiempo de implementación. Por ejemplo, en función de la tarea, es fácil configurar mal los ajustes de memoria. Anteriormente, no se producía ningún error hasta que implementaba la aplicación. Pero ahora AWS CDK pueden detectar un error de configuración y emitir un error al sintetizar la aplicación.
- Añade automáticamente permisos para Amazon Elastic Container Registry (Amazon ECR) si utilizas una imagen de Amazon ECR.
- Se escala automáticamente. AWS CDK Proporciona un método para que pueda escalar automáticamente las instancias cuando utilice un clúster de Amazon EC2. Esto ocurre automáticamente cuando usas una instancia en un clúster de Fargate.

  Además, AWS CDK evita que se elimine una instancia cuando el escalado automático intenta cerrar una instancia, pero una tarea se está ejecutando o está programada en esa instancia.

  Anteriormente, tenía que crear una función Lambda para disponer de esta funcionalidad.
- Proporciona soporte de activos para que pueda implementar una fuente desde su máquina en Amazon ECS en un solo paso. Anteriormente, para utilizar una fuente de aplicación había que realizar varios pasos manuales, como subirla a Amazon ECR y crear una imagen de Docker.

Consulte ECS para obtener más información.

> **Important**
>
> Las `ApplicationLoadBalancedFargateService` configuraciones que usaremos incluyen numerosos AWS componentes, algunos de los cuales tienen costos no triviales si se dejan aprovisionados en su AWS cuenta, incluso si no los usa. Asegúrate de limpiar up (cdk destroy) después de completar este ejemplo.

Crear el directorio e inicializar el AWS CDK

Empecemos por crear un directorio para almacenar el AWS CDK código y, a continuación, crear una AWS CDK aplicación en ese directorio.

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
```

Ahora puede importar el proyecto Maven a su IDE.

C#

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language csharp
```

Ahora puede abrirlo `src/MyEcsConstruct.sln` en Visual Studio.

Ejecute la aplicación y confirme que crea una pila vacía.

```
cdk synth
```

Cree un servicio Fargate

Existen dos formas diferentes de ejecutar las tareas de contenedores con Amazon ECS:

- Utilice el tipo de `Fargate` lanzamiento, en el que Amazon ECS administra por usted las máquinas físicas en las que se ejecutan los contenedores.
- Utilice el tipo de `EC2` lanzamiento, en el que se encarga de la administración, por ejemplo, especificando el escalado automático.

Para este ejemplo, crearemos un servicio Fargate que se ejecute en un clúster de ECS encabezado por un Application Load Balancer con acceso a Internet.

Añada las siguientes importaciones del módulo AWS Construct Library al archivo indicado.

TypeScript

Archivo: `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

Archivo: `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

Archivo: `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

Archivo: `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#

Archivo: `src/MyEcsConstruct/MyEcsConstructStack.cs`

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

Sustituya el comentario al final del constructor por el siguiente código.

TypeScript

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

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

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

JavaScript

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

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

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

Python

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

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

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

Java

```
        Vpc vpc = Vpc.Builder.create(this, "MyVpc")
                        .maxAzs(3)  // Default is all AZs in region
                        .build();
```

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

        // Create a load-balanced Fargate service and make it public
        ApplicationLoadBalancedFargateService.Builder.create(this,
 "MyFargateService")
                      .cluster(cluster)           // Required
                      .cpu(512)                   // Default is 256
                       .desiredCount(6)            // Default is 1
                       .taskImageOptions(
                               ApplicationLoadBalancedTaskImageOptions.builder()
                                       .image(ContainerImage.fromRegistry("amazon/
amazon-ecs-sample"))
                                       .build())
                       .memoryLimitMiB(2048)       // Default is 512
                       .publicLoadBalancer(true)   // Default is true
                       .build();
```

C#

```
            var vpc = new Vpc(this, "MyVpc", new VpcProps
            {
                MaxAzs = 3 // Default is all AZs in region
            });

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

            // Create a load-balanced Fargate service and make it public
            new ApplicationLoadBalancedFargateService(this, "MyFargateService",
                new ApplicationLoadBalancedFargateServiceProps
                {
                    Cluster = cluster,          // Required
                    DesiredCount = 6,           // Default is 1
                    TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
                    {
                        Image = ContainerImage.FromRegistry("amazon/amazon-ecs-
sample")
                    },
                    MemoryLimitMiB = 2048,      // Default is 256
                    PublicLoadBalancer = true   // Default is true
```

```
                }
            );
```

Guárdalo y asegúrate de que se ejecuta y crea una pila.

```
cdk synth
```

La pila está formada por cientos de líneas, por lo que no la mostraremos aquí. La pila debe contener una instancia predeterminada, una subred privada y una subred pública para las tres zonas de disponibilidad y un grupo de seguridad.

Implemente la pila.

```
cdk deploy
```

AWS CloudFormation muestra información sobre las docenas de pasos que debe realizar para implementar la aplicación.

Así de fácil es crear un servicio Amazon ECS con tecnología Fargate para ejecutar una imagen de Docker.

Limpieza

Para evitar AWS cargos inesperados, destruye tu AWS CDK pila cuando termines de hacer este ejercicio.

```
cdk destroy
```

AWS CDK ejemplos

Para ver más ejemplos de AWS CDK pilas y aplicaciones en tu lenguaje de programación compatible favorito, consulta el repositorio de AWS CDK ejemplos en GitHub.

AWS CDK herramientas

Esta sección contiene información sobre las AWS CDK herramientas que se enumeran a continuación.

Temas

AWS CDK Kit de herramientas (**cdk**comando)

El AWS CDK kit de herramientas, el comando CLI`cdk`, es la herramienta principal para interactuar con AWS CDK la aplicación. Ejecuta tu aplicación, consulta el modelo de aplicación que has definido y produce e implementa las AWS CloudFormation plantillas generadas por el. AWS CDK También proporciona otras funciones útiles para crear proyectos y trabajar con ellos. AWS CDK Este tema contiene información sobre los casos de uso más comunes del kit de herramientas CDK.

El AWS CDK kit de herramientas se instala con el Node Package Manager. En la mayoría de los casos, recomendamos instalarlo globalmente.

```
npm install -g aws-cdk             # install latest version
npm install -g aws-cdk@X.YY.Z      # install specific version
```

> **Tip**
>
> Si trabaja habitualmente con varias versiones del AWS CDK, considere la posibilidad de instalar una versión correspondiente del AWS CDK kit de herramientas en proyectos individuales de CDK. Para ello, omita en el comando`-g`. `npm install` Luego úsalo `npx aws-cdk` para invocarlo. Esto ejecuta la versión local si existe, y recurre a una versión global si no existe.

Comandos del kit de herramientas

Todos los comandos del CDK Toolkit comienzan con`cdk`, seguido de un subcomando (`list`,, `synthesize`o `deploy`, etc.). Algunos subcomandos tienen una versión más corta (`ls`,`synth`, etc.) que es equivalente. Las opciones y los argumentos siguen al subcomando en cualquier orden. Los comandos disponibles se resumen aquí.

Comando	Función
`cdk list` (`ls`)	Muestra las pilas de la aplicación
`cdk synthesize` (`synth`)	Sintetiza e imprime la CloudFormation plantilla para una o más pilas especificadas
`cdk bootstrap`	Implementa la pila de almacenamiento provisional del CDK Toolkit; consulte the section called “Bootstrapping (Proceso de arranque)”
`cdk deploy`	Despliega una o más pilas especificadas
`cdk destroy`	Destruye una o más pilas especificadas
`cdk diff`	Compara la pila especificada y sus dependenc ias con las pilas implementadas o con una plantilla local CloudFormation
`cdk import`	Utiliza las importaciones CloudFormation de recursos para incorporar los recursos existente s a una pila administrada por CDK
`cdk metadata`	Muestra los metadatos de la pila especificada
`cdk init`	Crea un nuevo proyecto de CDK en el directori o actual a partir de una plantilla especificada
`cdk context`	Administra los valores de contexto almacenad os en caché

Comando	Función
`cdk docs` (`doc`)	Abre la referencia de la API de CDK en su navegador
`cdk doctor`	Comprueba si su proyecto de CDK tiene posibles problemas

Para ver las opciones disponibles para cada comando, consultethe section called “Ayuda integrada”.

Especificar las opciones y sus valores

Las opciones de la línea de comandos comienzan con dos guiones ()`--`. Algunas opciones que se utilizan con frecuencia tienen sinónimos de una sola letra que comienzan con un solo guión (por ejemplo, `--app` tiene un sinónimo). `-a` El orden de las opciones en un comando del AWS CDK kit de herramientas no es importante.

Todas las opciones aceptan un valor, que debe seguir al nombre de la opción. El valor puede estar separado del nombre por un espacio en blanco o por un signo igual=. Las dos opciones siguientes son equivalentes.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

Algunas opciones son banderas (booleanas). Puede especificar `true` o `false` como su valor. Si no proporciona un valor, se considerará que el valor es`true`. También puede anteponer el nombre de la opción `no-` para dar a entender`false`.

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

Algunas opciones, a saber`--context`,`--parameters`,`--plugin`, y `--tags``--trust`, se pueden especificar más de una vez para especificar varios valores. Se indica que están escritas en `[array]` la ayuda del CDK Toolkit. Por ejemplo:

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

Ayuda integrada

El AWS CDK kit de herramientas tiene una ayuda integrada. Para ver la ayuda general sobre la utilidad y una lista de los subcomandos proporcionados, ejecute lo siguiente:

```
cdk --help
```

Para obtener ayuda sobre un subcomando concreto, por ejemplo`deploy`, especifíquelo antes de la `--help` bandera.

```
cdk deploy --help
```

Problema `cdk version` para mostrar la versión del AWS CDK kit de herramientas. Proporcione esta información cuando solicite asistencia.

Informes de versiones

Para obtener información sobre cómo AWS CDK se usa, las construcciones utilizadas por AWS CDK las aplicaciones se recopilan e informan mediante un recurso identificado como`AWS::CDK::Metadata`. Este recurso se añade a AWS CloudFormation las plantillas y se puede revisar fácilmente. Esta información también se puede utilizar AWS para identificar pilas mediante un constructo con problemas conocidos de seguridad o confiabilidad. También se puede usar para contactar a sus usuarios con información importante.

> **Note**
>
> Antes de la versión 1.93.0, AWS CDK indicaban los nombres y las versiones de los módulos cargados durante la síntesis, en lugar de las construcciones utilizadas en la pila.

De forma predeterminada, AWS CDK informa del uso de construcciones en los siguientes módulos de NPM que se utilizan en la pila:

- AWS CDK módulo principal
- AWS Construya módulos de biblioteca
- AWS Módulo Solutions Constructs
- AWS Módulo del kit de implementación de Render Farm

El `AWS::CDK::Metadata` recurso tiene un aspecto parecido al siguiente.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
  Properties:
    Analytics:
 "v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/
XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz
+AqqLDt2Y3YMnLYWwAAAA="
```

La `Analytics` propiedad es una lista comprimida con gzip, codificada en base64 y codificada en prefijo de las construcciones de la pila.

Para excluirse de los informes de versiones, utilice uno de los siguientes métodos:

- Utilice el cdk comando con el --no-version-reporting argumento para excluirse de un solo comando.

  ```
  cdk --no-version-reporting synth
  ```

  Recuerde que el AWS CDK kit de herramientas sintetiza plantillas nuevas antes de implementarlas, por lo que también debe agregarlas `--no-version-reporting` a los `cdk deploy` comandos.
- versionReportingEstablézcalo en falso en `./cdk.json` o. `~/.cdk.json` Esta opción se desactiva a menos que lo especifique `--version-reporting` en un comando individual.

  ```
  {
    "app": "...",
    "versionReporting": false
  }
  ```

Autenticación con AWS

Existen diferentes formas de configurar el acceso programático a AWS los recursos, según el entorno y el AWS acceso del que disponga.

Para elegir el método de autenticación y configurarlo para el kit de herramientas del CDK, consulte Autenticación y acceso en la Guía de referencia de los AWS SDK y las herramientas.

El enfoque recomendado para los nuevos usuarios que se desarrollen a nivel local, a los que su empleador no les haya proporcionado un método de autenticación, consiste en configurarlo. AWS IAM Identity Center Este método incluye la instalación del AWS CLI para facilitar la configuración y para iniciar sesión con regularidad en el portal de AWS acceso. Si elige este método, su entorno debería contener los siguientes elementos después de completar el procedimiento de Autenticación del Centro de Identidad de IAM descrito en la Guía de referencia de las herramientas y los SDK de AWS:

- El AWS CLI, que se utiliza para iniciar una sesión en el portal de AWS acceso antes de ejecutar la aplicación.
- Un AWS`config`archivo compartido que tiene un `[default]` perfil con un conjunto de valores de configuración a los que se puede hacer referencia desde AWS CDK. Para encontrar la ubicación de este archivo, consulte Ubicación de los archivos compartidos en la Guía de referencia de las herramientas y los SDK de AWS.
- El archivo compartido de `config` establece la configuración de `region`. Esto establece el uso predeterminado de The CDK Toolkit AWS CDK y Región de AWS del CDK Toolkit para las AWS solicitudes.
- El kit de herramientas CDK utiliza la configuración del proveedor de tokens de SSO del perfil para adquirir las credenciales antes de enviar las solicitudes a ellas. AWS El `sso_role_name` valor, que es un rol de IAM conectado a un conjunto de permisos del Centro de Identidad de IAM, debería permitir el acceso a los utilizados en la Servicios de AWS aplicación.

  El siguiente archivo `config` de ejemplo muestra la configuración de un perfil predeterminado con el proveedor de token de SSO. La configuración `sso_session` del perfil hace referencia a la sección llamada `sso-session`. La `sso-session` sección contiene la configuración para iniciar una sesión en el portal de AWS acceso.

```
[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
```

Inicie una sesión en el portal de AWS acceso

Antes de acceder Servicios de AWS, necesita una sesión activa en el portal de AWS acceso para que el CDK Toolkit utilice la autenticación del IAM Identity Center para resolver las credenciales. En función de la duración de las sesiones configuradas, el acceso eventualmente caducará y el CDK Toolkit detectará un error de autenticación. Ejecute el siguiente comando AWS CLI para iniciar sesión en el portal de AWS acceso.

```
aws sso login
```

Si la configuración de su proveedor de token de SSO utiliza un perfil con nombre en lugar del perfil predeterminado, el comando es`aws sso login --profile NAME`. Especifique también este perfil al emitir cdk comandos mediante la --profile opción o la variable de `AWS_PROFILE` entorno.

Para comprobar si ya tiene una sesión activa, ejecute el siguiente AWS CLI comando.

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

La respuesta a este comando debe indicar la cuenta y el conjunto de permisos del Centro de identidades de IAM configurados en el archivo compartido `config`.

> **Note**
>
> Si ya tiene una sesión activa en el portal de AWS acceso y la ejecuta`aws sso login`, no tendrá que proporcionar credenciales.
> Es posible que el proceso de inicio de sesión le pida que permita el AWS CLI acceso a sus datos. Dado que AWS CLI se basa en el SDK para Python, los mensajes de permiso pueden contener variaciones del `botocore` nombre.

Especificar la región y otras configuraciones

El kit de herramientas del CDK debe conocer la AWS región en la que se va a realizar la implementación y cómo autenticarse. AWS Esto es necesario para las operaciones de despliegue y para recuperar los valores de contexto durante la síntesis. Juntas, su cuenta y su región forman el entorno.

La región se puede especificar mediante variables de entorno o en archivos de configuración. Se trata de las mismas variables y archivos que utilizan otras AWS herramientas, como AWS CLI los distintos AWS SDK. El kit de herramientas del CDK busca esta información en el siguiente orden.

- La variable de entorno `AWS_DEFAULT_REGION`.
- Un perfil con nombre definido en el AWS `config` archivo estándar y especificado mediante la `--profile` opción de comandos. `cdk`
- La `[default]` sección del AWS `config` archivo estándar.

Además de especificar la AWS autenticación y una región en la `[default]` sección, también puede añadir una o más `[profile` *`NAME`*`]` secciones, donde *`NAME`* es el nombre del perfil. Para obtener más información sobre los perfiles con nombre, consulta los archivos de configuración y credenciales compartidos en la Guía de referencia de herramientas y AWS SDK.

El AWS `config` archivo estándar se encuentra en `~/.aws/config` (macOS/Linux) o `%USERPROFILE%\.aws\config` (Windows). Para obtener detalles y ubicaciones alternativas, consulte la ubicación de los archivos de configuración y credenciales compartidos en la Guía de referencia de herramientas y AWS SDK

El entorno que especificas en tu AWS CDK aplicación mediante la `env` propiedad de la pila se utiliza durante la síntesis. Se usa para generar una AWS CloudFormation plantilla específica del entorno y, durante la implementación, anula la cuenta o región especificada mediante uno de los métodos anteriores. Para obtener más información, consulte the section called “Entornos”.

> **Note**
>
> AWS CDK Utiliza credenciales de los mismos archivos fuente que otras AWS herramientas y SDK, incluido el. AWS Command Line Interface Sin embargo, AWS CDK es posible que se comporten de forma algo diferente a estas herramientas. Utiliza la parte AWS SDK for

JavaScript inferior del capó. Para obtener detalles completos sobre la configuración de las credenciales para el AWS SDK for JavaScript, consulte Configuración de credenciales.

Si lo desea, puede utilizar la opción `--role-arn` (`o-r`) para especificar el ARN de un rol de IAM que debe usarse para la implementación. La AWS cuenta que se utilice debe poder asumir esta función.

Especificar el comando de la aplicación

Muchas funciones del kit de herramientas del CDK requieren que se sinteticen una o más AWS CloudFormation plantillas, lo que a su vez requiere ejecutar la aplicación. AWS CDK Es compatible con programas escritos en varios lenguajes. Por lo tanto, utiliza una opción de configuración para especificar el comando exacto necesario para ejecutar la aplicación. Esta opción se puede especificar de dos maneras.

En primer lugar, y lo más habitual, se puede especificar con la `app` clave que se encuentra dentro del archivo`cdk.json`. Se encuentra en el directorio principal de tu AWS CDK proyecto. El kit de herramientas CDK proporciona un comando adecuado al crear un nuevo proyecto con. `cdk init` Este es el `cdk.json` de un TypeScript proyecto nuevo, por ejemplo.

```
{
  "app": "npx ts-node bin/hello-cdk.ts"
}
```

El kit de herramientas CDK busca `cdk.json` en el directorio de trabajo actual cuando intenta ejecutar tu aplicación. Por eso, puedes mantener un shell abierto en el directorio principal de tu proyecto para emitir los comandos del CDK Toolkit.

El kit de herramientas del CDK también busca la clave de la aplicación `~/.cdk.json` (es decir, en tu directorio principal) si no la encuentra. `./cdk.json` Añadir el comando de la aplicación aquí puede resultar útil si normalmente trabajas con código CDK en el mismo idioma.

Si estás en otro directorio o quieres ejecutar tu aplicación con un comando distinto del que aparece`cdk.json`, usa la opción `--app` (`o-a`) para especificarlo.

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

Al realizar la implementación, también puedes especificar un directorio que contenga conjuntos de nubes sintetizados, por ejemplo`cdk.out`, como el valor de--app. Las pilas especificadas se despliegan desde este directorio; la aplicación no se sintetiza.

Especificar pilas

Muchos comandos del CDK Toolkit (por ejemplo`cdk deploy`) funcionan en las pilas definidas en tu aplicación. Si tu aplicación contiene solo una pila, el kit de herramientas del CDK supone que hablas en serio si no especificas una pila de forma explícita.

De lo contrario, debes especificar la pila o pilas con las que quieres trabajar. Para ello, especifique las pilas deseadas por ID de forma individual en la línea de comandos. Recuerde que el ID es el valor especificado por el segundo argumento al crear una instancia de la pila.

```
cdk synth PipelineStack LambdaStack
```

También puedes usar caracteres comodín para especificar identificadores que coincidan con un patrón.

- ?coincide con cualquier carácter individual
- *coincide con cualquier número de caracteres (*solo coincide con todas las pilas)
- **coincide con todos los elementos de una jerarquía

También puede usar la --all opción para especificar todas las pilas.

Si tu aplicación usa CDK Pipelines, el kit de herramientas de CDK entiende tus pilas y etapas como una jerarquía. Además, la --all opción y el * comodín solo coinciden con las pilas de nivel superior. Para hacer coincidir todas las pilas, usa. ** También se usa ** para indicar todas las pilas de una jerarquía determinada.

Cuando utilice caracteres comodín, encierre el patrón entre comillas o evite los caracteres comodín con ellos. \ Si no lo hace, es posible que la consola intente expandir el patrón hasta incluir los nombres de los archivos del directorio actual. En el mejor de los casos, esto no hará lo que esperabas; en el peor de los casos, podrías desplegar pilas que no tenías intención de usar. Esto no es estrictamente necesario en Windows porque `cmd.exe` no expande los caracteres comodín, pero no obstante es una buena práctica.

```
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**
>
> El orden en el que se especifican las pilas no es necesariamente el orden en el que se procesarán. El AWS CDK kit de herramientas tiene en cuenta las dependencias entre las pilas a la hora de decidir el orden en el que se van a procesar. Por ejemplo, supongamos que una pila usa un valor producido por otra (como el ARN de un recurso definido en la segunda pila). En este caso, la segunda pila se sintetiza antes que la primera debido a esta dependencia. Puedes añadir dependencias entre las pilas de forma manual mediante el método de la pila. `addDependency()`

Cómo arrancar su entorno AWS

La implementación de pilas con la CDK requiere el aprovisionamiento de recursos especiales dedicados AWS CDK . El `cdk bootstrap` comando crea los recursos necesarios para usted. Solo necesita arrancar si va a implementar una pila que requiera estos recursos dedicados. Para obtener más información, consulte the section called “Bootstrapping (Proceso de arranque)”.

```
cdk bootstrap
```

Si se ejecuta sin argumentos, como se muestra aquí, el `cdk bootstrap` comando sintetiza la aplicación actual y arranca los entornos en los que se desplegarán sus pilas. Si la aplicación contiene pilas independientes del entorno, que no especifican un entorno de forma explícita, se arrancan la cuenta y la región predeterminadas, o bien se especifica el entorno mediante el cual. `--profile`

Fuera de una aplicación, debes especificar de forma explícita el entorno que se va a iniciar. También puedes hacerlo para iniciar un entorno que no esté especificado en tu aplicación o en tu perfil local. AWS Las credenciales deben estar configuradas (por ejemplo, en`~/.aws/credentials`) para la cuenta y la región especificadas. Puede especificar un perfil que contenga las credenciales necesarias.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.
cdk bootstrap 1111111111/us-east-1
```

```
cdk bootstrap --profile test 1111111111/us-east-1
```

> **Important**
>
> Cada entorno (combinación de cuenta/región) en el que se implemente dicha pila debe iniciarse por separado.

Es posible que se le cobre AWS por lo que AWS CDK almacenen en los recursos de arranque. Además, si la usa`-bootstrap-customer-key`, se creará una clave de AWS KMS, que también conlleva cargos por entorno.

> **Note**
>
> Las versiones anteriores de la plantilla bootstrap creaban una clave KMS de forma predeterminada. Para evitar cargos, vuelva a arrancar usando. `--no-bootstrap-customer-key`

> **Note**
>
> CDK Toolkit v2 no admite la plantilla de bootstrap original, denominada plantilla heredada, que se utiliza de forma predeterminada con CDK v1.

> **Important**
>
> La plantilla de bootstrap moderna concede de forma efectiva los permisos implícitos `--cloudformation-execution-policies` a cualquier AWS cuenta de la lista. `--trust` De forma predeterminada, esto amplía los permisos de lectura y escritura en cualquier recurso de la cuenta de arranque. Asegúrese de configurar la pila de arranque con políticas y cuentas de confianza con las que se sienta cómodo.

Crear una nueva aplicación

Para crear una nueva aplicación, cree un directorio para ella y, después, dentro del directorio, ejecute`cdk init`.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

Los idiomas admitidos (*IDIOMA*) son:

Código	Idioma
`typescript`	TypeScript
`javascript`	JavaScript
`python`	Python
`java`	Java
`csharp`	C#

TEMPLATE es una plantilla opcional. Si la plantilla deseada es una aplicación, la predeterminada, puede omitirla. Las plantillas disponibles son:

Plantilla	Descripción
`app`(predeterminado)	Crea una AWS CDK aplicación vacía.
`sample-app`	Crea una AWS CDK aplicación con una pila que contiene una cola de Amazon SQS y un tema de Amazon SNS.

Las plantillas utilizan el nombre de la carpeta del proyecto para generar nombres para los archivos y las clases de la nueva aplicación.

Listar pilas

Para ver una lista de los ID de las pilas de la AWS CDK aplicación, introduce uno de los siguientes comandos equivalentes:

```
cdk list
```

```
cdk ls
```

Si tu aplicación contiene pilas de CDK Pipelines, el kit de herramientas de CDK muestra los nombres de las pilas como rutas según su ubicación en la jerarquía de canalizaciones. (Por ejemplo,, `PipelineStack` y.) `PipelineStack/Prod` `PipelineStack/Prod/MyService`

Si tu aplicación contiene muchas pilas, puedes especificar los ID de pila totales o parciales de las pilas que se van a enumerar. Para obtener más información, consulte the section called “Especificar pilas”.

Agrega la `--long` marca para ver más información sobre las pilas, incluidos los nombres de las pilas y sus entornos (AWS cuenta y región).

Sintetizando pilas

El `cdk synthesize` comando (casi siempre abreviado`synth`) sintetiza una pila definida en tu aplicación en una plantilla. CloudFormation

```
cdk synth         # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"     # all stacks in app
```

> **Note**
>
> De hecho, el kit de herramientas CDK ejecuta tu aplicación y sintetiza plantillas nuevas antes de la mayoría de las operaciones (por ejemplo, al implementar o comparar pilas). Estas plantillas se almacenan de forma predeterminada en el directorio. `cdk.out` El `cdk synth` comando simplemente imprime las plantillas generadas para una o más pilas especificadas.

Consulte todas `cdk synth --help` las opciones disponibles. En la siguiente sección se describen algunas de las opciones que se utilizan con más frecuencia.

Especificar valores de contexto

Usa la `-c` opción `--context` o para pasar los valores de contexto de tiempo de ejecución a tu aplicación 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
```

Al implementar varias pilas, los valores de contexto especificados normalmente se transfieren a todas ellas. Si lo desea, puede especificar valores diferentes para cada pila anteponiendo el nombre de la pila al valor de contexto.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

Especificar el formato de visualización

De forma predeterminada, la plantilla sintetizada se muestra en formato YAML. En su lugar, añada la `--json` marca para mostrarla en formato JSON.

```
cdk synth --json MyStack
```

Especificar el directorio de salida

Agregue la opción `--output` (`-o`) para escribir las plantillas sintetizadas en un directorio que no sea`cdk.out`.

```
cdk synth --output=~/templates
```

Desplegar pilas

El `cdk deploy` subcomando despliega una o más pilas específicas en tu cuenta. AWS

```
cdk deploy        # if app contains only one stack
cdk deploy MyStack
cdk deploy Stack1 Stack2
cdk deploy "*"    # all stacks in app
```

> **Note**
>
> El kit de herramientas CDK ejecuta tu aplicación y sintetiza plantillas nuevas antes de implementarlas. AWS CloudFormation Por lo tanto, la mayoría de las opciones de línea de

comandos que puede utilizar `cdk synth` (por ejemplo,`--context`) también se pueden utilizar con. `cdk deploy`

Consulte todas `cdk deploy --help` las opciones disponibles. En la siguiente sección se describen algunas de las opciones más útiles.

Omitir la síntesis

El cdk deploy comando normalmente sintetiza las pilas de la aplicación antes de la implementación para garantizar que la implementación refleje la versión más reciente de la aplicación. Si sabes que no has cambiado el código desde la última vezcdk synth, puedes suprimir el paso de síntesis redundante durante la implementación. Para ello, especifica el `cdk.out` directorio de tu proyecto en la --app opción.

```
cdk deploy --app cdk.out StackOne StackTwo
```

Deshabilitar la reversión

AWS CloudFormation tiene la capacidad de revertir los cambios para que los despliegues sean atómicos. Esto significa que tienen éxito o fracasan en su conjunto. AWS CDK Hereda esta capacidad porque sintetiza e implementa AWS CloudFormation plantillas.

Rollback garantiza que sus recursos estén en un estado uniforme en todo momento, lo cual es vital para las pilas de producción. Sin embargo, mientras se sigue desarrollando la infraestructura, es inevitable que se produzcan algunos errores, y revertir las implementaciones fallidas puede ralentizar el proceso.

Por este motivo, el kit de herramientas del CDK le permite deshabilitar la reversión `--no-rollback` añadiéndolo a su comando. `cdk deploy` Con este indicador, las implementaciones fallidas no se anulan. En su lugar, los recursos desplegados antes que el recurso fallido permanecen en su lugar y el siguiente despliegue comienza con el recurso fallido. Pasará mucho menos tiempo esperando las implementaciones y mucho más tiempo desarrollando su infraestructura.

Intercambio en caliente

Usa la `--hotswap` marca con `cdk deploy` para intentar actualizar tus AWS recursos directamente en lugar de generar un conjunto de AWS CloudFormation cambios e implementarlo. La

implementación vuelve a la AWS CloudFormation implementación si no es posible el intercambio en caliente.

Actualmente, el intercambio en caliente admite funciones Lambda, máquinas de estado Step Functions e imágenes de contenedores de Amazon ECS. El `--hotswap` indicador también desactiva la reversión (es decir, implica). `--no-rollback`

> **Important**
>
> No se recomienda el intercambio en caliente para las implementaciones de producción.

Modo reloj

El modo de vigilancia (cdk deploy --watcho abreviatura) del kit de herramientas CDK supervisa continuamente los archivos y activos fuente de la aplicación CDK cdk watch para detectar cambios. Realiza un despliegue inmediato de las pilas especificadas cuando se detecta un cambio.

De forma predeterminada, estas implementaciones utilizan el `--hotswap` indicador, que acelera la implementación de los cambios en las funciones de Lambda. También se recurre a la implementación AWS CloudFormation si se ha cambiado la configuración de la infraestructura. Para poder realizar `cdk watch` siempre AWS CloudFormation despliegues completos, añada la `--no-hotswap` marca a`cdk watch`.

Todos los cambios realizados mientras `cdk watch` se está realizando una implementación se combinan en una sola implementación, que comienza tan pronto como se completa la implementación en curso.

El modo de vigilancia utiliza la `"watch"` clave del proyecto `cdk.json` para determinar qué archivos se van a supervisar. De forma predeterminada, estos archivos son los archivos y activos de la aplicación, pero esto se puede cambiar modificando las `"exclude"` entradas `"include"` y de la `"watch"` clave. En el siguiente `cdk.json` archivo se muestra un ejemplo de estas entradas.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
```

```
}
```

`cdk watch`ejecuta el `"build"` comando desde `cdk.json` para compilar la aplicación antes de la síntesis. Si su implementación requiere algún comando para compilar o empaquetar su código Lambda (o cualquier otra cosa que no esté en su aplicación de CDK), agréguelo aquí.

Los comodines tipo Git, `*` tanto como`**`, se pueden usar en las teclas `"watch"` y`"build"`. Cada ruta se interpreta en relación con el directorio principal de. `cdk.json` El valor predeterminado `include` es`**/*`, es decir, todos los archivos y directorios del directorio raíz del proyecto. `exclude`es opcional.

> **Important**
>
> No se recomienda el modo de vigilancia para las implementaciones de producción.

Especificar parámetros AWS CloudFormation

El AWS CDK kit de herramientas permite especificar AWS CloudFormation parámetros en el momento de la implementación. Puede proporcionarlos en la línea de comandos que sigue al `--parameters` indicador.

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

Para definir varios parámetros, utilice varios `--parameters` indicadores.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
 downloadBucketName=DownBucket
```

Si va a implementar varias pilas, puede especificar un valor diferente de cada parámetro para cada pila. Para ello, añada al nombre del parámetro el nombre de la pila y dos puntos. De lo contrario, se pasa el mismo valor a todas las pilas.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --
parameters YourStack:uploadBucketName=UpBucket
```

De forma predeterminada, AWS CDK conserva los valores de los parámetros de las implementaciones anteriores y los utiliza en las implementaciones posteriores si no se especifican de

forma explícita. Utilice la `--no-previous-parameters` marca para solicitar que se especifiquen todos los parámetros.

Especificar el archivo de salida

Si la pila declara AWS CloudFormation los resultados, normalmente se muestran en la pantalla al finalizar el despliegue. Para escribirlos en un archivo en formato JSON, usa la `--outputs-file` marca.

```
cdk deploy --outputs-file outputs.json MyStack
```

Cambios relacionados con la seguridad

Para protegerlo de los cambios no intencionados que afecten a su postura de seguridad, el AWS CDK kit de herramientas le pide que apruebe los cambios relacionados con la seguridad antes de implementarlos. Puede especificar el nivel de cambio que requiere aprobación:

```
cdk deploy --require-approval LEVEL
```

El *`NIVEL`* puede ser uno de los siguientes:

Plazo	Significado
`never`	La aprobación nunca es necesaria
`any-change`	Se requiere la aprobación de cualquier cambio o security-group-related IAM
`broadening` (predeterminado)	Requiere aprobación cuando se añaden declaraciones de IAM o normas de tráfico; las mudanzas no requieren aprobación

El ajuste también se puede configurar en el `cdk.json` archivo.

```
{
  "app": "...",
  "requireApproval": "never"
}
```

Comparación de pilas

El `cdk diff` comando compara la versión actual de una pila (y sus dependencias) definida en tu aplicación con las versiones ya implementadas o con una AWS CloudFormation plantilla guardada, y muestra una lista de cambios.

```
Stack HelloCdkStack
IAM Statement Changes
###############################################################################################
#   # Resource                    # Effect # Action                      # Principal
                  # Condition #
###############################################################################################
# + # ${Custom::S3AutoDeleteObject # Allow  # sts:AssumeRole              #
 Service:lambda.amazonaws.com #           #
#   # sCustomResourceProvider/Role #        #                           #
                  #           #
#   # .Arn}                       #        #                           #
                  #           #
###############################################################################################
# + # ${MyFirstBucket.Arn}        # Allow  # s3:DeleteObject*            # AWS:
${Custom::S3AutoDeleteOb #           #
#   # ${MyFirstBucket.Arn}/*      #        # s3:GetBucket*              #
 jectsCustomResourceProvider/ #           #
#   #                             #        # s3:GetObject*              # Role.Arn}
                  #           #
#   #                             #        # s3:List*                   #
                  #           #
###############################################################################################
IAM Policy Changes
###############################################################################################
#   # Resource                                              # Managed Policy ARN
                              #
###############################################################################################
# + # ${Custom::S3AutoDeleteObjectsCustomResourceProvider/Ro # {"Fn::Sub":"arn:
${AWS::Partition}:iam::aws:policy/serv #
#   # le}                                                   # ice-role/
AWSLambdaBasicExecutionRole"}                #
###############################################################################################
(NOTE: There may be security-related changes not in this list. See https://github.com/
aws/aws-cdk/issues/1299)

Parameters
```

```
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3Bucket
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3BucketBF7A7F3
 {"Type":"String","Description":"S3 bucket for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
S3VersionKey
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392S3VersionKeyFAF
 {"Type":"String","Description":"S3 key for asset version
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}
[+] Parameter
 AssetParameters/4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392/
ArtifactHash
 AssetParameters4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392ArtifactHashE56
 {"Type":"String","Description":"Artifact hash for asset
 \"4cd61014b71160e8c66fe167e43710d5ba068b80b134e9bd84508cf9238b2392\""}

Resources
[+] AWS::S3::BucketPolicy MyFirstBucket/Policy MyFirstBucketPolicy3243DEFD
[+] Custom::S3AutoDeleteObjects MyFirstBucket/AutoDeleteObjectsCustomResource
 MyFirstBucketAutoDeleteObjectsCustomResourceC52FCF6E
[+] AWS::IAM::Role Custom::S3AutoDeleteObjectsCustomResourceProvider/Role
 CustomS3AutoDeleteObjectsCustomResourceProviderRole3B1BD092
[+] AWS::Lambda::Function Custom::S3AutoDeleteObjectsCustomResourceProvider/Handler
 CustomS3AutoDeleteObjectsCustomResourceProviderHandler9D90184F
[~] AWS::S3::Bucket MyFirstBucket MyFirstBucketB8884501
 ## [~] DeletionPolicy
 #   ## [-] Retain
 #   ## [+] Delete
 ## [~] UpdateReplacePolicy
     ## [-] Retain
     ## [+] Delete
```

Para comparar las pilas de tu aplicación con la implementación existente, sigue estos pasos:

```
cdk diff MyStack
```

Para comparar las pilas de tu aplicación con las de una plantilla guardada CloudFormation :

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

Importación de recursos existentes en una pila

Puedes usar el `cdk import` comando para gestionar los recursos de CloudFormation una AWS CDK pila determinada. Esto resulta útil si va a AWS CDK migrar o mover recursos entre pilas o si va a cambiar su identificador lógico. `cdk import` Utiliza la importación de CloudFormation recursos. Consulta la lista de recursos que se pueden importar aquí.

Para importar un recurso existente a una AWS CDK pila, sigue los siguientes pasos:

- Asegúrese de que el recurso no esté siendo administrado actualmente por ninguna otra CloudFormation pila. Si es así, primero establezca la política de eliminación `RemovalPolicy.RETAIN` en la pila en la que se encuentra el recurso actualmente y realice una implementación. A continuación, elimine el recurso de la pila y realice otra implementación. Este proceso garantizará que el recurso deje de ser administrado por él CloudFormation , pero no lo elimine.
- Ejecute `cdk diff` a para asegurarse de que no haya cambios pendientes en la AWS CDK pila a la que desea importar los recursos. Los únicos cambios que se permiten en una operación de «importación» son la adición de nuevos recursos que desee importar.
- Añada componentes a los recursos que desee importar a su pila. Por ejemplo, si quieres importar un bucket de Amazon S3, añade algo como`new s3.Bucket(this, 'ImportedS3Bucket', {});`. No modifique ningún otro recurso.

  También debe asegurarse de modelar exactamente el estado que tiene el recurso actualmente en la definición. Para el ejemplo del segmento, asegúrate de incluir las AWS KMS claves, las políticas del ciclo de vida y cualquier otra información relevante sobre el segmento. Si no lo hace, es posible que las operaciones de actualización posteriores no den los resultados esperados.

  Puede elegir si desea incluir o no el nombre del depósito físico. Por lo general, recomendamos no incluir nombres de AWS CDK recursos en las definiciones de recursos para que sea más fácil implementar los recursos varias veces.
- Ejecute `cdk import` *`STACKNAME`*.
- Si los nombres de los recursos no están en su modelo, la CLI le pedirá que pase los nombres reales de los recursos que está importando. Después de esto, comienza la importación.
- Cuando `cdk import` los informes son correctos, el recurso ahora es administrado por AWS CDK y CloudFormation. Cualquier cambio posterior que realices en las propiedades de los recursos de tu AWS CDK aplicación, la configuración de construcción, se aplicará en la siguiente implementación.

- Para confirmar que la definición del recurso de tu AWS CDK aplicación coincide con el estado actual del recurso, puedes iniciar una operación de detección de CloudFormation desviaciones.

Actualmente, esta función no admite la importación de recursos a pilas anidadas.

Configuración () `cdk.json`

Los valores predeterminados de muchos indicadores de línea de comandos de CDK Toolkit se pueden almacenar en el `cdk.json` archivo de un proyecto o en el `.cdk.json` archivo del directorio de usuarios. A continuación se incluye una referencia alfabética a los ajustes de configuración compatibles.

Clave	Notas	Opción CDK Toolkit
`app`	El comando que ejecuta la aplicación CDK.	--app
`assetMetadata`	Si`false`, el CDK no agrega metadatos a los recursos que utilizan activos.	--no-asset-metadata
`bootstrapKmsKeyId`	Anula el ID de la AWS KMS clave utilizada para cifrar el bucket de despliegue de Amazon S3.	--bootstrap-kms-key-id
`build`	El comando que compila o crea la aplicación CDK antes de la síntesis. No se permite la entrada. `~/.cdk.json`	--build
`browser`	El comando para iniciar un navegador web para el `cdk docs` subcomando.	--browser
`context`	Consulte the section called “Context”. Los valores de contexto de un archivo de	--context

Clave	Notas	Opción CDK Toolkit
	configuración no se borrarán antes. cdk context --clear (El kit de herramientas CDK coloca los valores de contexto almacenados en caché). `cdk.context.json`	
`debug`	Si `true`, CDK Toolkit emite información más detallada útil para la depuración.	--debug
`language`	El lenguaje que se utilizará para inicializar nuevos proyectos.	--language
`lookups`	Si `false`, no se permiten búsquedas de contexto. La síntesis fallará si es necesario realizar alguna búsqueda de contexto.	--no-lookups
`notices`	Si `false`, suprime la visualización de mensajes sobre vulnerabilidades de seguridad, regresiones y versiones no compatibles.	--no-notices
`output`	El nombre del directorio en el que se emitirá el ensamblaje de nubes sintetizado (predeterminado). `"cdk.out"`	--output

Clave	Notas	Opción CDK Toolkit
`outputsFile`	El archivo en el que se escribirán los AWS CloudFormation resultados de las pilas desplegadas (en formato JSON).	--outputs-file
`pathMetadata`	Si`false`, los metadatos de la ruta CDK no se agregan a las plantillas sintetizadas.	--no-path-metadata
`plugin`	Matriz JSON que especifica los nombres de los paquetes o las rutas locales de los paquetes que amplían la CDK	--plugin
`profile`	Nombre del AWS perfil predeterminado utilizado para especificar las credenciales de la región y la cuenta.	--profile
`progress`	Si se establece en`"events"`, el kit de herramientas CDK muestra todos los AWS CloudFormation eventos durante la implementación, en lugar de una barra de progreso.	--progress
`requireApproval`	Nivel de aprobación predeterminado para los cambios de seguridad. Consulte the section called “Cambios relacionados con la seguridad”	--require-approval

Clave	Notas	Opción CDK Toolkit
`rollback`	Si `false`, las implement aciones fallidas no se revierten.	--no-rollback
`staging`	Si `false`, los activos no se copian en el directori o de salida (utilícelo para la depuración local de los archivos de origen con AWS SAM).	--no-staging
`tags`	Objeto JSON que contiene etiquetas (pares clave-valor) para la pila.	--tags
`toolkitBucketName`	El nombre del bucket de Amazon S3 utilizado para implementar activos como funciones de Lambda e imágenes de contenedores (consulte. the section called “Cómo arrancar su entorno AWS”	--toolkit-bucket-name
`toolkitStackName`	El nombre de la pila de bootstrap (consulte. the section called “Cómo arrancar su entorno AWS”	--toolkit-stack-name
`versionReporting`	Si `false`, opta por no participa r en los informes de versiones.	--no-version-reporting

Clave	Notas	Opción CDK Toolkit
`watch`	El objeto JSON contiene `"include"` `"exclude"` claves que indican qué archivos deberían (o no) activar la reconstrucción del proyecto cuando se modifique n. Consulte the section called “Modo reloj”.	--watch

Referencia de comandos de cdk migrate

Referencia para el AWS Cloud Development Kit (AWS CDK) comando Command Line Interface (CLI)`cdk migrate`. Para obtener más información sobre su uso`cdk migrate`, consulteMigre los recursos y AWS CloudFormation plantillas existentes a AWS CDK.

El `cdk migrate` comando migra los AWS recursos desplegados, las AWS CloudFormation pilas y las AWS CloudFormation plantillas locales a. AWS CDK

Temas

- Uso
- Opciones

Uso

```
$ cdk migrate <options>
```

Opciones

Opciones obligatorias

`--stack-name` *`STRING`*

El nombre de la AWS CloudFormation pila que se creará en la aplicación CDK después de la migración.

Obligatorio: sí

Opciones condicionales

`--from-path` *PATH*

La ruta a la AWS CloudFormation plantilla que se va a migrar. Proporcione esta opción para especificar una plantilla local.

Obligatorio: condicional. Obligatorio si se migra desde una AWS CloudFormation plantilla local.

`--from-scan` *STRING*

Al migrar los recursos desplegados desde un AWS entorno, utilice esta opción para especificar si se debe iniciar un nuevo análisis o si se AWS CDK CLI debe utilizar el último análisis realizado correctamente.

Obligatorio: condicional. Se requiere cuando se migra desde recursos implementados AWS .

Valores aceptados:`most-recent`, `new`

`--from-stack`

Proporcione esta opción para migrar desde una AWS CloudFormation pila implementada. Se utiliza `--stack-name` para especificar el nombre de la AWS CloudFormation pila implementada.

Obligatorio: condicional. Necesario si se migra desde una AWS CloudFormation pila implementada.

Opciones opcionales

`--account` *STRING*

La cuenta de la que se va a recuperar la plantilla de AWS CloudFormation pila.

Obligatorio: no

Predeterminada: AWS CDK CLI obtiene la información de la cuenta de las fuentes predeterminadas.

`--compress`

Proporcione esta opción para comprimir el proyecto de CDK generado en un ZIP archivo.

Obligatorio: no

`--filter` *ARRAY*

Se utiliza al migrar los recursos desplegados desde una AWS cuenta y. Región de AWS Esta opción especifica un filtro para determinar qué recursos implementados se van a migrar.

Esta opción acepta una matriz de pares clave-valor, donde la clave representa el tipo de filtro y el valor representa el valor que se va a filtrar.

Se aceptan las siguientes claves:

- `resource-identifier`— Un identificador del recurso. El valor puede ser el identificador lógico o físico del recurso. Por ejemplo, `resource-identifier="ClusterName"`.
- `resource-type-prefix`— El prefijo del tipo de AWS CloudFormation recurso. Por ejemplo, especifique filtrar todos `resource-type-prefix="AWS::DynamoDB::"` los recursos de Amazon DynamoDB.
- `tag-key`— La clave de una etiqueta de recurso. Por ejemplo, `tag-key="myTagKey"`.
- `tag-value`— El valor de una etiqueta de recurso. Por ejemplo, `tag-value="myTagValue"`.

Proporcione varios pares clave-valor para la lógica `AND` condicional. El siguiente ejemplo filtra cualquier recurso de DynamoDB que esté etiquetado como clave `myTagKey` de etiqueta:. `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

Proporcione la `--filter` opción varias veces en un solo comando para la lógica `OR` condicional. El siguiente ejemplo filtra cualquier recurso que sea un recurso de DynamoDB o que esté etiquetado como clave `myTagKey` de etiqueta:. `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

Obligatorio: no

`--language` *STRING*

El lenguaje de programación que se utilizará en el proyecto de CDK creado durante la migración.

Obligatorio: no

Valores aceptados:`typescript`,,`python`, `javacsharp`,`go`.

Valor predeterminado: `typescript`

`--output-path` *PATH*

La ruta de salida del proyecto de CDK migrado.

Obligatorio: no

Predeterminado: de forma predeterminada, AWS CDK CLI utilizará su directorio de trabajo actual.

`--region` *`STRING`*

El Región de AWS para recuperar la plantilla de AWS CloudFormation pila.

Obligatorio: no

Predeterminado: AWS CDK CLI obtiene Región de AWS información de las fuentes predeterminadas.

AWS Kit de herramientas para Visual Studio Code

El AWS Toolkit for Visual Studio Code es un complemento de código abierto para Visual Studio Code que facilita la creación, la depuración y la implementación de aplicaciones. AWS El kit de herramientas proporciona una experiencia integrada para desarrollar aplicaciones. AWS CDK Incluye la función AWS CDK Explorer para enumerar sus AWS CDK proyectos y explorar los distintos componentes de la aplicación CDK. Instale el AWS kit de herramientas y obtenga más información sobre el uso del AWS CDK Explorador.

AWS SAM integración

Los AWS CDK y los AWS Serverless Application Model (AWS SAM) pueden funcionar juntos para permitirle crear y probar localmente las aplicaciones sin servidor definidas en la CDK. Para obtener información completa, consulte la AWS Cloud Development Kit (AWS CDK)Guía para AWS SAM desarrolladores. Para instalar la CLI de SAM, consulte Instalación de la AWS SAM CLI.

Comprobación de construcciones

Con ellas AWS CDK, su infraestructura puede ser tan comprobable como cualquier otro código que escriba. El enfoque estándar para probar AWS CDK aplicaciones utiliza el módulo AWS CDK de aserciones y marcos de prueba populares como Jest para TypeScript y JavaScript Pytest para Python.

Hay dos categorías de pruebas que puedes escribir para las aplicaciones. AWS CDK

- Las afirmaciones detalladas prueban aspectos específicos de la AWS CloudFormation plantilla generada, como «este recurso tiene esta propiedad con este valor». Estas pruebas pueden detectar regresiones. También son útiles cuando se desarrollan nuevas funciones mediante un desarrollo basado en pruebas. (Puedes escribir primero una prueba y, después, hacerla pasar escribiendo una implementación correcta). Las aserciones detalladas son las pruebas que se utilizan con más frecuencia.
- Las pruebas instantáneas comparan la AWS CloudFormation plantilla sintetizada con una plantilla de referencia previamente almacenada. Las pruebas instantáneas le permiten refactorizar libremente, ya que puede estar seguro de que el código refactorizado funciona exactamente de la misma manera que el original. Si los cambios eran intencionales, puede aceptar un punto de referencia nuevo para pruebas futuras. Sin embargo, las actualizaciones de CDK también pueden provocar cambios en las plantillas sintetizadas, por lo que no puede confiar únicamente en las instantáneas para asegurarse de que la implementación es correcta.

> **Note**
>
> Las versiones completas de las TypeScript aplicaciones Python y Java utilizadas como ejemplos en este tema están disponibles en GitHub.

Introducción

Para ilustrar cómo escribir estas pruebas, crearemos una pila que contenga una máquina de AWS Step Functions estados y una AWS Lambda función. La función Lambda está suscrita a un tema de Amazon SNS y simplemente reenvía el mensaje a la máquina de estados.

En primer lugar, cree un proyecto de aplicación de CDK vacío con el kit de herramientas de CDK e instale las bibliotecas que necesitemos. Todas las construcciones que usaremos están en el paquete

principal de CDK, que es una dependencia predeterminada en los proyectos creados con el kit de herramientas de CDK. Sin embargo, debe instalar su marco de pruebas.

TypeScript

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

Cree un directorio para sus pruebas.

```
mkdir test
```

Edita los proyectos `package.json` para indicarle a NPM cómo ejecutar Jest e indicarle a Jest qué tipos de archivos recopilar. Los cambios necesarios son los siguientes.

- Añada una nueva `test` clave a la `scripts` sección
- Agrega Jest y sus tipos a la sección `devDependencies`
- Agregue una nueva clave `jest` de nivel superior con una declaración `moduleFileExtensions`

Estos cambios se muestran en el siguiente esquema. Coloque el nuevo texto donde se indica`package.json`. Los marcadores de posición «...» indican las partes existentes del archivo que no se deben cambiar.

```
{
  ...
  "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
```

Cree un directorio para sus pruebas.

```
mkdir test
```

Edita los proyectos `package.json` para indicarle a NPM cómo ejecutar Jest e indicarle a Jest qué tipos de archivos recopilar. Los cambios necesarios son los siguientes.

- Añada una nueva `test` clave a la `scripts` sección
- Agrega Jest a la sección `devDependencies`
- Agregue una nueva clave `jest` de nivel superior con una declaración `moduleFileExtensions`

Estos cambios se muestran en el siguiente esquema. Coloque el nuevo texto donde se indica`package.json`. Los marcadores de posición «...» indican las partes existentes del archivo que no se deben cambiar.

```
{
  ...
  "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
```

Abre el proyecto en el IDE de Java que prefieras. (En Eclipse, usa Archivo > Importar > Proyectos Maven existentes).

C#

```
mkdir state-machine && cd-state-machine
cdk init --language=csharp
```

Abra `src\StateMachine.sln` en Visual Studio.

Haga clic con el botón derecho en la solución en el Explorador de soluciones y elija Agregar > Nuevo proyecto. Busque MSTest C# y añada un proyecto de prueba de MSTest para C#. (El nombre predeterminado es correcto). `TestProject1`

Haga clic con el botón derecho del ratón **`TestProject1`** y seleccione Añadir > Referencia del `StateMachine` proyecto y añada el proyecto como referencia.

La pila de ejemplos

Esta es la pila que se probará en este tema. Como hemos descrito anteriormente, contiene una función Lambda y una máquina de estados de Step Functions, y acepta uno o más temas de Amazon SNS. La función Lambda está suscrita a los temas de Amazon SNS y los reenvía a la máquina de estados.

No tiene que hacer nada especial para que la aplicación sea comprobable. De hecho, esta pila de CDK no es diferente en ningún aspecto importante de las demás pilas de ejemplo de esta guía.

TypeScript

Coloque el siguiente código en: `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

Introduce el siguiente código en`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

Introduce el siguiente código en`state_machine/state_machine_stack.py`:

```
from typing import List
```

```
import aws_cdk.aws_lambda as lambda_
import aws_cdk.aws_sns as sns
import aws_cdk.aws_sns_subscriptions as sns_subscriptions
import aws_cdk.aws_stepfunctions as sfn
import aws_cdk as cdk

class StateMachineStack(cdk.Stack):
    def __init__(
        self,
        scope: cdk.Construct,
        construct_id: str,
        *,
        topics: List[sns.Topic],
        **kwargs
    ) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # In the future this state machine will do some work...
        state_machine = sfn.StateMachine(
            self, "StateMachine", definition=sfn.Pass(self, "StartState")
        )

        # This Lambda function starts the state machine.
        func = lambda_.Function(
            self,
            "LambdaFunction",
            runtime=lambda_.Runtime.NODEJS_18_X,
            handler="handler",
            code=lambda_.Code.from_asset("./start-state-machine"),
            environment={
                "STATE_MACHINE_ARN": state_machine.state_machine_arn,
            },
        )
        state_machine.grant_start_execution(func)

        subscription = sns_subscriptions.LambdaSubscription(func)
        for topic in topics:
            topic.add_subscription(subscription)
```

Java

```
package software.amazon.samples.awscdkassertionssamples;
```

```
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.sns.ITopicSubscription;
import software.amazon.awscdk.services.sns.Topic;
import software.amazon.awscdk.services.sns.subscriptions.LambdaSubscription;
import software.amazon.awscdk.services.stepfunctions.Pass;
import software.amazon.awscdk.services.stepfunctions.StateMachine;

import java.util.Collections;
import java.util.List;

public class StateMachineStack extends Stack {
    public StateMachineStack(final Construct scope, final String id, final
 List<Topic> topics) {
        this(scope, id, null, topics);
    }

    public StateMachineStack(final Construct scope, final String id, final
 StackProps props, final List<Topic> topics) {
        super(scope, id, props);

        // In the future this state machine will do some work...
        final StateMachine stateMachine = StateMachine.Builder.create(this,
 "StateMachine")
                .definition(new Pass(this, "StartState"))
                .build();

        // This Lambda function starts the state machine.
        final Function func = Function.Builder.create(this, "LambdaFunction")
                .runtime(Runtime.NODEJS_18_X)
                .handler("handler")
                .code(Code.fromAsset("./start-state-machine"))
                .environment(Collections.singletonMap("STATE_MACHINE_ARN",
 stateMachine.getStateMachineArn()))
                .build();
        stateMachine.grantStartExecution(func);

        final ITopicSubscription subscription = new LambdaSubscription(func);
        for (final Topic topic : topics) {
```

```
            topic.addSubscription(subscription);
        }
    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.StepFunctions;
using Amazon.CDK.AWS.SNS;
using Amazon.CDK.AWS.SNS.Subscriptions;
using Constructs;

using System.Collections.Generic;

namespace AwsCdkAssertionSamples
{
    public class StateMachineStackProps : StackProps
    {
        public Topic[] Topics;
    }

    public class StateMachineStack : Stack
    {

        internal StateMachineStack(Construct scope, string id,
 StateMachineStackProps props = null) : base(scope, id, props)
        {
            // In the future this state machine will do some work...
            var stateMachine = new StateMachine(this, "StateMachine", new
 StateMachineProps
            {
                Definition = new Pass(this, "StartState")
            });

            // This Lambda function starts the state machine.
            var func = new Function(this, "LambdaFunction", new FunctionProps
            {
                Runtime = Runtime.NODEJS_18_X,
                Handler = "handler",
                Code = Code.FromAsset("./start-state-machine"),
                Environment = new Dictionary<string, string>
```

```
                {
                    { "STATE_MACHINE_ARN", stateMachine.StateMachineArn }
                }
            });
            stateMachine.GrantStartExecution(func);

            foreach (Topic topic in props?.Topics ?? new Topic[0])
            {
                var subscription = new LambdaSubscription(func);
            }

        }
    }
}
```

Modificaremos el punto de entrada principal de la aplicación para que en realidad no instanciemos nuestra pila. No queremos desplegarla accidentalmente. Nuestras pruebas crearán una aplicación y una instancia de la pila para probarlas. Se trata de una táctica útil cuando se combina con un desarrollo basado en pruebas: asegúrate de que la pila supere todas las pruebas antes de habilitar el despliegue.

TypeScript

En `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

En `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

En `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 función Lambda

Nuestra pila de ejemplos incluye una función Lambda que inicia nuestra máquina de estados. Debemos proporcionar el código fuente de esta función para que la CDK pueda empaquetarla e implementarla como parte de la creación del recurso de la función Lambda.

- Crea la carpeta `start-state-machine` en el directorio principal de la aplicación.
- En esta carpeta, crea al menos un archivo. Por ejemplo, puede guardar el siguiente código en`start-state-machines/index.js`.

  ```
  exports.handler = async function (event, context) {
   return 'hello world';
  };
  ```

  Sin embargo, cualquier archivo funcionará, ya que en realidad no vamos a implementar la pila.

Ejecución de pruebas

Como referencia, estos son los comandos que usas para ejecutar pruebas en tu AWS CDK aplicación. Estos son los mismos comandos que utilizarías para ejecutar las pruebas en cualquier proyecto que utilice el mismo marco de pruebas. En el caso de los lenguajes que requieren un paso de compilación, inclúyelo para asegurarte de que las pruebas se hayan compilado.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

Cree su solución (F6) para descubrir las pruebas y, a continuación, ejecútelas (Probar > Ejecutar todas las pruebas). Para elegir qué pruebas ejecutar, abra el Explorador de pruebas (Prueba > Explorador de pruebas).

O bien:

```
dotnet test src
```

Afirmaciones detalladas

El primer paso para probar una pila con aserciones detalladas es sintetizar la pila, ya que estamos escribiendo las afirmaciones en función de la plantilla generada. AWS CloudFormation

El nuestro `StateMachineStackStack` requiere que le pasemos el tema de Amazon SNS para que se reenvíe a la máquina de estados. Por eso, en nuestra prueba, crearemos una pila separada para incluir el tema.

Normalmente, cuando escribes una aplicación de CDK, puedes subclasificar `Stack` e instanciar el tema Amazon SNS en el constructor de la pila. En nuestra prueba, instanciamos `Stack` directamente, luego pasamos esta pila como ámbito de aplicación y la adjuntamos a `Topic` la pila.

Esto es funcionalmente equivalente y menos detallado. También ayuda a que las pilas que se utilizan solo en las pruebas tengan un aspecto diferente al de las pilas que se van a implementar.

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
        }
    }
}
```

Ahora podemos afirmar que se crearon la función Lambda y la suscripción a Amazon SNS.

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);
```

Nuestra prueba de función Lambda afirma que dos propiedades particulares del recurso de la función tienen valores específicos. De forma predeterminada, el `hasResourceProperties` método realiza una coincidencia parcial con las propiedades del recurso, tal como se indica en la plantilla sintetizada CloudFormation . Esta prueba requiere que las propiedades proporcionadas existan y tengan los valores especificados, pero el recurso también puede tener otras propiedades, que no se prueban.

Nuestra afirmación de Amazon SNS afirma que la plantilla sintetizada contiene una suscripción, pero nada sobre la suscripción en sí. Incluimos esta afirmación principalmente para ilustrar cómo hacer valer la cantidad de recursos. La `Template` clase ofrece métodos más específicos para escribir afirmaciones en las `Mapping` secciones `ResourcesOutputs`, y de la CloudFormation plantilla.

Comparadores

El comportamiento de coincidencia parcial predeterminado de se `hasResourceProperties` puede cambiar utilizando comparadores de la `Match`clase.

Los comparadores van desde indulgente (`Match.anyValue`) hasta estricta (). `Match.objectEquals` Se pueden anidar para aplicar diferentes métodos de coincidencia a diferentes partes de las propiedades del recurso. Al `Match.objectEquals` utilizarlos `Match.anyValue` juntos, por ejemplo, podemos probar más exhaustivamente la función de IAM de la máquina de estados, sin requerir valores específicos para las propiedades que puedan cambiar.

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" }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }));
```

Muchos CloudFormation recursos incluyen objetos JSON serializados representados como cadenas. El `Match.serializedJson()` comparador se puede usar para hacer coincidir las propiedades de este JSON.

Por ejemplo, las máquinas de estados Step Functions se definen mediante una cadena en el lenguaje Amazon States Language basado en JSON. Lo usaremos `Match.serializedJson()` para asegurarnos de que nuestro estado inicial sea el único paso. Nuevamente, usaremos comparadores anidados para aplicar diferentes tipos de coincidencias a diferentes partes del objeto.

TypeScript

```
    // Assert on the state machine's definition with the Match.serializedJson()
    // matcher.
```

```
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    // Match.objectEquals() is used implicitly, but we use it explicitly
    // here for extra clarity.
    Match.objectEquals({
      StartAt: "StartState",
      States: {
        StartState: {
          Type: "Pass",
          End: true,
          // Make sure this state doesn't provide a next state -- we can't
          // provide both Next and set End to true.
          Next: Match.absent(),
        },
      },
    })
  ),
});
```

JavaScript

```
// Assert on the state machine's definition with the Match.serializedJson()
// matcher.
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    // Match.objectEquals() is used implicitly, but we use it explicitly
    // here for extra clarity.
    Match.objectEquals({
      StartAt: "StartState",
      States: {
        StartState: {
          Type: "Pass",
          End: true,
          // Make sure this state doesn't provide a next state -- we can't
          // provide both Next and set End to true.
          Next: Match.absent(),
        },
      },
    })
  ),
});
```

Python

```
   # Assert on the state machine's definition with the serialized_json matcher.
   template.has_resource_properties(
       "AWS::StepFunctions::StateMachine",
       {
           "DefinitionString": Match.serialized_json(
               # Match.object_equals() is the default, but specify it here for
clarity
               Match.object_equals(
                   {
                       "StartAt": "StartState",
                       "States": {
                           "StartState": {
                               "Type": "Pass",
                               "End": True,
                               # Make sure this state doesn't provide a next state
--
                               # we can't provide both Next and set End to true.
                               "Next": Match.absent(),
                           },
                       },
                   }
               )
           ),
       },
   )
```

Java

```
       // Assert on the state machine's definition with the Match.serializedJson()
matcher.
       template.hasResourceProperties("AWS::StepFunctions::StateMachine",
Collections.singletonMap(
               "DefinitionString", Match.serializedJson(
                       // Match.objectEquals() is used implicitly, but we use it
explicitly here for extra clarity.
                       Match.objectEquals(Map.of(
                               "StartAt", "StartState",
                               "States", Collections.singletonMap(
                                       "StartState", Map.of(
                                               "Type", "Pass",
                                               "End", true,
```

```
                                                // Make sure this state doesn't
provide a next state -- we can't provide
                                                // both Next and set End to true.
                                                "Next", Match.absent()
                                            )
                                    )
                            ))
                    )
            ));
```

C#

```
            // Assert on the state machine's definition with the
Match.serializedJson() matcher
            template.HasResourceProperties("AWS::StepFunctions::StateMachine", new
ObjectDict
            {
                { "DefinitionString", Match.SerializedJson(
                    // Match.objectEquals() is used implicitly, but we use it
explicitly here for extra clarity.
                    Match.ObjectEquals(new ObjectDict {
                        { "StartAt", "StartState" },
                        { "States", new ObjectDict
                        {
                            { "StartState", new ObjectDict {
                                { "Type", "Pass" },
                                { "End", "True" },
                                // Make sure this state doesn't provide a next state
-- we can't provide
                                // both Next and set End to true.
                                { "Next", Match.Absent() }
                            }}
                        }}
                    })
                )}});
```

Capturando

Suele ser útil probar las propiedades para asegurarse de que siguen formatos específicos o que tienen el mismo valor que otra propiedad, sin necesidad de conocer sus valores exactos con antelación. El `assertions` módulo proporciona esta capacidad en su `Capture`clase.

Al especificar una `Capture` instancia en lugar de un valor en`hasResourceProperties`, ese valor se conserva en el `Capture` objeto. El valor capturado real se puede recuperar mediante los `as` métodos del objeto, incluidos `asNumber()``asString()`, `yasObject`, y someterlo a pruebas. `Capture`Utilícelo con un comparador para especificar la ubicación exacta del valor que se va a capturar dentro de las propiedades del recurso, incluidas las propiedades JSON serializadas.

El siguiente ejemplo comprueba que el estado inicial de nuestra máquina de estados tenga un nombre que comience por. `Start` También comprueba que este estado esté presente en la lista de estados de la máquina.

TypeScript

```
// Capture some data from the state machine's definition.
const startAtCapture = new Capture();
const statesCapture = new Capture();
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    Match.objectLike({
      StartAt: startAtCapture,
      States: statesCapture,
    })
  ),
});

// Assert that the start state starts with "Start".
expect(startAtCapture.asString()).toEqual(expect.stringMatching(/^Start/));

// Assert that the start state actually exists in the states object of the
// state machine definition.
expect(statesCapture.asObject()).toHaveProperty(startAtCapture.asString());
```

JavaScript

```
// Capture some data from the state machine's definition.
const startAtCapture = new Capture();
const statesCapture = new Capture();
template.hasResourceProperties("AWS::StepFunctions::StateMachine", {
  DefinitionString: Match.serializedJson(
    Match.objectLike({
      StartAt: startAtCapture,
      States: statesCapture,
    })
```

```
      ),
    });

    // Assert that the start state starts with "Start".
    expect(startAtCapture.asString()).toEqual(expect.stringMatching(/^Start/));

    // Assert that the start state actually exists in the states object of the
    // state machine definition.
    expect(statesCapture.asObject()).toHaveProperty(startAtCapture.asString());
```

Python

```
import re

    from aws_cdk.assertions import Capture

    # ...

    # Capture some data from the state machine's definition.
    start_at_capture = Capture()
    states_capture = Capture()
    template.has_resource_properties(
        "AWS::StepFunctions::StateMachine",
        {
            "DefinitionString": Match.serialized_json(
                Match.object_like(
                    {
                        "StartAt": start_at_capture,
                        "States": states_capture,
                    }
                )
            ),
        },
    )

    # Assert that the start state starts with "Start".
    assert re.match("^Start", start_at_capture.as_string())

    # Assert that the start state actually exists in the states object of the
    # state machine definition.
    assert start_at_capture.as_string() in states_capture.as_object()
```

Java

```
        // Capture some data from the state machine's definition.
        final Capture startAtCapture = new Capture();
        final Capture statesCapture = new Capture();
        template.hasResourceProperties("AWS::StepFunctions::StateMachine",
Collections.singletonMap(
                "DefinitionString", Match.serializedJson(
                        Match.objectLike(Map.of(
                                "StartAt", startAtCapture,
                                "States", statesCapture
                        ))
                )
        ));

        // Assert that the start state starts with "Start".
        assertThat(startAtCapture.asString()).matches("^Start.+");

        // Assert that the start state actually exists in the states object of the
state machine definition.
        assertThat(statesCapture.asObject()).containsKey(startAtCapture.asString());
```

C#

```
            // Capture some data from the state machine's definition.
            var startAtCapture = new Capture();
            var statesCapture = new Capture();
            template.HasResourceProperties("AWS::StepFunctions::StateMachine", new
ObjectDict
            {
                { "DefinitionString", Match.SerializedJson(
                    new ObjectDict
                    {
                        { "StartAt", startAtCapture },
                        { "States", statesCapture }
                    }
                )}
            });

            Assert.IsTrue(startAtCapture.ToString().StartsWith("Start"));

Assert.IsTrue(statesCapture.AsObject().ContainsKey(startAtCapture.ToString()));
```

Pruebas instantáneas

En las pruebas de instantáneas, se compara toda la CloudFormation plantilla sintetizada con una plantilla de referencia previamente almacenada (a menudo denominada «maestra»). A diferencia de las afirmaciones detalladas, las pruebas instantáneas no son útiles para detectar regresiones. Esto se debe a que las pruebas instantáneas se aplican a toda la plantilla y, además de los cambios en el código, pueden provocar pequeñas (o not-so-small) diferencias en los resultados de la síntesis. Es posible que estos cambios ni siquiera afecten a la implementación, pero aun así provocarán que una prueba de instantáneas falle.

Por ejemplo, puede actualizar una construcción de CDK para incorporar una nueva práctica recomendada, lo que puede provocar cambios en los recursos sintetizados o en la forma en que están organizados. Como alternativa, puede actualizar el kit de herramientas del CDK a una versión que incluya metadatos adicionales. Los cambios en los valores de contexto también pueden afectar a la plantilla sintetizada.

Sin embargo, las pruebas instantáneas pueden ser de gran ayuda a la hora de refactorizar, siempre y cuando se mantengan constantes todos los demás factores que puedan afectar a la plantilla sintetizada. Sabrá inmediatamente si un cambio realizado ha modificado la plantilla de forma involuntaria. Si el cambio es intencionado, simplemente acepte la nueva plantilla como referencia.

Por ejemplo, si tenemos esta `DeadLetterQueue` construcción:

TypeScript

```
export class DeadLetterQueue extends sqs.Queue {
  public readonly messagesInQueueAlarm: cloudwatch.IAlarm;

  constructor(scope: Construct, id: string) {
    super(scope, id);

    // Add the alarm
    this.messagesInQueueAlarm = new cloudwatch.Alarm(this, 'Alarm', {
      alarmDescription: 'There are messages in the Dead Letter Queue',
      evaluationPeriods: 1,
      threshold: 1,
      metric: this.metricApproximateNumberOfMessagesVisible(),
    });
  }
}
```

JavaScript

```
class DeadLetterQueue extends sqs.Queue {

  constructor(scope, id) {
    super(scope, id);

    // Add the alarm
    this.messagesInQueueAlarm = new cloudwatch.Alarm(this, 'Alarm', {
      alarmDescription: 'There are messages in the Dead Letter Queue',
      evaluationPeriods: 1,
      threshold: 1,
      metric: this.metricApproximateNumberOfMessagesVisible(),
    });
  }
}

module.exports = { DeadLetterQueue }
```

Python

```
class DeadLetterQueue(sqs.Queue):
    def __init__(self, scope: Construct, id: str):
        super().__init__(scope, id)

        self.messages_in_queue_alarm = cloudwatch.Alarm(
            self,
            "Alarm",
            alarm_description="There are messages in the Dead Letter Queue.",
            evaluation_periods=1,
            threshold=1,
            metric=self.metric_approximate_number_of_messages_visible(),
        )
```

Java

```
public class DeadLetterQueue extends Queue {
    private final IAlarm messagesInQueueAlarm;

    public DeadLetterQueue(@NotNull Construct scope, @NotNull String id) {
        super(scope, id);

        this.messagesInQueueAlarm = Alarm.Builder.create(this, "Alarm")
```

```
                .alarmDescription("There are messages in the Dead Letter Queue.")
                .evaluationPeriods(1)
                .threshold(1)
                .metric(this.metricApproximateNumberOfMessagesVisible())
                .build();
    }

    public IAlarm getMessagesInQueueAlarm() {
        return messagesInQueueAlarm;
    }
}
```

C#

```
namespace AwsCdkAssertionSamples
{
    public class DeadLetterQueue : Queue
    {
        public IAlarm messagesInQueueAlarm;

        public DeadLetterQueue(Construct scope, string id) : base(scope, id)
        {
            messagesInQueueAlarm = new Alarm(this, "Alarm", new AlarmProps
            {
                AlarmDescription = "There are messages in the Dead Letter Queue.",
                EvaluationPeriods = 1,
                Threshold = 1,
                Metric = this.MetricApproximateNumberOfMessagesVisible()
            });
        }
    }
}
```

Podemos probarlo así:

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());
        }
    }
}
```

Consejos para las pruebas

Recuerde que sus pruebas durarán tanto como el código que prueben, y se leerán y modificarán con la misma frecuencia. Por lo tanto, vale la pena tomarse un momento para considerar la mejor manera de escribirlas.

No copies ni pegues líneas de configuración ni afirmaciones comunes. En su lugar, refactoriza esta lógica en accesorios o funciones auxiliares. Usa buenos nombres que reflejen lo que realmente evalúa cada prueba.

No intentes hacer demasiado en una sola prueba. Preferiblemente, una prueba debe evaluar una y solo una conducta. Si accidentalmente rompes ese comportamiento, exactamente una prueba debería fallar y el nombre de la prueba debería indicar qué es lo que falló. Sin embargo, lo ideal es esforzarse por lograrlo; a veces, de manera inevitable (o inadvertida), escribirás pruebas que evalúan más de un comportamiento. Las pruebas instantáneas son, por las razones que ya hemos descrito, especialmente propensas a este problema, así que úsalas con moderación.

Seguridad para AWS Cloud Development Kit (AWS CDK)

La seguridad en la nube de Amazon Web Services (AWS) es la máxima prioridad. Como AWS cliente, usted se beneficia de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad. La seguridad es una responsabilidad compartida entre usted AWS y usted. En el modelo de responsabilidad compartida, se habla de "seguridad de la nube" y "seguridad en la nube":

Seguridad de la nube: AWS se encarga de proteger la infraestructura en la que se ejecutan todos los servicios que se ofrecen en la AWS nube y de proporcionarle servicios que pueda utilizar de forma segura. Nuestra responsabilidad en materia de seguridad es nuestra máxima prioridad AWS, y auditores externos comprueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los programas de AWS conformidad.

Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice y otros factores, como la confidencialidad de sus datos, los requisitos de su organización y las leyes y reglamentos aplicables.

AWS CDK Sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Temas

- Administración de identidad y acceso para el AWS Cloud Development Kit (AWS CDK)
- Validación de conformidad para AWS Cloud Development Kit (AWS CDK)
- Resiliencia para AWS Cloud Development Kit (AWS CDK)
- Seguridad de infraestructura para AWS Cloud Development Kit (AWS CDK)

Administración de identidad y acceso para el AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a AWS los recursos. Los administradores de

IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se realice. AWS

Usuario del servicio: si Servicios de AWS solía hacer su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador.

Administrador de servicios: si está a cargo de AWS los recursos de su empresa, probablemente tenga acceso total a AWS los recursos. Su trabajo consiste en determinar a qué Servicios de AWS recursos deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a AWS.

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (IAM Identity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, asumes un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Para acceder AWS mediante programación, AWS proporciona kits de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. AWS CDK Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre el uso del método recomendado para firmar las solicitudes usted mismo, consulte Proceso de firma de Signature Version 4 en la Referencia general de AWS.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Uso de la autenticación multifactor (MFA) en AWSen la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidades web AWS Directory Service, el directorio del Centro de Identidad o cualquier usuario al que acceda Servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS Single Sign-On. Puede crear usuarios y grupos en el Centro de identidades de IAM o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en

todas sus Cuentas de AWS aplicaciones. Para obtener más información, consulte ¿Qué es el Centro de identidades de IAM? en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un usuario de IAM es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales. Para más información, consulte Cuándo crear un usuario de IAM (en lugar de un rol) en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una persona determinada. Puede asumir temporalmente una función de IAM en el AWS Management Console cambiando de función. Puede asumir un rol llamando a una operación de AWS API AWS CLI o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte Uso de roles de IAM en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para

federación, consulte Creación de un rol para un proveedor de identidades de terceros en la Guía del usuario de IAM. Si utiliza IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS Single Sign-On.

- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunas Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.
  - Rol de servicio: un rol de servicio es un rol de IAM que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
  - Función vinculada a un servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.
- Aplicaciones que se ejecutan en Amazon EC2: puede usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una instancia EC2 y realizan AWS CLI solicitudes a la API. AWS Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia de EC2. Para asignar una AWS función a una instancia EC2 y ponerla a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia de EC2 obtener

credenciales temporales. Para más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2 en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte Cuándo crear un rol de IAM (en lugar de un usuario) en la Guía del usuario de IAM.

Validación de conformidad para AWS Cloud Development Kit (AWS CDK)

AWS CDK Sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Auditores externos evalúan la seguridad y el cumplimiento de los AWS servicios como parte de varios programas de AWS cumplimiento. Estos incluyen SOC, PCI, FedRAMP, HIPAA y otros. AWS proporciona una lista de servicios que se actualiza con frecuencia dentro del ámbito de los programas de cumplimiento específicos en AWS Services in Scope by Compliance AWS Program.

Los informes de auditoría de terceros están disponibles para su descarga mediante AWS Artifact. Para obtener más información, consulte Descargar informes en AWS Artifact.

Para obtener más información sobre los programas de AWS conformidad, consulte Programas AWS de conformidad.

Su responsabilidad de cumplimiento al utilizar el servicio AWS CDK para acceder a un AWS servicio viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su organización y las leyes y reglamentos aplicables. Si el uso de un AWS servicio está sujeto al cumplimiento de normas como HIPAA, PCI o FedRAMP, proporciona recursos para ayudarlo a: AWS

- Guías de inicio rápido sobre seguridad y conformidad: guías de implementación en las que se analizan las consideraciones arquitectónicas y se proporcionan los pasos necesarios para implementar entornos básicos centrados en la seguridad y el cumplimiento. AWS
- AWS Recursos de conformidad: una colección de libros de trabajo y guías que pueden aplicarse a su sector y ubicación.

- AWS Config: un servicio que evalúa en qué medida las configuraciones de los recursos cumplen las prácticas internas, las directrices del sector y la normativa.
- AWS Security Hub— Una visión completa del estado de su seguridad AWS que le ayuda a comprobar su conformidad con los estándares y las mejores prácticas del sector de la seguridad.

Resiliencia para AWS Cloud Development Kit (AWS CDK)

La infraestructura global de Amazon Web Services (AWS) se basa en AWS regiones y zonas de disponibilidad.

AWS Las regiones proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia.

Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre zonas de disponibilidad sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS

AWS CDK Sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Seguridad de infraestructura para AWS Cloud Development Kit (AWS CDK)

AWS CDK Sigue el modelo de responsabilidad compartida a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los AWS servicios, consulte la página de documentación sobre la seguridad del AWS servicio y AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad.

Solución de AWS CDK problemas comunes

En este tema se describe cómo solucionar los siguientes problemas con el AWS CDK.

Tras actualizar el AWS CDK, el AWS CDK kit de herramientas (CLI) informa de una discordancia con la biblioteca Construct AWS

La versión del AWS CDK kit de herramientas (que proporciona el `cdk` comando) debe ser al menos igual a la versión del módulo principal de AWS Construct Library,. `aws-cdk-lib` El kit de herramientas está diseñado para ser compatible con versiones anteriores. La última versión 2.x del kit de herramientas se puede utilizar con cualquier versión 1.x o 2.x de la biblioteca. Por este motivo, le recomendamos que instale este componente globalmente y lo mantenga actualizado.

```
npm update -g aws-cdk
```

Si necesita trabajar con varias versiones del kit de AWS CDK herramientas, instale una versión específica del kit de herramientas de forma local en la carpeta de su proyecto.

Si está utilizando TypeScript o JavaScript, el directorio de su proyecto ya contiene una copia local versionada del kit de herramientas del CDK.

Si utiliza otro idioma, úselo `npm` para instalar el AWS CDK kit de herramientas, omitiendo la `-g` marca y especificando la versión deseada. Por ejemplo:

```
npm install aws-cdk@2.0
```

Para ejecutar un AWS CDK kit de herramientas instalado localmente, utilice el comando `npx aws-cdk` en lugar de solo. `cdk` Por ejemplo:

```
npx aws-cdk deploy MyStack
```

`npx aws-cdk`ejecuta la versión local del AWS CDK kit de herramientas, si existe alguna. Recurre a la versión global cuando un proyecto no tiene una instalación local. Puede que le resulte conveniente configurar un alias de shell para asegurarse de que siempre `cdk` se invoque de esta manera.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

(volver a la lista)

Al implementar mi AWS CDK pila, recibo un **`NoSuchBucket`** error

Su AWS entorno no se ha iniciado y, por lo tanto, no tiene un bucket de Amazon S3 para almacenar los recursos durante la implementación. Puede crear el depósito provisional y otros recursos necesarios con el siguiente comando:

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

Para evitar generar AWS cargos inesperados, AWS CDK no inicia automáticamente ningún entorno. Debe iniciar de forma explícita cada entorno en el que vaya a realizar la implementación.

De forma predeterminada, los recursos de arranque se crean en la región o regiones que utilizan las pilas de la aplicación actual. AWS CDK Como alternativa, se crean en la región especificada

en su AWS perfil local (establecida por`aws configure`), utilizando la cuenta de ese perfil. Puede especificar una cuenta y una región diferentes en la línea de comandos de la siguiente manera. (Debe especificar la cuenta y la región si no se encuentra en el directorio de una aplicación).

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

Para obtener más información, consulte the section called “Bootstrapping (Proceso de arranque)”.

(volver a la lista)

Al implementar mi AWS CDK pila, recibo un **`forbidden: null`** mensaje

Está implementando una pila que requiere recursos de arranque, pero está utilizando un rol o una cuenta de IAM que no tiene permiso para escribir en ella. (El segmento provisional se usa cuando se implementan pilas que contienen activos o que sintetizan una AWS CloudFormation plantilla de más de 50 000). Usa una cuenta o un rol que tenga permiso para realizar la acción `s3:*` contra el depósito mencionado en el mensaje de error.

(volver a la lista)

Al sintetizar una AWS CDK pila, recibo el mensaje **`--app is required either in command-line, in cdk.json or in ~/.cdk.json`**

Este mensaje normalmente significa que no estás en el directorio principal de tu AWS CDK proyecto cuando lo publicas. `cdk synth` El archivo `cdk.json` de este directorio, creado por el `cdk init` comando, contiene la línea de comandos necesaria para ejecutar (y, por lo tanto, sintetizar) AWS CDK la aplicación. En el caso de una TypeScript aplicación, por ejemplo, el valor `cdk.json` predeterminado es el siguiente:

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

Te recomendamos que `cdk` emitas comandos solo en el directorio principal de tu proyecto, de modo que el AWS CDK kit de herramientas pueda `cdk.json` encontrarlos allí y ejecutar tu aplicación correctamente.

Si esto no resulta práctico por alguna razón, el AWS CDK kit de herramientas busca la línea de comandos de la aplicación en otras dos ubicaciones:

- `cdk.json`En tu directorio principal
- En el propio `cdk synth` comando usando la `-a` opción

Por ejemplo, puedes sintetizar una pila de una TypeScript aplicación de la siguiente manera.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

(volver a la lista)

Al sintetizar una AWS CDK pila, recibo un error porque la AWS CloudFormation plantilla contiene demasiados recursos

AWS CDK Genera e implementa AWS CloudFormation plantillas. AWS CloudFormation tiene un límite estricto en la cantidad de recursos que puede contener una pila. Con el AWS CDK, puedes superar este límite más rápido de lo que cabría esperar.

> **Note**
>
> El límite AWS CloudFormation de recursos es de 500 al momento de escribir este artículo. Consulte AWS CloudFormation las cuotas para el límite de recursos actual.

Las estructuras de nivel superior basadas en la intención de la biblioteca AWS Construct proporcionan automáticamente todos los recursos auxiliares necesarios para el registro, la administración de claves, la autorización y otros fines. Por ejemplo, si se concede acceso a un recurso a otro, se generan todos los objetos de IAM necesarios para que los servicios pertinentes se comuniquen.

Según nuestra experiencia, el uso real de construcciones basadas en la intención da como resultado de 1 a 5 AWS CloudFormation recursos por construcción, aunque esto puede variar. En el caso de las aplicaciones sin servidor, lo habitual es disponer de 5 a 8 recursos por punto final AWS de la API.

Los patrones, que representan un mayor nivel de abstracción, permiten definir incluso más AWS recursos con incluso menos código. ¡El AWS CDK código dethe section called “ECS”, por ejemplo, genera más de 50 AWS CloudFormation recursos y define solo tres construcciones!

Superar el límite AWS CloudFormation de recursos es un error durante la AWS CloudFormation síntesis. AWS CDK Emite una advertencia si tu pila supera el 80% del límite. Puedes usar un

límite diferente configurando la `maxResources` propiedad en tu pila, o deshabilitar la validación `maxResources` configurándola en 0.

> **Tip**
>
> Puede obtener un recuento exacto de los recursos de la salida sintetizada mediante el siguiente script de utilidades. (Como todos los AWS CDK desarrolladores necesitan Node.js, el script está escrito en él) JavaScript.

```
// rescount.js - count the resources defined in a stack
// invoke with: node rescount.js <path-to-stack-json>
// e.g. node rescount.js cdk.out/MyStack.template.json

import * as fs from 'fs';
const path = process.argv[2];

if (path) fs.readFile(path, 'utf8', function(err, contents) {
  console.log(err ? `${err}` :
  `${Object.keys(JSON.parse(contents).Resources).length} resources defined in
 ${path}`);
}); else console.log("Please specify the path to the stack's output .json
 file");
```

A medida que el recuento de recursos de su pila se acerque al límite, considere la posibilidad de rediseñar la arquitectura para reducir la cantidad de recursos que contiene su pila: por ejemplo, combinando algunas funciones de Lambda o dividiendo la pila en varias pilas. El CDK admite referencias entre pilas, por lo que puedes separar la funcionalidad de tu aplicación en pilas diferentes de la forma que más te convenga.

> **Note**
>
> AWS CloudFormation los expertos suelen sugerir el uso de pilas anidadas como solución al límite de recursos. AWS CDK Apoya este enfoque a través del `NestedStack`constructo.

(volver a la lista)

Especificé tres (o más) zonas de disponibilidad para mi grupo de Auto Scaling o VPC, pero solo se implementó en dos

Para obtener el número de zonas de disponibilidad que solicita, especifique la cuenta y la región en la `env` propiedad de la pila. Si no especifica ambas, de forma predeterminada AWS CDK, sintetiza la pila como independiente del entorno. A continuación, puede implementar la pila en una región específica utilizando. AWS CloudFormation Como algunas regiones solo tienen dos zonas de disponibilidad, una plantilla independiente del entorno no utiliza más de dos.

> **Note**
>
> En el pasado, las regiones se lanzaban ocasionalmente con una sola zona de disponibilidad. AWS CDK Las pilas independientes del entorno no se pueden implementar en dichas regiones. Sin embargo, al momento de escribir este artículo, todas AWS las regiones tienen al menos dos zonas de disponibilidad.

Puedes cambiar este comportamiento anulando la propiedad de tu pila `availablilityZones`(Python:`availability_zones`) para especificar de forma explícita las zonas que quieres usar.

Para obtener más información sobre cómo especificar la cuenta y la región de una pila en el momento de la síntesis y, al mismo tiempo, conservar la flexibilidad de implementarla en cualquier región, consultethe section called “Entornos”.

(volver a la lista)

Mi bucket de S3, tabla de DynamoDB u otro recurso no se eliminan cuando ejecuto el error **`cdk destroy`**

De forma predeterminada, los recursos que pueden contener datos de usuario tienen la propiedad `removalPolicy` (Python:`removal_policy`) de`RETAIN`, y el recurso no se elimina cuando se destruye la pila. En su lugar, el recurso queda huérfano de la pila. A continuación, debes eliminar el recurso manualmente una vez que se destruya la pila. Hasta que lo hagas, se producirá un error al volver a desplegar la pila. Esto se debe a que el nombre del nuevo recurso que se está creando durante la implementación entra en conflicto con el nombre del recurso huérfano.

Si estableces la política de eliminación de un recurso en`DESTROY`, ese recurso se eliminará cuando se destruya la pila.

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 no puede eliminar un bucket de Amazon S3 que no esté vacío. Si estableces una política de retirada de un bucket de Amazon S3 y este contiene datos, el intento de destruir la pila fallará porque no se puede eliminar el bucket. `DESTROY` Puedes hacer que AWS CDK elimines los objetos del depósito antes de intentar destruirlo. Para ello, coloca el puntal del `autoDeleteObjects` depósito en. `true`

(volver a la lista)

Claves OpenPGP para y jsii AWS CDK

Este tema contiene las claves OpenPGP actuales e históricas para el jsii y el jsii. AWS CDK

Claves actuales

Estas claves deben usarse para validar las versiones actuales de jsii AWS CDK y jsii.

AWS CDK Clave OpenPGP

ID de clave:	0x42B9CF2286CD987A
Tipo:	RSA
Tamaño:	4096/4096
Creado:	2022-07-05
Expira:	2026-07-04
ID de usuario:	Kit de desarrollo en la nube de AWS < aws-cdk@amazon.com >
Huella digital clave:	69B5 2D5B A295 1D11 FA65 413B 42B9 CF22 86CD 987A

Seleccione el icono «Copiar» para copiar la siguiente clave de 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-----
```

jsii (clave OpenPGP)

ID de clave:	0x056C4E15DAE3D8D9
Tipo:	RSA
Tamaño:	4096/4096
Creado:	2022-07-05
Expira:	2026-07-04
ID de usuario:	Equipo JSII de AWS < aws-jsii@amazon.com >
Huella digital clave:	1E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

Seleccione el icono «Copiar» para copiar la siguiente clave de 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-----
```

Claves históricas

Estas claves se pueden utilizar para validar las versiones del AWS CDK y el jsii antes del 5 de julio de 2022.

> **Important**
>
> Las nuevas claves se crean antes de que caduquen las anteriores. Como resultado, en un momento dado, puede ser válida más de una clave. Las claves se utilizan para firmar artefactos a partir del día en que se crean, así que utilice la clave emitida más recientemente, donde la validez de las claves se superponga.

AWS CDK Clave OpenPGP (7 de abril de 2022)

> **Note**
>
> Esta clave no se usó para firmar artefactos después del 5 de julio de 2022. AWS CDK

ID de clave:	0x015584281F44A3C3
Tipo:	RSA
Tamaño:	4096/4096
Creado:	2022-04-07
Expira:	2026-04-06
ID de usuario:	Kit de desarrollo en la nube de AWS < aws-cdk@amazon.com >
Huella digital clave:	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

Seleccione el icono «Copiar» para copiar la siguiente clave de 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-----
```

clave OpenPGP jsii (2022-04-07)

> **Note**
>
> Esta clave no se usó para firmar artefactos jsii después del 5 de julio de 2022.

ID de clave:	0x985F5BC974B79356
Tipo:	RSA
Tamaño:	4096/4096
Creado:	2022-04-07
Expira:	2026-04-06
ID de usuario:	Equipo JSII de AWS < aws-jsii@amazon.com >
Huella digital clave:	35A7 1785 8FA6 282D C5AC CD95 985F 5BC9 74B7 9356

Seleccione el icono «Copiar» para copiar la siguiente clave de 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 Clave OpenPGP (19/06/2018)

ID de clave:	0x0566A784E17F3870
Tipo:	RSA
Tamaño:	4096/4096
Creado:	2018-06-19
Caduca:	18 de junio de 2022
ID de usuario:	Equipo CDK de AWS < aws-cdk@amazon.com >

Huella digital clave:	E88B E3B6 F0B1 E350 9E36 4F96 0566 A784 E17F 3870

Seleccione el icono «Copiar» para copiar la siguiente clave de 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-----
```

clave OpenPGP jsii (2018-08-06)

ID de clave:	0x1C7ACE4CB2A1B93A
Tipo:	RSA

Tamaño:	4096/4096
Creado:	2018-08-06
Expira:	2022-08-05
ID de usuario:	Equipo JSII de AWS < aws-jsii@amazon.com >
Huella digital clave:	85EF 6522 4CE2 18EC 72DB 28EC 1C7A CE4C B2A1 B93A

Seleccione el icono «Copiar» para copiar la siguiente clave de 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 Historia de la Guía para desarrolladores

Consulte Versiones para obtener información sobre las AWS CDK versiones. AWS CDK Se actualiza aproximadamente una vez a la semana. Es posible que se publiquen versiones de mantenimiento entre versiones semanales para abordar problemas críticos. Cada versión incluye un AWS CDK kit de herramientas (CDK CLI), una biblioteca de AWS construcción y una referencia de API compatibles. Por lo general, las actualizaciones de esta guía no se sincronizan con las versiones. AWS CDK

> **Note**
>
> La siguiente tabla representa los hitos importantes de la documentación. Corregimos errores y mejoramos el contenido de forma continua.

Cambio	Descripción	Fecha
Añada documentación para la función CDK Migrate	Utilice el AWS CDK CLI `cdk migrate` comando para migrar AWS los recursos implementados, las AWS CloudFormation pilas implementadas y las AWS CloudFormation plantillas locales a. AWS CDK Para obtener más información, consulte Migrar a AWS CDK.	2 de febrero de 2024
Actualizaciones de las prácticas recomendadas de IAM	Se ha actualizado la guía para implementar las prácticas recomendadas de IAM. Para obtener más información, consulte prácticas recomendadas de seguridad en IAM.	23 de marzo de 2023

Documento `cdk.json`	Agregue la documentación de los valores de `cdk.json` configuración.	20 de abril de 2022
Administración de dependencias	Añada un tema sobre la gestión de las dependencias con. AWS CDK	7 de abril de 2022
Elimine las llaves dobles de los ejemplos de Java	Sustituya este antipatrón por Java 9 en todas partes. `Map.of`	9 de marzo de 2022
AWS CDK versión v2	Se publica la versión 2 de la Guía para AWS CDK desarrolladores. Historial de documentos de CDK v1.	4 de diciembre de 2021

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.