Guide du développeur

AWS Cloud Development Kit (AWS CDK) v2

Version 2

AWS Cloud Development Kit (AWS CDK) v2: Guide du développeur

Table of Contents

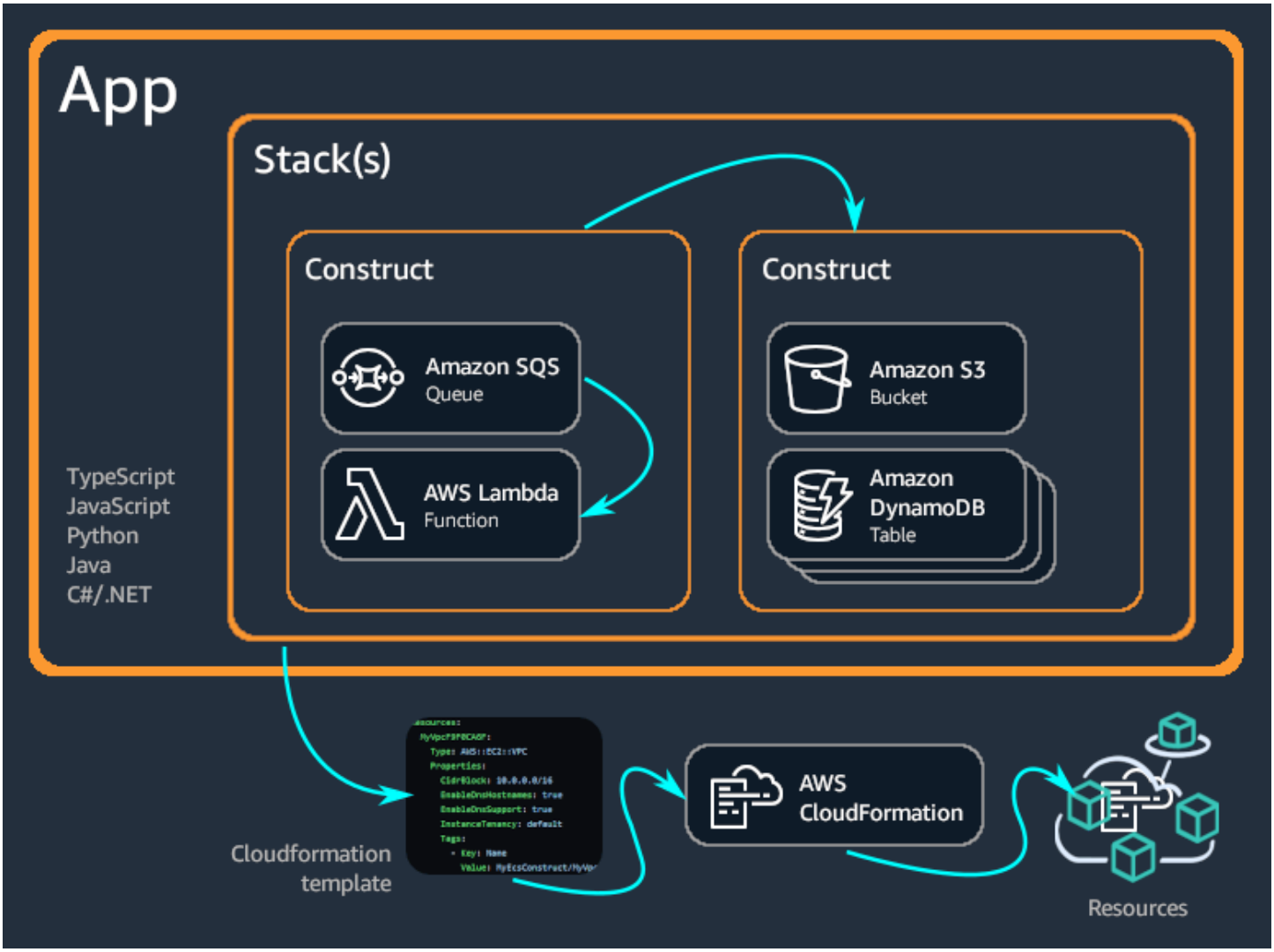
Qu'est-ce que c'est AWS CDK ?

AWS Cloud Development Kit (AWS CDK) Il s'agit d'un framework de développement de logiciels open source permettant de définir l'infrastructure cloud dans le code et de la provisionner via ce dernier. AWS CloudFormation

AWS CDK Il se compose de deux parties principales :

- AWS CDK Bibliothèque de constructions : collection de codes modulaires et réutilisables préécrits, appelés constructions, que vous pouvez utiliser, modifier et intégrer pour développer rapidement votre infrastructure. L'objectif de la bibliothèque AWS CDK Construct est de réduire la complexité requise pour définir et intégrer les AWS services ensemble lors de la création d'applications AWS.
- AWS CDK Boîte à outils — Un outil en ligne de commande pour interagir avec les applications CDK. Utilisez le AWS CDK kit d'outils pour créer, gérer et déployer vos AWS CDK projets.

Les AWS CDK supportsTypeScript, JavaScriptPython,Java,C#/.Net, etGo. Vous pouvez utiliser n'importe lequel de ces langages de programmation pris en charge pour définir des composants cloud réutilisables appelés constructions. Vous les composez ensemble en piles et en applications. Ensuite, vous déployez vos applications CDK pour AWS CloudFormation approvisionner ou mettre à jour vos ressources.



Rubriques

- Les avantages du AWS CDK
- Exemple du AWS CDK
- AWS CDK features
- Étapes suivantes
- En savoir plus

Les avantages du AWS CDK

Utilisez le AWS CDK pour développer des applications fiables, évolutives et économiques dans le cloud avec la puissance expressive considérable d'un langage de programmation. Cette approche présente de nombreux avantages, notamment :

Développez et gérez votre infrastructure sous forme de code (IaC)

Pratiquez l'infrastructure sous forme de code pour créer, déployer et maintenir l'infrastructure de manière programmatique, descriptive et déclarative. Avec IaC, vous traitez l'infrastructure de la même manière que les développeurs traitent le code. Cela se traduit par une approche évolutive et structurée de la gestion de l'infrastructure. Pour en savoir plus sur l'IaC, consultez la section Infrastructure en tant que code dans le AWS livre blanc Introduction à DevOps on.

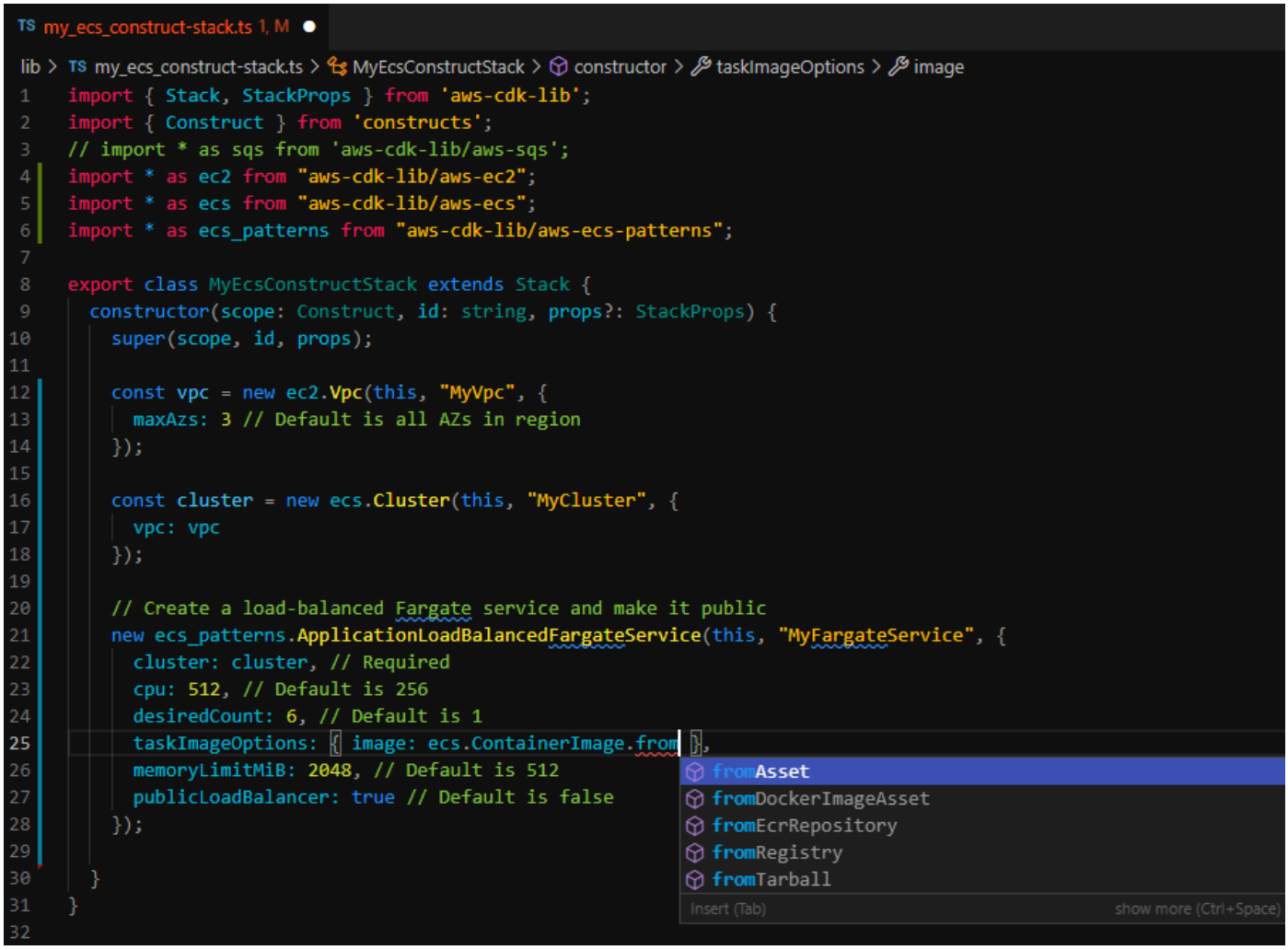
Vous pouvez ainsi regrouper votre infrastructure AWS CDK, le code de votre application et votre configuration au même endroit, afin de disposer d'un système complet et déployable dans le cloud à chaque étape. Utilisez les meilleures pratiques d'ingénierie logicielle, telles que les révisions de code, les tests unitaires et le contrôle des sources, pour renforcer la robustesse de votre infrastructure.

Définissez votre infrastructure cloud à l'aide de langages de programmation polyvalents

Avec le AWS CDK, vous pouvez utiliser l'un des langages de programmation suivants pour définir votre infrastructure cloud : TypeScriptJavaScript,Python,Java,C#/.Net, etGo. Choisissez votre langage préféré et utilisez des éléments de programmation tels que les paramètres, les conditions, les boucles, la composition et l'héritage pour définir le résultat souhaité pour votre infrastructure.

Utilisez le même langage de programmation pour définir votre infrastructure et la logique de votre application.

Bénéficiez des avantages du développement d'une infrastructure dans votre IDE (environnement de développement intégré) préféré, tels que la mise en évidence de la syntaxe et la complétion intelligente du code.



Déployez l'infrastructure via AWS CloudFormation

AWS CDK s'intègre AWS CloudFormation pour déployer et approvisionner votre infrastructure AWS. AWS CloudFormation est un système géré Service AWS qui offre une prise en charge étendue des configurations de ressources et de propriétés pour le provisionnement de services sur AWS. Vous pouvez ainsi effectuer des déploiements d'infrastructure de manière prévisible et répétée, avec annulation en cas d'erreur. AWS CloudFormation Si vous le connaissez déjà AWS CloudFormation, vous n'avez pas besoin de vous familiariser avec un nouveau service de gestion IaC pour démarrer avec le AWS CDK.

Commencez à développer rapidement votre application avec des constructions

Développez plus rapidement en utilisant et en partageant des composants réutilisables appelés constructions. Utilisez des constructions de bas niveau pour définir les AWS CloudFormation ressources individuelles et leurs propriétés. Utilisez des structures de haut niveau pour définir rapidement des composants plus importants de votre application, avec des valeurs par défaut

judicieuses et sécurisées pour vos AWS ressources, afin de définir une infrastructure plus complète avec moins de code.

Créez vos propres structures personnalisées pour vos cas d'utilisation uniques et partagez-les au sein de votre organisation ou même avec le public.

Exemple du AWS CDK

Voici un exemple d'utilisation de la bibliothèque AWS CDK Constructs pour créer un service Amazon Elastic Container Service (Amazon ECS) avec un type de lancement. AWS Fargate (Fargate) Pour plus de détails sur cet exemple, consultezthe 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

}
```

Cette classe produit un AWS CloudFormation modèle de plus de 500 lignes. Le déploiement de AWS CDK l'application produit plus de 50 ressources des types suivants.

- 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

Le AWS CDKGitHub référentiel

Pour le AWS CDK GitHub dépôt officiel, voir aws-cdk. Ici, vous pouvez soumettre des problèmes, consulter notre licence, suivre les versions, etc.

Parce qu' AWS CDK il s'agit d'un outil open source, l'équipe vous encourage à contribuer à en faire un outil encore meilleur. Pour plus de détails, voir Contribuer au AWS Cloud Development Kit (AWS CDK).

La référence de AWS CDK l'API

La bibliothèque AWS CDK Construct fournit des API pour définir votre application CDK et ajouter des constructions CDK à l'application. Pour plus d'informations, consultez la page Référence de l'API AWS CDK.

Le modèle de programmation Construct

Le modèle de programmation par construction (CPM) étend les concepts sous-jacents à d'autres domaines. AWS CDK Les autres outils utilisant le CPM incluent :

- CDK pour Terraform (CDKtf)
- CDK pour Kubernetes (CDK8s)
- Projen, pour la construction de configurations de projets

Le Construct Hub

Le Construct Hub est un registre en ligne dans lequel vous pouvez trouver, publier et partager des AWS CDK bibliothèques open source.

Étapes suivantes

Pour commencer à utiliser le AWS CDK, voirCommencer à utiliser le AWS CDK.

En savoir plus

Pour en savoir plus sur le AWS CDK, consultez les rubriques suivantes :

- AWS CDK concepts— Concepts et termes importants pour AWS CDK.
- AWS CDK Atelier — Atelier pratique pour apprendre et utiliser le AWS CDK.
- AWS CDK Modèles — Collection open source de modèles d'architecture AWS sans serveur, conçue pour eux AWS CDK par AWS des experts.
- AWS CDK exemples de code — GitHub référentiel d'exemples de AWS CDK projets.
- cdk.dev — Hub géré par la communauté pour le AWS CDK, y compris un espace de travail communautaire. Slack
- Awesome CDK — GitHub référentiel contenant une liste organisée de projets AWS CDK open source, de guides, de blogs et d'autres ressources.
- AWS Constructions de solutions — Modèles d'infrastructure de configuration sous forme de code (IaC) approuvés qui peuvent facilement être assemblés dans des applications prêtes à être mises en production.
- AWS Blog sur les outils de développement — Les articles de blog sont filtrés pour le AWS CDK.
- AWS CDK activé Stack Overflow — Questions marquées avec aws-cdk activé. Stack Overflow
- AWS CDK tutoriel pour AWS Cloud9 — Tutoriel sur l'utilisation AWS CDK de l'environnement de AWS Cloud9 développement.

Pour en savoir plus sur les sujets liés à la AWS CDK, consultez les rubriques suivantes :

- AWS CloudFormation concepts — Comme AWS CDK il est conçu pour fonctionner avec AWS CloudFormation, nous vous recommandons d'apprendre et de comprendre les AWS CloudFormation concepts clés.

- AWS Glossaire — Définitions des principaux termes utilisés dans l'ensemble. AWS

Pour en savoir plus sur les outils associés AWS CDK qui peuvent être utilisés pour simplifier le développement et le déploiement d'applications sans serveur, consultez les pages suivantes :

- AWS Serverless Application Model— Un outil de développement open source qui simplifie et améliore l'expérience de création et d'exécution d'applications sans serveur sur. AWS
- AWSChalice— Un framework pour écrire des applications sans serveur. Python

AWS CDK concepts

Découvrez les concepts de base qui sous-tendent le AWS Cloud Development Kit (AWS CDK).

AWS CDK et IaC

AWS CDK Il s'agit d'un framework open source que vous pouvez utiliser pour gérer votre AWS infrastructure à l'aide de code. Cette approche est connue sous le nom d'infrastructure sous le nom de code (IaC). En gérant et en provisionnant votre infrastructure sous forme de code, vous traitez votre infrastructure de la même manière que les développeurs traitent le code. Cela offre de nombreux avantages, tels que le contrôle des versions et l'évolutivité. Pour en savoir plus sur IaC, consultez Qu'est-ce que l'infrastructure en tant que code ?

AWS CDK et AWS CloudFormation

Le AWS CDK est étroitement intégré à AWS CloudFormation. AWS CloudFormation est un service entièrement géré que vous pouvez utiliser pour gérer et approvisionner votre infrastructure AWS. Avec AWS CloudFormation, vous définissez votre infrastructure dans des modèles et vous les déployez sur AWS CloudFormation. Le AWS CloudFormation service provisionne ensuite votre infrastructure conformément à la configuration définie sur vos modèles.

AWS CloudFormation les modèles sont déclaratifs, ce qui signifie qu'ils déclarent l'état ou le résultat souhaité de votre infrastructure. À l'aide de JSON ou de YAML, vous déclarez votre AWS infrastructure en définissant AWS des ressources et des propriétés. Les ressources représentent les nombreux services présents AWS et les propriétés représentent la configuration que vous souhaitez pour ces services. Lorsque vous déployez votre modèle sur AWS CloudFormation, vos ressources et leurs propriétés configurées sont provisionnées comme décrit sur votre modèle.

Avec le AWS CDK, vous pouvez gérer votre infrastructure de manière impérative, en utilisant des langages de programmation polyvalents. Au lieu de simplement définir un état souhaité de manière déclarative, vous pouvez définir la logique ou la séquence nécessaire pour atteindre l'état souhaité. Par exemple, vous pouvez utiliser `if` des instructions ou des boucles conditionnelles qui déterminent comment atteindre l'état final souhaité pour votre infrastructure.

L'infrastructure créée avec le AWS CDK est finalement traduite ou synthétisée sous forme de AWS CloudFormation modèles et déployée à l'aide du AWS CloudFormation service. Ainsi, bien qu'il

AWS CDK propose une approche différente pour créer votre infrastructure, vous bénéficiez toujours des avantages AWS CloudFormation, tels qu'une assistance étendue à la configuration AWS des ressources et des processus de déploiement robustes.

Pour en savoir plus AWS CloudFormation, voir Qu'est-ce que c'est AWS CloudFormation ? dans le guide de AWS CloudFormation l'utilisateur.

AWS CDK et abstractions

Avec AWS CloudFormation, vous devez définir dans les moindres détails la façon dont vos ressources sont configurées. Cela offre l'avantage d'avoir un contrôle total sur votre infrastructure. Toutefois, cela nécessite que vous appreniez, compreniez et créiez des modèles robustes contenant les détails de configuration des ressources et les relations entre les ressources, telles que les autorisations et les interactions basées sur les événements.

Avec le AWS CDK, vous pouvez avoir le même contrôle sur la configuration de vos ressources. Cependant, il propose AWS CDK également de puissantes abstractions, qui peuvent accélérer et simplifier le processus de développement de l'infrastructure. Par exemple, il AWS CDK inclut des constructions qui fournissent des configurations par défaut raisonnables et des méthodes d'assistance qui génèrent du code standard pour vous. AWS CDK Il propose également des outils, tels que l'interface de ligne de AWS CDK commande (AWS CDK CLI), qui exécutent des actions de gestion de l'infrastructure pour vous.

En savoir plus sur les AWS CDK concepts de base

Interaction avec le AWS CDK

Lorsque vous utilisez le AWS CDK, vous interagissez principalement avec la bibliothèque AWS Construct et le AWS CDK CLI.

Développer avec le AWS CDK

Il AWS CDK peut être écrit dans n'importe quel langage de programmation pris en charge. Vous commencez par un projet CDK, qui contient une structure de dossiers et de fichiers, y compris des actifs. Dans le projet, vous créez une application CDK. Dans l'application, vous définissez une pile, qui représente directement une CloudFormation pile. Au sein de la pile, vous définissez vos AWS ressources et vos propriétés à l'aide de constructions.

Déploiement à l'aide du AWS CDK

Vous déployez des applications CDK dans un AWS environnement. Avant le déploiement, vous devez effectuer un démarrage unique pour préparer votre environnement.

En savoir plus

Pour en savoir plus sur les concepts de AWS CDK base, consultez les rubriques de cette section.

Langages de programmation pris en charge

AWS Cloud Development Kit (AWS CDK) Il dispose d'un support de premier ordre pour les langages de programmation généraux suivants :

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

JVMD'autres .NET CLR langues peuvent également être utilisées en théorie, mais nous n'offrons pas de support officiel pour le moment.

> **Note**
>
> Ce guide ne contient actuellement pas d'instructions ni d'exemples de code pour l'Goexception dethe section called “Dans Go”.

AWS CDK Il est développé dans une seule langue,TypeScript. Pour prendre en charge les autres langues, il AWS CDK utilise un outil appelé JSIIpour générer des liaisons linguistiques.

Nous essayons de proposer les conventions habituelles de chaque langue afin de rendre le développement AWS CDK aussi naturel et intuitif que possible. Par exemple, nous distribuons les modules AWS Construct Library en utilisant le référentiel standard de votre langue préférée, et

vous les installez à l'aide du gestionnaire de packages standard du langage. Les méthodes et les propriétés sont également nommées selon les modèles de dénomination recommandés par votre langue.

Voici quelques exemples de code :

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**
>
> Ces extraits de code sont uniquement destinés à des fins d'illustration. Ils sont incomplets et ne fonctionneront pas tels quels.

La bibliothèque AWS Construct est distribuée à l'aide des outils de gestion de paquets standard de chaque langueNPM, notamment PyPiMaven,, etNuGet. Nous fournissons également une version de la référence AWS CDK d'API pour chaque langue.

Pour vous aider à utiliser le AWS CDK dans la langue de votre choix, ce guide inclut les rubriques suivantes relatives aux langues prises en charge :

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

TypeScripta été le premier langage pris en charge par le AWS CDK, et une grande partie de l' AWS CDK exemple de code est écrite enTypeScript. Ce guide inclut une rubrique expliquant spécifiquement comment adapter le TypeScript AWS CDK code pour l'utiliser avec les autres langages pris en charge. Pour plus d'informations, voir Comparaison AWS CDKTypeScript avec d'autres langues.

AWS CDK projets

Un AWS Cloud Development Kit (AWS CDK) projet représente les fichiers et les dossiers qui contiennent votre code CDK. Le contenu peut varier en fonction de votre langage de programmation.

Vous pouvez créer votre AWS CDK projet manuellement ou à l'aide de la AWS CDK commande Command Line Interface (AWS CDK CLI)`cdk init`. Dans cette rubrique, nous aborderons la structure du projet et les conventions de dénomination des fichiers et dossiers créés par la CLI AWS CDK. Vous pouvez personnaliser et organiser vos projets CDK en fonction de vos besoins.

> **Note**
>
> La structure de projet créée par le AWS CDK CLI peut varier d'une version à l'autre au fil du temps.

Rubriques

Fichiers et dossiers universels

.git

Si vous l'avez `git` installé, un Git référentiel est AWS CDK CLI automatiquement initialisé pour votre projet. Le `.git` répertoire contient des informations sur le référentiel.

.gitignore

Fichier texte utilisé par Git pour spécifier les fichiers et les dossiers à ignorer.

README.md

Fichier texte qui vous fournit des conseils de base et des informations importantes pour la gestion de votre AWS CDK projet. Modifiez ce fichier si nécessaire pour documenter les informations importantes concernant votre projet CDK.

cdk.json

Fichier de configuration pour AWS CDK. Ce fichier fournit des instructions AWS CDK CLI sur la façon d'exécuter votre application.

Fichiers et dossiers spécifiques à la langue

Les fichiers et dossiers suivants sont propres à chaque langage de programmation pris en charge.

TypeScript

Voici un exemple de projet créé dans le `my-cdk-ts-project` répertoire à l'aide de la `cdk init --language typescript` commande :

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

.npm ignorer

Fichier qui indique les fichiers et les dossiers à ignorer lors de la publication d'un package npm dans le registre. Ce fichier est similaire aux npm packages `.gitignore`, mais il est spécifique à ceux-ci.

bin/.ts my-cdk-ts-project

Le fichier d'application définit votre application CDK. Les projets CDK peuvent contenir un ou plusieurs fichiers d'application. Les fichiers de candidature sont stockés dans le `bin` dossier.

Voici un exemple de fichier d'application de base qui définit une application 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

Fichier de configuration pourJest. Jestest un framework de JavaScript test populaire.

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

Le fichier de pile définit votre pile de CDK. Au sein de votre pile, vous définissez les AWS ressources et les propriétés à l'aide de constructions.

Voici un exemple de fichier de pile de base qui définit une pile 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

Dossier commun aux Node.js projets contenant des dépendances pour votre projet.

package-lock.json

Fichier de métadonnées qui fonctionne avec le `package.json` fichier pour gérer les versions des dépendances.

package.json

Fichier de métadonnées couramment utilisé dans les Node.js projets. Ce fichier contient des informations sur votre projet CDK, telles que le nom du projet, les définitions de script, les dépendances et d'autres informations au niveau du projet d'importation.

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

Un dossier de test est créé pour organiser les tests de votre projet CDK. Un exemple de fichier de test est également créé.

Vous pouvez y écrire des tests TypeScript et les utiliser Jest pour compiler votre TypeScript code avant d'exécuter les tests.

tsconfig.json

Fichier de configuration utilisé dans les TypeScript projets qui spécifie les options du compilateur et les paramètres du projet.

JavaScript

Voici un exemple de projet créé dans le `my-cdk-js-project` répertoire à l'aide de la `cdk init --language javascript` commande :

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

.npm ignorer

Fichier qui indique les fichiers et les dossiers à ignorer lors de la publication d'un package npm dans le registre. Ce fichier est similaire aux npm packages `.gitignore`, mais il est spécifique à ceux-ci.

bin/.js my-cdk-js-project

Le fichier d'application définit votre application CDK. Les projets CDK peuvent contenir un ou plusieurs fichiers d'application. Les fichiers de candidature sont stockés dans le `bin` dossier.

Voici un exemple de fichier d'application de base qui définit une application 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

Fichier de configuration pourJest. Jestest un framework de JavaScript test populaire.

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

Le fichier de pile définit votre pile de CDK. Au sein de votre pile, vous définissez les AWS ressources et les propriétés à l'aide de constructions.

Voici un exemple de fichier de pile de base qui définit une pile 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

Dossier commun aux Node.js projets contenant des dépendances pour votre projet.

package-lock.json

Fichier de métadonnées qui fonctionne avec le `package.json` fichier pour gérer les versions des dépendances.

package.json

Fichier de métadonnées couramment utilisé dans les Node.js projets. Ce fichier contient des informations sur votre projet CDK, telles que le nom du projet, les définitions de script, les dépendances et d'autres informations au niveau du projet d'importation.

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

Un dossier de test est créé pour organiser les tests de votre projet CDK. Un exemple de fichier de test est également créé.

Vous pouvez y écrire des tests JavaScript et les utiliser Jest pour compiler votre JavaScript code avant d'exécuter les tests.

Python

Voici un exemple de projet créé dans le `my-cdk-py-project` répertoire à l'aide de la `cdk init --language python` commande :

```
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

Le CDK crée CLI automatiquement un environnement virtuel pour votre projet. Le `.venv` répertoire fait référence à cet environnement virtuel.

app.py

Le fichier d'application définit votre application CDK. Les projets CDK peuvent contenir un ou plusieurs fichiers d'application.

Voici un exemple de fichier d'application de base qui définit une application 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

Répertoire contenant vos fichiers de pile. Le CDK CLI crée ce qui suit ici :

- `__init__.py` — Un fichier de définition de Python package vide.
- `my_cdk_py_project`— Fichier qui définit votre pile de CDK. Vous définissez ensuite les AWS ressources et les propriétés au sein de la pile à l'aide de constructions.

Voici un exemple de fichier de pile :

```
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

Fichier similaire à`requirements.txt`, mais utilisé pour gérer les dépendances spécifiquement à des fins de développement plutôt que de production.

requirements.txt

Fichier commun utilisé dans les Python projets pour spécifier et gérer les dépendances des projets.

source.bat

Le fichier batch Windows correspondant est utilisé pour configurer l'environnement Python virtuel.

tests

Répertoire contenant les tests pour votre projet CDK.

Voici un exemple de test unitaire :

```
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

Voici un exemple de projet créé dans le `my-cdk-java-project` répertoire à l'aide de la `cdk init --language java` commande :

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

```
  ### main
  ### test
```

pom.xml

Fichier contenant les informations de configuration et les métadonnées relatives à votre projet CDK. Ce fichier fait partie deMaven.

src/main

Répertoire contenant votre application et vos fichiers de pile.

Voici un exemple de fichier de candidature :

```
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();
 }
}
```

Voici un exemple de fichier de pile :

```
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

Répertoire contenant vos fichiers de test. Voici un exemple :

```
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#

Voici un exemple de projet créé dans le `my-cdk-csharp-project` répertoire à l'aide de la `cdk init --language csharp` commande :

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

src/ MyCdkCsharpProject

Répertoire contenant votre application et vos fichiers de pile.

Voici un exemple de fichier de candidature :

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

Voici un exemple de fichier de pile :

```
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
  }
 }
}
```

Ce répertoire contient également les éléments suivants :

- `GlobalSuppressions.cs`— Fichier utilisé pour supprimer des avertissements ou des erreurs spécifiques du compilateur dans votre projet.
- `.csproj`— Fichier XML utilisé pour définir les paramètres du projet, les dépendances et les configurations de construction.

src/.sln MyCdkCsharpProject

Microsoft Visual Studio Solution Fileutilisé pour organiser et gérer des projets connexes.

Go

Voici un exemple de projet créé dans le `my-cdk-go-project` répertoire à l'aide de la `cdk init --language go` commande :

```
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

Fichier contenant des informations sur le module et utilisé pour gérer les dépendances et le versionnement de votre Go projet.

my-cdk-go-project.go

Fichier qui définit votre application CDK et ses piles.

Voici un exemple :

```
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

Fichier qui définit un exemple de test.

Voici un exemple :

```
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 applications

L' AWS Cloud Development Kit (AWS CDK) application ou l'application est une collection d'une ou de plusieurs piles de CDK. Les piles sont un ensemble d'une ou plusieurs constructions qui définissent les AWS ressources et les propriétés. Par conséquent, le regroupement global de vos piles et de vos constructions est connu sous le nom d'application CDK.

Rubriques

Définition des applications

Vous créez une application en définissant une instance d'application dans le fichier d'application de votre projet. Pour ce faire, vous devez importer et utiliser la Appconstruction depuis la AWS bibliothèque de constructions. La App construction ne nécessite aucun argument d'initialisation. C'est la seule construction qui peut être utilisée comme racine.

Les `Stack` classes `App` et de la bibliothèque AWS de constructions sont des constructions uniques. Comparées à d'autres constructions, elles ne configurent pas AWS les ressources par elles-mêmes. Ils sont plutôt utilisés pour fournir un contexte à vos autres constructions. Toutes les constructions qui représentent AWS des ressources doivent être définies, directement ou indirectement, dans le cadre d'une `Stack` construction. `Stack`les constructions sont définies dans le cadre d'une App construction.

Les applications sont ensuite synthétisées pour créer des AWS CloudFormation modèles pour vos piles. Voici un exemple :

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)
```

Les piles d'une même application peuvent facilement faire référence aux ressources et aux propriétés des autres. Il AWS CDK déduit les dépendances entre les piles afin qu'elles puissent être déployées dans le bon ordre. Vous pouvez déployer tout ou partie des piles d'une application à l'aide d'une seule `cdk deploy` commande.

L'arbre de construction

Les constructions sont définies à l'intérieur d'autres constructions en utilisant l'`scope`argument transmis à chaque construction, avec la `App` classe comme racine. De cette façon, une AWS CDK application définit une hiérarchie de constructions connue sous le nom d'arbre de construction.

La racine de cet arbre est votre application, qui est une instance de la `App` classe. Dans l'application, vous instanciez une ou plusieurs piles. Dans les piles, vous instanciez des constructions, qui

peuvent elles-mêmes instancier des ressources ou d'autres constructions, et ainsi de suite dans l'arborescence.

Les constructions sont toujours définies explicitement dans le cadre d'une autre construction, ce qui crée des relations entre les constructions. Presque toujours, vous devez passer `this` (en Python`self`) comme portée, indiquant que la nouvelle construction est un enfant de la construction actuelle. Le modèle prévu est que vous dériviez votre construction `Construct`, puis que vous instanciez les constructions qu'elle utilise dans son constructeur.

Le fait de transmettre explicitement la portée permet à chaque construction de s'ajouter à l'arbre, ce comportement étant entièrement contenu dans la classe `Construct` de base. Il fonctionne de la même manière dans toutes les langues prises en charge par le AWS CDK et ne nécessite aucune personnalisation supplémentaire.

> **Important**
>
> Techniquement, il est possible de transmettre une certaine portée autrement que `this` lors de l'instanciation d'une construction. Vous pouvez ajouter des constructions n'importe où dans l'arborescence, ou même dans une autre pile de la même application. Par exemple, vous pouvez écrire une fonction de style mixin qui ajoute des constructions à une portée transmise en tant qu'argument. La difficulté pratique ici est que vous ne pouvez pas facilement vous assurer que les identifiants que vous choisissez pour vos constructions sont uniques dans le cadre de la portée de quelqu'un d'autre. Cette pratique rend également votre code plus difficile à comprendre, à maintenir et à réutiliser. Il est presque toujours préférable de trouver un moyen d'exprimer son intention sans abuser de l'`scope`argument.

AWS CDK Utilise les identifiants de toutes les constructions situées sur le chemin allant de la racine de l'arbre à chaque construction enfant pour générer les identifiants uniques requis par AWS CloudFormation. Cette approche signifie que les identifiants de construction doivent uniquement être uniques dans leur champ d'application, plutôt que dans l'ensemble de la pile, comme dans le mode natif AWS CloudFormation. Toutefois, si vous déplacez une construction vers une autre étendue, son identifiant unique de pile généré change et elle ne AWS CloudFormation sera pas considérée comme la même ressource.

L'arbre de construction est distinct des constructions que vous définissez dans votre AWS CDK code. Cependant, il est accessible via `node` l'attribut de n'importe quelle construction, qui est une référence

au nœud qui représente cette construction dans l'arbre. Chaque nœud est une Nodeinstance dont les attributs donnent accès à la racine de l'arbre ainsi qu'aux étendues parents et aux enfants du nœud.

1. `node.children`— Les enfants directs de la construction.
2. `node.id`— L'identifiant de la construction comprise dans son champ d'application.
3. `node.path`— Le chemin complet de la construction, y compris les identifiants de tous ses parents.
4. `node.root`— La racine de l'arbre de construction (l'application).
5. `node.scope`— La portée (parent) de la construction, ou indéfinie si le nœud est la racine.
6. `node.scopes`— Tous les parents de la construction, jusqu'à la racine.
7. `node.uniqueId`— L'identifiant alphanumérique unique de cette construction dans l'arbre (par défaut, généré à partir `node.path` d'un hachage).

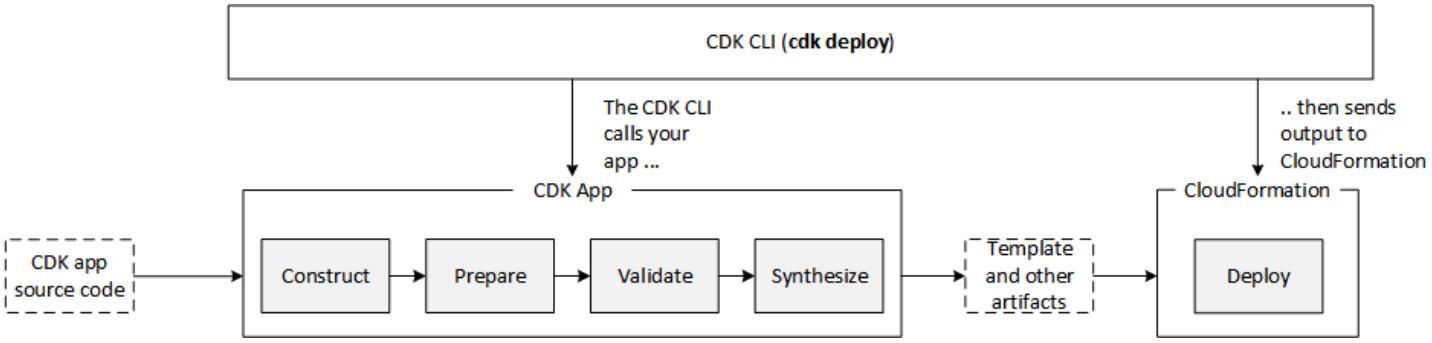
L'arbre de construction définit un ordre implicite dans lequel les constructions sont synthétisées avec les ressources du modèle final AWS CloudFormation . Lorsqu'une ressource doit être créée avant une autre, AWS CloudFormation ou lorsque la bibliothèque de AWS construction en déduit généralement la dépendance. Ils s'assurent ensuite que les ressources sont créées dans le bon ordre.

Vous pouvez également ajouter une dépendance explicite entre deux nœuds en utilisantnode.addDependency(). Pour plus d'informations, consultez la section Dépendances dans la référence de AWS CDK l'API.

AWS CDK Il fournit un moyen simple de visiter chaque nœud de l'arbre de construction et d'effectuer une opération sur chacun d'eux. Pour plus d’informations, consultez the section called “Aspects”.

Le cycle de vie des applications

Lorsque vous déployez votre application CDK, les phases suivantes ont lieu. C'est ce que l'on appelle le cycle de vie des applications :



Une AWS CDK application passe par les phases suivantes de son cycle de vie.

- Construction (ou initialisation) : votre code instancie toutes les constructions définies, puis les lie entre elles. À ce stade, toutes les constructions (app, stacks et leurs constructions dérivées) sont instanciées et la chaîne de constructeurs est exécutée. La majeure partie du code de votre application est exécutée à cette étape.
- Préparation — Toutes les constructions qui ont implémenté la `prepare` méthode participent à une dernière série de modifications afin de définir leur état final. La phase de préparation se déroule automatiquement. En tant qu'utilisateur, vous ne recevez aucun commentaire concernant cette phase. Il est rare d'avoir besoin d'utiliser le crochet « Prepare » et n'est généralement pas recommandé. Soyez très prudent lorsque vous mutez l'arbre de construction au cours de cette phase, car l'ordre des opérations peut avoir un impact sur le comportement.
- Validation : toutes les constructions qui ont implémenté la `validate` méthode peuvent se valider elles-mêmes pour s'assurer qu'elles sont dans un état permettant un déploiement correct. Vous serez informé de tout échec de validation survenant au cours de cette phase. En général, nous recommandons d'effectuer la validation dès que possible (généralement dès que vous recevez des informations) et de lancer des exceptions le plus tôt possible. La validation précoce améliore la fiabilité, car les traces de pile seront plus précises et garantiront que votre code peut continuer à s'exécuter en toute sécurité.
- Synthèse — Il s'agit de la dernière étape de l'exécution de votre AWS CDK application. Il est déclenché par un appel à`app.synth()`, traverse l'arbre de construction et invoque la `synthesize` méthode sur toutes les constructions. Les constructions implémentées `synthesize` peuvent participer à la synthèse et émettre des artefacts de déploiement vers l'assemblage cloud qui en résulte. Ces artefacts incluent des AWS CloudFormation modèles, des ensembles AWS Lambda d'applications, des fichiers et des Docker images, ainsi que d'autres artefacts de déploiement. the section called “Assemblages cloud”décrit le résultat de cette phase. Dans la plupart des cas, il n'est pas nécessaire d'implémenter la `synthesize` méthode.

- Déploiement — Au cours de cette phase, l' AWS CDK CLIassemblage cloud des artefacts de déploiement produit par la phase de synthèse est déployé dans un AWS environnement. Il télécharge les actifs sur Amazon S3 et Amazon ECR, ou partout où ils doivent aller. Il lance ensuite un AWS CloudFormation déploiement pour déployer l'application et créer les ressources.

Au début de la phase de AWS CloudFormation déploiement, votre AWS CDK application est déjà terminée et s'est terminée. Cela a les conséquences suivantes :

- L' AWS CDK application ne peut pas répondre aux événements qui se produisent pendant le déploiement, tels que la création d'une ressource ou la fin complète du déploiement. Pour exécuter du code pendant la phase de déploiement, vous devez l'injecter dans le AWS CloudFormation modèle en tant que ressource personnalisée. Pour plus d'informations sur l'ajout d'une ressource personnalisée à votre application, consultez le AWS CloudFormation module ou l'exemple de ressource personnalisée.
- L' AWS CDK application devra peut-être fonctionner avec des valeurs qui ne peuvent pas être connues au moment de son exécution. Par exemple, si l' AWS CDK application définit un compartiment Amazon S3 avec un nom généré automatiquement et que vous récupérez l'attribut `bucket.bucketName` (Python :`bucket_name`), cette valeur n'est pas le nom du compartiment déployé. Au lieu de cela, vous obtenez une `Token` valeur. Pour déterminer si une valeur particulière est disponible, appelez `cdk.isUnresolved(value)` (Python :`is_unresolved`). Consultez the section called “Jetons” pour plus de détails.

Assemblages cloud

L'appel à `app.synth()` est ce qui indique AWS CDK au de synthétiser un assemblage cloud à partir d'une application. En général, vous n'interagissez pas directement avec les assemblages cloud. Il s'agit de fichiers contenant tout ce dont vous avez besoin pour déployer votre application dans un environnement cloud. Par exemple, il inclut un AWS CloudFormation modèle pour chaque pile de votre application. Il inclut également une copie de tous les fichiers ou images Docker auxquels vous faites référence dans votre application.

Consultez la spécification des assemblages cloud pour plus de détails sur le formatage des assemblages cloud.

Pour interagir avec l'assemblage cloud créé par votre AWS CDK application, vous utilisez généralement le AWS CDK CLI. Cependant, tout outil capable de lire le format d'assemblage du cloud peut être utilisé pour déployer votre application.

Exécution de votre application

Le CDK CLI doit savoir comment exécuter votre AWS CDK application. Si vous avez créé le projet à partir d'un modèle à l'aide de la `cdk init` commande, le `cdk.json` fichier de votre application inclut une `app` clé. Cette clé indique la commande nécessaire pour la langue dans laquelle l'application est écrite. Si votre langage nécessite une compilation, la ligne de commande exécute cette étape avant d'exécuter l'application. Vous ne pouvez donc pas oublier de le faire.

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 vous n'avez pas créé votre projet à l'aide du CDKCLI, ou si vous souhaitez remplacer la ligne de commande indiquée`cdk.json`, vous pouvez utiliser l'--appoption lors de l'émission de la `cdk` commande.

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

La partie *exécutable* de la commande indique la commande qui doit être exécutée pour exécuter votre application CDK. Utilisez les guillemets comme indiqué, car ces commandes contiennent des espaces. La *commande cdk* est une sous-commande similaire à synth ou deploy qui indique au CDK CLI ce que vous voulez faire avec votre application. Suivez cette procédure avec toutes les options supplémentaires nécessaires pour cette sous-commande.

Ils AWS CDK CLI peuvent également interagir directement avec un assemblage cloud déjà synthétisé. Pour ce faire, transmettez le répertoire dans lequel l'assemblage cloud est stocké--app. L'exemple suivant répertorie les piles définies dans l'assemblage cloud stocké sous`./my-cloud-assembly`.

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

Piles

Une AWS Cloud Development Kit (AWS CDK) pile est un ensemble d'une ou de plusieurs constructions qui définissent les AWS ressources. Chaque pile CDK représente une AWS CloudFormation pile dans votre application CDK. Lors du déploiement, les constructions d'une pile sont fournies en tant qu'unité unique, appelée pile. AWS CloudFormation Pour en savoir plus sur les AWS CloudFormation piles, consultez la section Utilisation des piles dans le Guide de l'AWS CloudFormation utilisateur.

Étant donné que les piles CDK sont mises en œuvre par le biais de AWS CloudFormation piles, des AWS CloudFormation quotas et des limitations s'appliquent. Pour en savoir plus, consultez la section AWS CloudFormation Quotas.

Rubriques

- Définition des piles

- Utilisation des piles

Définition des piles

Les piles sont définies dans le contexte d'une application. Vous définissez une pile à l'aide de la `Stack` classe de la bibliothèque AWS Construct. Les piles peuvent être définies de l'une des manières suivantes :

- Directement dans le cadre de l'application.
- Indirectement par n'importe quelle construction de l'arbre.

L'exemple suivant définit une application CDK contenant deux piles :

TypeScript

```
const app = new App();

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

app.synth();
```

JavaScript

```
const app = new App();

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

app.synth();
```

Python

```
app = App()

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

app.synth()
```

Java

```
App app = new App();

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

app.synth();
```

C#

```
var app = new App();

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

app.Synth();
```

L'exemple suivant est un modèle courant pour définir une pile dans un fichier distinct. Ici, nous étendons ou héritons de la `Stack` classe et définissons un constructeur qui accepte `scopeid`, et. `props` Ensuite, nous invoquons le constructeur de la `Stack` classe de base en utilisant `super` avec les éléments reçus `scopeid`, et`props`.

TypeScript

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

    //...
  }
}
```

JavaScript

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

    //...
```

```
  }
}
```

Python

```
class HelloCdkStack(Stack):

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

        # ...
```

Java

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

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

        // ...
    }
}
```

C#

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

Go

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

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

  return stack
}
```

L'exemple suivant déclare une classe de pile nommée `MyFirstStack` qui inclut un seul compartiment 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
}
```

Cependant, ce code n'a déclaré qu'une pile. Pour que la pile soit réellement synthétisée dans un AWS CloudFormation modèle et déployée, elle doit être instanciée. Et, comme toutes les constructions CDK, elle doit être instanciée dans un certain contexte. `App`C'est ce contexte.

Si vous utilisez le modèle de AWS CDK développement standard, vos piles sont instanciées dans le même fichier que celui dans lequel vous instanciez l'objet. `App`

TypeScript

Le fichier nommé d'après votre projet (par exemple,`hello-cdk.ts`) dans `bin` le dossier de votre projet.

JavaScript

Le fichier nommé d'après votre projet (par exemple,`hello-cdk.js`) dans `bin` le dossier de votre projet.

Python

Le fichier situé `app.py` dans le répertoire principal de votre projet.

Java

Le fichier nommé*`ProjectName`*`App.java`, par exemple`HelloCdkApp.java`, est profondément imbriqué sous le `src/main` répertoire.

C#

Le fichier nommé `Program.cs` ci-dessous`src\`*`ProjectName`*, par exemple`src\HelloCdk\Program.cs`.

L'API Stack

L'objet Stack fournit une API riche, notamment les éléments suivants :

- `Stack.of(construct)`— Méthode statique qui renvoie la pile dans laquelle une construction est définie. Ceci est utile si vous devez interagir avec une pile à partir d'une construction réutilisable. L'appel échoue si aucune pile n'est trouvée dans la portée.
- `stack.stackName`(Python :`stack_name`) — Renvoie le nom physique de la pile. Comme mentionné précédemment, toutes les AWS CDK piles ont un nom physique qu'elles AWS CDK peuvent résoudre lors de la synthèse.

- `stack.region`et `stack.account` — Renvoie la AWS région et le compte, respectivement, dans lesquels cette pile sera déployée. Ces propriétés renvoient l'une des valeurs suivantes :
  - Le compte ou la région explicitement spécifié lors de la définition de la pile
  - Un jeton codé par chaîne qui résout les AWS CloudFormation pseudo-paramètres du compte et de la région pour indiquer que cette pile est indépendante de l'environnement

  Pour plus d'informations sur la manière dont les environnements sont déterminés pour les piles, consultezthe section called “Environnements”.
- `stack.addDependency(stack)`(Python : `stack.add_dependency(stack)` — Peut être utilisé pour définir explicitement l'ordre de dépendance entre deux piles. Cet ordre est respecté par la cdk deploy commande lors du déploiement de plusieurs piles à la fois.
- `stack.tags`— Renvoie un TagManagerque vous pouvez utiliser pour ajouter ou supprimer des balises au niveau de la pile. Ce gestionnaire de balises balise toutes les ressources de la pile et balise également la pile elle-même lorsqu'elle est créée AWS CloudFormation.
- `stack.partition`, `stack.urlSuffix` (Python :`url_suffix`), `stack.stackId` (Python :`stack_id`) et `stack.notificationArn` (Python :`notification_arn`) — Renvoie des jetons qui se résolvent en AWS CloudFormation pseudo-paramètres respectifs, tels que`{ "Ref": "AWS::Partition" }`. Ces jetons sont associés à l'objet de pile spécifique afin que le AWS CDK framework puisse identifier les références entre piles.
- `stack.availabilityZones`(Python :`availability_zones`) — Renvoie l'ensemble des zones de disponibilité disponibles dans l'environnement dans lequel cette pile est déployée. Pour les piles indépendantes de l'environnement, cela renvoie toujours une baie avec deux zones de disponibilité. Pour les piles spécifiques à un environnement, le AWS CDK interroge l'environnement et renvoie l'ensemble exact de zones de disponibilité disponibles dans la région que vous avez spécifiée.
- `stack.parseArn(arn)`and `stack.formatArn(comps)` (Python :`parse_arn`,`format_arn`) — Peut être utilisé pour travailler avec Amazon Resource Names (ARN).
- `stack.toJsonString(obj)`(Python :`to_json_string`) — Peut être utilisé pour formater un objet arbitraire sous forme de chaîne JSON pouvant être intégrée dans un AWS CloudFormation modèle. L'objet peut inclure des jetons, des attributs et des références, qui ne sont résolus que lors du déploiement.
- `stack.templateOptions`(Python :`template_options`) — À utiliser pour spécifier les options du AWS CloudFormation modèle, telles que Transform, Description et Metadata, pour votre pile.

Utilisation des piles

Pour répertorier toutes les piles d'une application CDK, utilisez la cdk ls commande. L'exemple précédent produirait ce qui suit :

```
stack1
stack2
```

Les piles sont déployées dans le cadre d'une AWS CloudFormation pile dans un AWS environnement. L'environnement couvre un domaine spécifique Compte AWS et Région AWS.

Lorsque vous exécutez la cdk synth commande pour une application comportant plusieurs piles, l'assemblage cloud inclut un modèle distinct pour chaque instance de pile. Même si les deux piles sont des instances de la même classe, elles sont émises AWS CDK sous forme de deux modèles individuels.

Vous pouvez synthétiser chaque modèle en spécifiant le nom de la pile dans la cdk synth commande. L'exemple suivant synthétise le modèle pour stack1.

```
$ cdk synth stack1
```

Cette approche est conceptuellement différente de la façon dont les AWS CloudFormation modèles sont normalement utilisés, où un modèle peut être déployé plusieurs fois et paramétré par le biais de paramètres.AWS CloudFormation Bien que AWS CloudFormation les paramètres puissent être définis dans le AWS CDK, ils sont généralement déconseillés car AWS CloudFormation les paramètres ne sont résolus que lors du déploiement. Cela signifie que vous ne pouvez pas déterminer leur valeur dans votre code.

Par exemple, pour inclure de manière conditionnelle une ressource dans votre application en fonction d'une valeur de paramètre, vous devez définir une AWS CloudFormation condition et l'associer à la ressource. AWS CDK Il adopte une approche dans laquelle les modèles concrets sont résolus au moment de la synthèse. Par conséquent, vous pouvez utiliser une instruction if pour vérifier la valeur afin de déterminer si une ressource doit être définie ou si un comportement doit être appliqué.

> **Note**
>
> AWS CDK Il fournit autant de résolution que possible pendant le temps de synthèse afin de permettre une utilisation idiomatique et naturelle de votre langage de programmation.

Comme toute autre construction, les piles peuvent être composées en groupes. Le code suivant montre un exemple de service composé de trois piles : un plan de contrôle, un plan de données et des piles de surveillance. La structure du service est définie deux fois : une fois pour l'environnement bêta et une fois pour l'environnement de production.

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();
    }
}
```

Cette AWS CDK application se compose finalement de six piles, trois pour chaque environnement :

```
$ cdk ls

betacpDA8372D3
betadataE23DB2BA
betamon632BD457
prodcp187264CE
proddataF7378CE5
prodmon631A1083
```

Les noms physiques des AWS CloudFormation piles sont automatiquement déterminés en AWS CDK fonction du chemin de construction de la pile dans l'arbre. Par défaut, le nom d'une pile est dérivé de l'ID de construction de l'`Stack`objet. Cependant, vous pouvez spécifier un nom explicite en utilisant l'`stackName`accessoire (en Python`stack_name`), comme suit.

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"
});
```

Piles imbriquées

La NestedStackconstruction permet de contourner la limite de AWS CloudFormation 500 ressources pour les piles. Une pile imbriquée est considérée comme une seule ressource dans la pile qui la contient. Cependant, il peut contenir jusqu'à 500 ressources, y compris des piles imbriquées supplémentaires.

La portée d'une pile imbriquée doit être une `NestedStack` construction `Stack` or. La pile imbriquée n'a pas besoin d'être déclarée de manière lexicale dans sa pile parent. Il suffit de passer la pile parent comme premier paramètre (`scope`) lors de l'instanciation de la pile imbriquée. Hormis cette restriction, la définition des constructions dans une pile imbriquée fonctionne exactement de la même manière que dans une pile ordinaire.

Au moment de la synthèse, la pile imbriquée est synthétisée dans son propre AWS CloudFormation modèle, qui est téléchargé dans le compartiment AWS CDK intermédiaire lors du déploiement. Les piles imbriquées sont liées à leur pile parent et ne sont pas traitées comme des artefacts de déploiement indépendants. Ils ne sont pas répertoriés par `cdk list` et ne peuvent pas être déployés par`cdk deploy`.

Les références entre les piles parentes et les piles imbriquées sont automatiquement traduites en paramètres de pile et en sorties dans les AWS CloudFormation modèles générés, comme pour toute référence entre piles.

> **Warning**
>
> Les modifications apportées à la posture de sécurité ne sont pas affichées avant le déploiement pour les piles imbriquées. Ces informations ne sont affichées que pour les piles de niveau supérieur.

Constructions

Les constructions sont les éléments de base des AWS Cloud Development Kit (AWS CDK) applications. Une construction est un composant de votre application qui représente une ou plusieurs AWS CloudFormation ressources et leur configuration. Vous créez votre application, pièce par pièce, en important et en configurant des constructions.

Les constructions sont des classes que vous importez dans vos applications CDK. Les constructions sont disponibles dans la bibliothèque de AWS constructions. Vous pouvez également créer et distribuer vos propres constructions, ou utiliser des constructions créées par des développeurs tiers.

Les constructions font partie du modèle de programmation des constructions (CPM). Ils peuvent être utilisés avec d'autres outils tels que CDK pour Terraform (CDKTF), CDK pour Kubernetes (CDK8s) et. Projen

Rubriques

AWS Construire une bibliothèque

La AWS bibliothèque de constructions contient une collection de constructions développées et maintenues par AWS. Il est organisé en différents modules qui contiennent des constructions

représentant toutes les ressources disponibles sur AWS. Pour obtenir des informations de référence, consultez la référence de AWS CDK l'API.

Le package CDK principal est appeléaws-cdk-lib, et il contient la majorité de la bibliothèque AWS Construct. Il contient également des classes de base telles que Stack etApp.

Le nom réel du package CDK principal varie en fonction de la langue.

TypeScript

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

JavaScript

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

Python

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

Java

Danspom.xml, ajoutez	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Import	`import software.amazon.awscdk.App;`

C#

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

Go

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

> **Note**
>
> Si vous avez créé un projet CDK à l'aide decdk init, vous n'avez pas besoin de l'installer `aws-cdk-lib` manuellement.

La bibliothèque AWS Construct contient également le `constructs` package avec la classe `Construct` de base. Il se trouve dans son propre package car il est utilisé par d'autres outils basés sur des constructions AWS CDK, notamment CDK pour Terraform et CDK pour Kubernetes.

De nombreux tiers ont également publié des constructions compatibles avec le AWS CDK. Visitez Construct Hub pour découvrir l'écosystème AWS CDK des partenaires de construction.

Construisez des niveaux

Les constructions de la bibliothèque AWS de constructions sont classées en trois niveaux. Chaque niveau offre un niveau d'abstraction croissant. Plus l'abstraction est élevée, plus la configuration est facile et nécessite moins d'expertise. Plus l'abstraction est faible, plus la personnalisation est importante, ce qui nécessite plus d'expertise.

Constructions de niveau 1 (L1)

Les constructions L1, également appelées ressources CFN, sont les constructions de niveau inférieur et n'offrent aucune abstraction. Chaque construction L1 renvoie directement à une seule AWS CloudFormation ressource. Avec les constructions L1, vous importez une construction qui représente une ressource spécifique AWS CloudFormation . Vous définissez ensuite les propriétés de la ressource dans votre instance de construction.

Les constructions L1 sont idéales à utiliser lorsque vous connaissez les propriétés de vos AWS ressources AWS CloudFormation et que vous avez besoin d'un contrôle total sur leur définition.

Dans la AWS bibliothèque de constructions, les constructions L1 sont nommées en commençant par`Cfn`, suivi d'un identifiant pour la AWS CloudFormation ressource qu'elles représentent. Par exemple, la `CfnBucket` construction est une construction L1 qui représente une `AWS::S3::Bucket` AWS CloudFormation ressource.

Les constructions L1 sont générées à partir de la spécification de la AWS CloudFormation ressource. Si une ressource existe dans AWS CloudFormation, elle sera disponible AWS CDK sous forme de construction L1. Les nouvelles ressources ou propriétés peuvent prendre jusqu'à une semaine pour être disponibles dans la bibliothèque AWS Construct. Pour plus d'informations, consultez la référence aux types de AWS ressources et de propriétés dans le guide de AWS CloudFormation l'utilisateur.

Constructions de niveau 2 (L2)

Les constructions L2, également appelées constructions sélectionnées, sont soigneusement développées par l'équipe CDK et sont généralement le type de construction le plus utilisé. Les constructions L2 correspondent directement à des AWS CloudFormation ressources uniques, comme les constructions L1. Par rapport aux constructions L1, les constructions L2 fournissent une abstraction de niveau supérieur grâce à une API intuitive basée sur l'intention. Les constructions L2 incluent des configurations de propriétés par défaut judicieuses, des politiques de sécurité conformes aux meilleures pratiques et génèrent une grande partie du code standard et de la logique de collage pour vous.

Les constructions L2 fournissent également des méthodes d'assistance pour la plupart des ressources qui simplifient et accélèrent la définition des propriétés, des autorisations, des interactions basées sur des événements entre les ressources, etc.

La `s3.Bucket` classe est un exemple de construction L2 pour une ressource de bucket Amazon Simple Storage Service (Amazon S3).

La AWS bibliothèque de constructions contient des constructions L2 désignées comme stables et prêtes à être utilisées en production. Pour les constructions L2 en cours de développement, elles sont désignées comme expérimentales et proposées dans un module distinct.

Constructions de niveau 3 (L3)

Les constructions L3, également appelées modèles, constituent le plus haut niveau d'abstraction. Chaque construction L3 peut contenir un ensemble de ressources configurées pour fonctionner ensemble afin d'accomplir une tâche ou un service spécifique au sein de votre application. Les constructions L3 sont utilisées pour créer des AWS architectures complètes pour des cas d'utilisation particuliers dans votre application.

Pour fournir des conceptions de système complètes ou des parties importantes d'un système plus vaste, les constructions L3 proposent des configurations de propriétés par défaut bien définies. Ils sont construits autour d'une approche particulière visant à résoudre un problème et à fournir une solution. Avec les constructions L3, vous pouvez créer et configurer plusieurs ressources rapidement, avec un minimum d'entrées et de code.

La `ecsPatterns.ApplicationLoadBalancedFargateService` classe est un exemple de construction L3 qui représente un AWS Fargate service exécuté sur un cluster Amazon Elastic Container Service (Amazon ECS) et dirigé par un équilibreur de charge d'application.

À l'instar des constructions L2, les constructions L3 prêtes à être utilisées en production sont incluses dans la bibliothèque de constructions. AWS Les modules en cours de développement sont proposés dans des modules distincts.

Définition de constructions

Montage

La composition est le modèle clé pour définir des abstractions de haut niveau par le biais de constructions. Une construction de haut niveau peut être composée d'un nombre quelconque de constructions de niveau inférieur. Du bas vers le haut, vous utilisez des structures pour organiser les AWS ressources individuelles que vous souhaitez déployer. Vous utilisez les abstractions qui conviennent à votre objectif, avec autant de niveaux que vous le souhaitez.

Avec la composition, vous définissez des composants réutilisables et vous les partagez comme n'importe quel autre code. Par exemple, une équipe peut définir une construction qui met en œuvre les meilleures pratiques de l'entreprise pour une table Amazon DynamoDB, notamment la

sauvegarde, la réplication globale, le dimensionnement automatique et la surveillance. L'équipe peut partager le concept en interne avec d'autres équipes ou en public.

Les équipes peuvent utiliser des constructions comme n'importe quel autre package de bibliothèque. Lorsque la bibliothèque est mise à jour, les développeurs ont accès aux améliorations et aux corrections de bogues de la nouvelle version, comme pour toute autre bibliothèque de code.

Initialisation

Les constructions sont implémentées dans des classes qui étendent la classe de base `Construct`. Vous définissez une construction en instanciant la classe. Toutes les constructions prennent trois paramètres lorsqu'elles sont initialisées :

- scope — Le parent ou le propriétaire de la construction. Il peut s'agir d'une pile ou d'une autre construction. La portée détermine la place de la construction dans l'arbre de construction. Vous devez généralement transmettre `this` (`self`in Python), qui représente l'objet actuel, pour la portée.
- id — Identifiant qui doit être unique dans le champ d'application. L'identifiant sert d'espace de noms pour tout ce qui est défini dans la construction. Il est utilisé pour générer des identifiants uniques, tels que des noms de ressources et des identifiants AWS CloudFormation logiques.

  Les identifiants doivent uniquement être uniques au sein d'un périmètre. Cela vous permet d'instancier et de réutiliser des constructions sans vous soucier des constructions et des identifiants qu'elles peuvent contenir, et de composer des constructions sous forme d'abstractions de niveau supérieur. De plus, les portées permettent de faire référence à des groupes de constructions en une seule fois. Les exemples incluent le balisage ou la spécification de l'endroit où les constructions seront déployées.
- props — Ensemble de propriétés ou d'arguments de mots clés, selon le langage, qui définissent la configuration initiale de la construction. Les constructions de niveau supérieur fournissent davantage de valeurs par défaut, et si tous les éléments accessoires sont facultatifs, vous pouvez complètement omettre le paramètre props.

Configuration

La plupart des constructions acceptent `props` comme troisième argument (ou en Python, des arguments de mots clés) une collection nom/valeur qui définit la configuration de la construction. L'exemple suivant définit un bucket avec le chiffrement AWS Key Management Service (AWS KMS)

et l'hébergement statique de sites Web activés. Comme elle ne spécifie pas explicitement de clé de chiffrement, la `Bucket` construction en définit une nouvelle `kms.Key` et l'associe au bucket.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Go

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

```
})
```

Interaction avec les constructions

Les constructions sont des classes qui étendent la classe Construct de base. Après avoir instancié une construction, l'objet de construction expose un ensemble de méthodes et de propriétés qui vous permettent d'interagir avec la construction et de la transmettre comme référence à d'autres parties du système.

Le AWS CDK framework n'impose aucune restriction aux API des constructions. Les auteurs peuvent définir l'API de leur choix. Toutefois, les AWS constructions incluses dans la AWS bibliothèque de constructions, telles que`s3.Bucket`, suivent les directives et les modèles courants. Cela garantit une expérience cohérente sur toutes les AWS ressources.

La plupart AWS des constructions comportent un ensemble de méthodes d'octroi que vous pouvez utiliser pour accorder des autorisations AWS Identity and Access Management (IAM) sur cette construction à un principal. L'exemple suivant accorde au groupe IAM `data-science` l'autorisation de lire depuis le compartiment Amazon S3`raw-data`.

TypeScript

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

JavaScript

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

Python

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

Java

```
Bucket rawData = new Bucket(this, "raw-data");
```

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

C#

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

Go

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

Un autre modèle courant est que AWS les constructions définissent l'un des attributs de la ressource à partir de données fournies ailleurs. Les attributs peuvent inclure des noms de ressources Amazon (ARN), des noms ou des URL.

Le code suivant définit une AWS Lambda fonction et l'associe à une file d'attente Amazon Simple Queue Service (Amazon SQS) via l'URL de la file d'attente dans une variable d'environnement.

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()),
 },
})
```

Pour plus d'informations sur les modèles d'API les plus courants de la bibliothèque AWS Construct, consultezthe section called “Ressources”.

Structure de l'application et de la pile

Les `Stack` classes `App` et de la bibliothèque AWS de constructions sont des constructions uniques. Comparées à d'autres constructions, elles ne configurent pas AWS les ressources par elles-mêmes. Ils sont plutôt utilisés pour fournir un contexte à vos autres constructions. Toutes les constructions qui représentent AWS des ressources doivent être définies, directement ou indirectement, dans le cadre d'une `Stack` construction. `Stack`les constructions sont définies dans le cadre d'une `App` construction.

Pour en savoir plus sur les applications CDK, consultezAWS CDK applications. Pour en savoir plus sur les piles CDK, consultez. Piles

L'exemple suivant définit une application avec une pile unique. Au sein de la pile, une construction L2 est utilisée pour configurer une ressource de compartiment 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

Pile définie dans `HelloCdkStack.java` le fichier :

```
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();
    }
}
```

Application définie dans `HelloCdkApp.java` le fichier :

```
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
}
```

Travailler avec des constructions

Utilisation des constructions L1

Les constructions L1 mappent directement aux ressources individuelles AWS CloudFormation . Vous devez fournir la configuration requise pour la ressource.

Dans cet exemple, nous créons un `bucket` objet à l'aide de la construction `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"),
```

```
 })
```

Les propriétés de construction qui ne sont pas de simples booléens, chaînes, nombres ou conteneurs sont traitées différemment dans les langues prises en charge.

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, ces propriétés sont représentées par des types définis comme des classes internes de la construction L1. Par exemple, la propriété optionnelle `cors_configuration` de a `CfnBucket` nécessite un wrapper de type`CfnBucket.CorsConfigurationProperty`. Nous définissons ici `cors_configuration` sur une `CfnBucket` instance.

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

```
        )]
    )
)
```

Java

En Java, ces propriétés sont représentées par des types définis comme des classes internes de la construction L1. Par exemple, la propriété optionnelle `corsConfiguration` de a `CfnBucket` nécessite un wrapper de type`CfnBucket.CorsConfigurationProperty`. Nous définissons ici `corsConfiguration` sur une `CfnBucket` instance.

```
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#, ces propriétés sont représentées par des types définis comme des classes internes de la construction L1. Par exemple, la propriété optionnelle `CorsConfiguration` de a `CfnBucket` nécessite un wrapper de type`CfnBucket.CorsConfigurationProperty`. Nous définissons ici `CorsConfiguration` sur une `CfnBucket` instance.

```
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

Dans Go, ces types sont nommés à l'aide du nom de la construction L1, d'un trait de soulignement et du nom de la propriété. Par exemple, la propriété optionnelle `CorsConfiguration` de a `CfnBucket` nécessite un wrapper de type`CfnBucket_CorsConfigurationProperty`. Nous définissons ici `CorsConfiguration` sur une `CfnBucket` instance.

```
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

Vous ne pouvez pas utiliser de types de propriétés L2 avec des constructions L1, ou vice versa. Lorsque vous travaillez avec des constructions L1, utilisez toujours les types définis pour la construction L1 que vous utilisez. N'utilisez pas de types provenant d'autres constructions L1 (certains peuvent porter le même nom, mais ils ne sont pas du même type). Certaines de nos références d'API spécifiques au langage contiennent actuellement des erreurs dans les chemins d'accès aux types de propriétés L1, ou ne documentent pas du tout ces classes. Nous espérons pouvoir régler ce problème bientôt. En attendant, n'oubliez pas que ces types sont toujours des classes internes de la construction L1 avec laquelle ils sont utilisés.

Utilisation de constructions L2

Dans l'exemple suivant, nous définissons un compartiment Amazon S3 en créant un objet à partir de la construction `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`n'est pas le nom du bucket AWS CloudFormation créé. Il s'agit d'un identifiant logique attribué à la nouvelle construction dans le contexte de votre application CDK. La valeur PhysicalName sera utilisée pour nommer la AWS CloudFormation ressource.

Utilisation de constructions tierces

Construct Hub est une ressource qui vous aidera à découvrir des constructions supplémentaires provenant de AWS tiers et de la communauté des CDK open source.

Écrire vos propres constructions

En plus d'utiliser des constructions existantes, vous pouvez également écrire vos propres constructions et permettre à n'importe qui de les utiliser dans ses applications. Toutes les constructions sont égales dans le AWS CDK. Les constructions de la bibliothèque de constructions sont traitées de la même manière qu'une structure de AWS construction d'une bibliothèque tierce publiée viaNPM, Maven ou. PyPI Les constructions publiées dans le référentiel de packages interne de votre entreprise sont également traitées de la même manière.

Pour déclarer une nouvelle construction, créez une classe qui étend la classe de base Construct, dans le `constructs` package, puis suivez le modèle pour les arguments de l'initialiseur.

L'exemple suivant montre comment déclarer une construction qui représente un compartiment Amazon S3. Le compartiment S3 envoie une notification Amazon Simple Notification Service (Amazon SNS) chaque fois que quelqu'un y charge un fichier.

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**
>
> Notre `NotifyingBucket` construction n'hérite pas de `Bucket` mais plutôt de`Construct`. Nous utilisons la composition, et non l'héritage, pour regrouper un compartiment Amazon S3 et une rubrique Amazon SNS. En général, la composition est préférée à l'héritage lors du développement de AWS CDK constructions.

Le `NotifyingBucket` constructeur possède une signature de construction typique : `scope``id`, et`props`. Le dernier argument`props`, est facultatif (obtient la valeur par défaut`{}`) car tous les accessoires sont facultatifs. (La `Construct` classe de base ne prend aucun `props` argument.) Vous pouvez définir une instance de cette construction dans votre application sans`props`, par exemple :

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)
```

Vous pouvez également utiliser `props` (en Java, un paramètre supplémentaire) pour spécifier le préfixe de chemin à filtrer, par exemple :

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/"),
})
```

Généralement, vous souhaiterez également exposer certaines propriétés ou méthodes de vos constructions. Il n'est pas très utile d'avoir un sujet caché derrière votre construction, car les utilisateurs de votre construction ne peuvent pas s'y abonner. L'ajout d'une `topic` propriété permet aux consommateurs d'accéder au sujet interne, comme illustré dans l'exemple suivant :

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

Pour ce faire dans Go, nous aurons besoin d'un peu plus de plomberie. Notre `NewNotifyingBucket` fonction d'origine renvoyait un`awss3.Bucket`. Nous devrons étendre `Bucket` pour inclure un `topic` membre en créant une `NotifyingBucket` structure. Notre fonction renverra alors ce type.

```
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
}
```

Désormais, les consommateurs peuvent s'abonner au sujet, par exemple :

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))
```

En savoir plus

La vidéo suivante fournit un aperçu complet des constructions CDK et explique comment les utiliser dans vos applications CDK.

Explication des constructions CDK

Environnements

La cible est un environnement dans Région AWS lequel Compte AWS les piles sont déployées. Toutes les piles de votre application CDK sont explicitement ou implicitement associées à un environnement (). `env`

Rubriques

- Configuration des environnements
- Environnements d'amorçage

Configuration des environnements

Pour les piles de production, nous vous recommandons de spécifier explicitement l'environnement de chaque pile de votre application à l'aide de la `env` propriété. L'exemple suivant spécifie des environnements différents pour ses deux piles différentes.

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 });
```

Lorsque vous codez en dur le compte et la région cibles, comme indiqué dans l'exemple précédent, la pile est toujours déployée sur ce compte et cette région spécifiques. Pour rendre la pile déployable vers une cible différente, mais pour déterminer la cible au moment de la synthèse, votre pile peut utiliser deux variables d'environnement fournies par la AWS CDK CLI : `CDK_DEFAULT_ACCOUNT` et`CDK_DEFAULT_REGION`. Ces variables sont définies en fonction du AWS profil spécifié à l'aide de l'`--profile`option, ou du AWS profil par défaut si vous n'en spécifiez aucun.

Le fragment de code suivant montre comment accéder au compte et à la région transmis par la AWS CDK CLI dans votre pile.

TypeScript

Accédez aux variables d'environnement via l'`process`objet de Node.

> **Note**
>
> Vous avez besoin du `DefinitelyTyped` module pour l'utiliser `process` dans TypeScript. `cdk init`installe ce module pour vous. Cependant, vous devez installer ce module manuellement si vous travaillez sur un projet créé avant son ajout, ou si vous n'avez pas configuré votre projet à l'aide de`cdk init`.
>
> ```
> npm install @types/node
> ```

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

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

JavaScript

Accédez aux variables d'environnement via l'`process`objet de Node.

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

Python

Utilisez le `environ` dictionnaire du `os` module pour accéder aux variables d'environnement.

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

Java

`System.getenv()`À utiliser pour obtenir la valeur d'une variable d'environnement.

```
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#

`System.Environment.GetEnvironmentVariable()`À utiliser pour obtenir la valeur d'une variable d'environnement.

```
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() });
```

Spécifiez-le Région AWS à l'aide d'un code de région. Pour obtenir une liste, consultez la section Points de terminaison régionaux.

Cela AWS CDK fait la distinction entre ne pas spécifier du tout la `env` propriété et la spécifier à l'aide de `CDK_DEFAULT_ACCOUNT` et`CDK_DEFAULT_REGION`. Le premier implique que la pile doit synthétiser un modèle indépendant de l'environnement. Les constructions définies dans une telle pile ne peuvent utiliser aucune information concernant leur environnement. Par exemple, vous ne pouvez pas écrire du code `if (stack.region === 'us-east-1')` ou utiliser des fonctionnalités d'infrastructure telles que VPC.FromLookup (Python :`from_lookup`), qui nécessitent d'interroger

votre compte. AWS Ces fonctionnalités ne fonctionnent pas du tout tant que vous n'avez pas spécifié un environnement explicite ; pour les utiliser, vous devez le spécifier`env`.

Lorsque vous transmettez votre environnement à l'aide de `CDK_DEFAULT_ACCOUNT` et`CDK_DEFAULT_REGION`, la pile est déployée dans le compte et la région déterminés par la AWS CDK CLI au moment de la synthèse. Cela permet au code dépendant de l'environnement de fonctionner, mais cela signifie également que le modèle synthétisé peut être différent en fonction de la machine, de l'utilisateur ou de la session sous laquelle il est synthétisé. Ce comportement est souvent acceptable, voire souhaitable au cours du développement, mais il s'agirait probablement d'un anti-modèle pour une utilisation en production.

Vous pouvez définir comme `env` bon vous semble, en utilisant n'importe quelle expression valide. Par exemple, vous pouvez écrire votre pile de manière à prendre en charge deux variables d'environnement supplémentaires afin de remplacer le compte et la région au moment de la synthèse. Nous les `CDK_DEPLOY_ACCOUNT` appellerons `CDK_DEPLOY_REGION` ici, mais vous pouvez les nommer comme vous voulez, car ils ne sont pas définis par le AWS CDK. Dans l'environnement de la pile suivante, des variables d'environnement alternatives sont utilisées si elles sont définies. S'ils ne sont pas définis, ils retournent à l'environnement par défaut fourni par le 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() });
```

Lorsque l'environnement de votre pile est déclaré de cette façon, vous pouvez écrire un court script ou un fichier batch comme celui-ci pour définir les variables à partir des arguments de la ligne de commande, puis appeler`cdk deploy`. Tous les arguments situés au-delà des deux premiers sont transmis à `cdk deploy` et peuvent être utilisés pour spécifier des options ou des piles de ligne de commande.

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
```

Enregistrez le script sous`cdk-deploy-to.sh`, puis exécutez-le `chmod +x cdk-deploy-to.sh` pour le rendre exécutable.

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 version Windows du script fournit PowerShell les mêmes fonctionnalités que la version macOS/Linux. Il contient également des instructions permettant de l'exécuter sous forme de fichier batch afin qu'il puisse être facilement invoqué à partir d'une ligne de commande. Il doit être enregistré sous le nom`cdk-deploy-to.bat`. Le fichier `cdk-deploy-to.ps1` sera créé lorsque le fichier batch sera invoqué.

Vous pouvez ensuite écrire des scripts supplémentaires qui appellent le script « deploy-to » pour les déployer dans des environnements spécifiques (même plusieurs environnements par 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 %*
```

Lorsque vous déployez dans plusieurs environnements, déterminez si vous souhaitez poursuivre le déploiement dans d'autres environnements après l'échec d'un déploiement. L'exemple suivant permet d'éviter le déploiement dans le second environnement de production si le premier n'aboutit pas.

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 %*
```

Les développeurs peuvent toujours utiliser la `cdk deploy` commande normale pour effectuer le déploiement dans leurs propres AWS environnements à des fins de développement.

Si vous ne spécifiez pas d'environnement lorsque vous instanciez une pile, celle-ci est considérée comme indépendante de l'environnement. AWS CloudFormation les modèles synthétisés à partir d'une telle pile essaieront d'utiliser la résolution au moment du déploiement sur des attributs liés à l'environnement tels que`stack.account`, `stack.region` et (`stack.availabilityZones`Python :). `availability_zones`

Lors cdk deploy de l'utilisation pour déployer des piles indépendantes de l'environnement, ils AWS CDK CLI utiliseront le AWS CLI profil spécifié pour déterminer où déployer. Si aucun profil n'est spécifié, le profil par défaut est utilisé. Il AWS CDK CLI suit un protocole similaire au AWS CLI pour déterminer les AWS informations d'identification à utiliser lorsque vous effectuez des opérations sur votre AWS compte. Consultez the section called “AWS CDK Boîte à outils” pour plus de détails.

Dans une pile indépendante de l'environnement, toutes les constructions utilisant des zones de disponibilité verront deux zones de disponibilité, ce qui permettra de déployer la pile dans n'importe quelle région.

Environnements d'amorçage

Vous devez amorcer chaque environnement dans lequel vous allez déployer des piles de CDK. Le bootstrapping prépare l'environnement au déploiement. Pour en savoir plus, consultez Action d'amorçage.

Action d'amorçage

Le bootstrapping est le processus de préparation d'un environnement pour le déploiement. Le bootstrapping est une action ponctuelle que vous devez effectuer pour chaque environnement dans lequel vous déployez des ressources.

Rubriques

Environnements d'amorçage

> **Important**
>
> Des frais peuvent vous être AWS facturés pour les données stockées dans les ressources d'amorçage.

Le bootstrapping fournit des ressources dans votre environnement, telles qu'un bucket Amazon Simple Storage Service (Amazon S3) pour le stockage de fichiers et des rôles AWS Identity and Access Management (IAM) qui accordent les autorisations nécessaires pour effectuer des déploiements. Ces ressources sont provisionnées dans une AWS CloudFormation pile, appelée pile bootstrap. Il est généralement nommé`CDKToolkit`. Comme toute AWS CloudFormation pile, elle apparaîtra dans la AWS CloudFormation console de votre environnement une fois déployée.

> **Note**
>
> CDK v2 utilise un modèle bootstrap moderne. L'ancien modèle de CDK v1 n'est pas pris en charge dans la v2.

Les environnements sont indépendants. Si vous souhaitez effectuer un déploiement dans plusieurs environnements, chaque environnement doit être amorcé séparément.

Si vous tentez de déployer une application CDK dans un environnement qui n'a pas été amorcé, vous recevrez un message d'erreur vous rappelant de démarrer l'environnement.

Bootstrapping avec CDK Pipelines

Si vous utilisez CDK Pipelines pour effectuer un déploiement dans l'environnement d'un autre compte et que vous recevez un message comme celui-ci :

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

Ce message d'erreur signifie que les rôles IAM appropriés n'existent pas dans l'autre environnement. La cause la plus probable est que l'environnement n'a pas été amorcé. Démarrez l'environnement et réessayez.

> **Note**
>
> Si l'environnement est amorcé, ne supprimez pas et ne recréez pas la pile d'amorçage de l'environnement. La suppression de la pile bootstrap supprimera les AWS ressources initialement provisionnées dans l'environnement pour prendre en charge les déploiements de CDK. Cela empêchera le pipeline de fonctionner. Essayez plutôt de mettre à jour la pile bootstrap vers une nouvelle version en exécutant à nouveau la CLI `cdk bootstrap` commande CDK.

Comment démarrer

Lorsque vous démarrez un environnement, un AWS CloudFormation modèle est déployé dans cet environnement spécifique. Ce modèle fournit des ressources dans votre compte pour préparer votre environnement au déploiement.

Le modèle d'amorçage accepte des paramètres qui personnalisent certains aspects des ressources d'amorçage. Pour plus d'informations, consultez the section called “Personnalisation du bootstrap”.

Vous pouvez démarrer le bootstrap de l'une des manières suivantes :

- Utilisez la commande AWS CDK CLI.cdk bootstrap C'est la méthode la plus simple et elle fonctionne bien si vous n'avez que quelques environnements à démarrer.
- Déployez le modèle fourni par le à l' AWS CDK CLIaide d'un autre outil de AWS CloudFormation déploiement. Cela vous permet d'utiliser AWS CloudFormation StackSets ou AWS Control Tower également la AWS CloudFormation console ou le AWS CLI. Vous pouvez apporter de petites modifications au modèle avant le déploiement. Cette approche est plus flexible et convient aux déploiements à grande échelle.

Ce n'est pas une erreur de démarrer un environnement plusieurs fois. Si un environnement que vous bootstrap a déjà été bootstrap, sa pile de bootstrap sera mise à niveau si nécessaire. Sinon, il ne se passera rien.

Bootstrapping avec le AWS CDKCLI

Utilisez la `cdk bootstrap` commande pour démarrer un ou plusieurs AWS environnements.

L'exemple suivant permet de démarrer deux environnements :

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

Les exemples suivants montrent plusieurs méthodes d'amorçage d'environnements. Comme le montre le deuxième exemple, le `aws://` préfixe est facultatif lors de la spécification d'un environnement.

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

Lorsque vous l'exécutezcdk bootstrap, le CDK synthétise CLI toujours l'application CDK dans le répertoire actuel. Si vous ne spécifiez pas au moins un environnement, le CDK CLI amorcera tous les environnements référencés dans l'application.

Pour les piles indépendantes de l'environnement, le CDK CLI tentera de déterminer un environnement à partir de sources par défaut. Il peut s'agir d'un environnement spécifié à l'aide de l'--profileoption, à partir de variables d'environnement ou de AWS CLI sources par défaut. S'il est détecté, l'environnement est ensuite amorcé.

Par exemple, la commande suivante synthétise l' AWS CDK application actuelle à l'aide du `prod` AWS profil, puis amorce ses environnements.

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

Démarrage à partir du modèle AWS CloudFormation

Vous pouvez démarrer un environnement en obtenant et en déployant le modèle de démarrage. AWS CloudFormation

Pour obtenir une copie de ce modèle dans le fichier`bootstrap-template.yaml`, exécutez la commande suivante :

macOS/Linux

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

Windows

Sous Windows, PowerShell doit être utilisé pour préserver le codage du modèle.

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

Le modèle est également disponible dans le AWS CDK GitHub référentiel.

Déployez ce modèle à l'aide de la CLI CDK ou de votre mécanisme de déploiement préféré pour les AWS CloudFormation modèles. Pour déployer à l'aide de la CLI CDK, exécutezcdk bootstrap --template ***TEMPLATE_FILENAME***. Vous pouvez également le déployer à l'aide AWS CLI de la commande ci-dessous, ou le déployer sur un ou plusieurs comptes à la fois à l'aide de 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
```

Personnalisation du bootstrap

Il existe deux méthodes pour personnaliser le démarrage des ressources dans votre environnement :

- Utilisez les paramètres de ligne de commande avec la `cdk bootstrap` commande. Cela vous permet de modifier certains aspects du modèle.
- Modifiez le modèle de bootstrap par défaut et déployez-le vous-même. Cela vous donne un contrôle plus complet sur les ressources du bootstrap.

Les options de ligne de commande suivantes, lorsqu'elles sont utilisées avec CDK CLIcdk bootstrap, fournissent les ajustements couramment utilisés au modèle d'amorçage :

- - -bootstrap-bucket-name remplace le nom du compartiment Amazon S3. Il peut être nécessaire de modifier votre application CDK (voirthe section called “Synthétiseurs Stack”).
- --bootstrap-kms-key-idremplace la AWS KMS clé utilisée pour chiffrer le compartiment S3.
- --cloudformation-execution-policiesspécifie les ARN des politiques gérées qui doivent être associées au rôle de déploiement assumé AWS CloudFormation lors du déploiement de vos piles. Par défaut, les piles sont déployées avec des autorisations d'administrateur complètes à l'aide de cette `AdministratorAccess` politique.

  Les ARN de politique doivent être transmis sous la forme d'un argument de chaîne unique, les ARN individuels étant séparés par des virgules. Par exemple :

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

  > **Important**
  >
  > Pour éviter les échecs de déploiement, assurez-vous que les politiques que vous spécifiez sont suffisantes pour tous les déploiements que vous allez effectuer dans l'environnement amorcé.

- --qualifierest une chaîne qui est ajoutée aux noms de toutes les ressources de la pile bootstrap. Un qualificatif vous permet d'éviter les conflits de noms de ressources lorsque vous provisionnez plusieurs piles de bootstrap dans le même environnement. La valeur par défaut est `hnb659fds` (cette valeur n'a aucune signification).

  La modification du qualificatif nécessite également que votre application CDK transmette la valeur modifiée au synthétiseur de pile. Pour plus d’informations, consultez the section called “Synthétiseurs Stack”.
- --tagsajoute une ou plusieurs AWS CloudFormation balises à la pile bootstrap.

- --trustrépertorie les AWS comptes qui peuvent être déployés dans l'environnement en cours de démarrage.

  Utilisez cet indicateur lors du démarrage d'un environnement dans lequel un pipeline CDK sera déployé dans un autre environnement. Le compte qui effectue le bootstrap est toujours fiable.
- --trust-for-lookuprépertorie les AWS comptes susceptibles de rechercher des informations contextuelles à partir de l'environnement en cours de démarrage.

  Utilisez cet indicateur pour autoriser les comptes à synthétiser les piles qui seront déployées dans l'environnement, sans pour autant leur donner l'autorisation de déployer ces piles directement.
- --termination-protectionempêche la suppression de la pile bootstrap. Pour plus d'informations, consultez la section Protection d'une pile contre la suppression dans le Guide de AWS CloudFormation l'utilisateur.

> **Important**
>
> Le modèle bootstrap moderne accorde efficacement les autorisations implicites `--cloudformation-execution-policies` à n'importe quel AWS compte de la `--trust` liste. Par défaut, cela étend les autorisations de lecture et d'écriture à n'importe quelle ressource du compte bootstrap. Assurez-vous de configurer la pile d'amorçage avec des politiques et des comptes fiables avec lesquels vous êtes à l'aise.

Personnalisation du modèle

Lorsque vous avez besoin de plus de personnalisation que ce que le CDK CLI peut fournir, vous pouvez modifier le modèle de bootstrap en fonction de vos besoins. Tout d'abord, vous obtenez le modèle à l'aide de l'--show-templateoption. Voici un exemple :

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

Toutes les modifications que vous apportez doivent respecter le modèle de contrat de démarrage. Pour vous assurer que vos personnalisations ne seront pas accidentellement remplacées ultérieurement par une personne cdk bootstrap utilisant le modèle par défaut, modifiez la valeur par défaut du paramètre du `BootstrapVariant` modèle. La CLI CDK autorise uniquement le remplacement de la pile bootstrap par des modèles dont la version est identique `BootstrapVariant` ou supérieure à celle du modèle actuellement déployé.

Vous pouvez ensuite déployer votre modèle modifié comme décrit dansthe section called “Démarrage à partir du modèle AWS CloudFormation”, ou en utilisantcdk bootstrap --template.

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

Différences entre les modèles de bootstrap

Comme mentionné précédemment, la AWS CDK v1 supportait deux modèles d'amorçage, anciens et modernes. CDK v2 ne prend en charge que le modèle moderne. À titre de référence, voici les principales différences entre ces deux modèles.

Fonctionnalité	Legacy (v1 uniquement)	Moderne (v1 et v2)
Déploiements entre comptes	Non autorisée	Autorisé
AWS CloudFormation Autorisations	Déploie en utilisant les autorisations de l'utilisateur actuel (déterminées par le AWS profil, les variables d'environnement, etc.)	Déploie en utilisant les autorisations spécifiées lors du provisionnement de la pile bootstrap (par exemple, en utilisant) `--trust`
Contrôle de version	Une seule version de bootstrap stack est disponible	La pile Bootstrap est versionnée ; de nouvelles ressources peuvent être ajoutées dans les versions futures, et les AWS CDK applications peuvent nécessite r une version minimale
Ressources*	Compartiment Amazon S3	Compartiment Amazon S3
		AWS KMS key
		Rôles IAM
		Référentiel Amazon ECR
		Paramètre SSM pour le versionnement

Fonctionnalité	Legacy (v1 uniquement)	Moderne (v1 et v2)
Désignation des ressources	Généré automatiquement	Déterministe
Chiffrement des compartiments	Clé par défaut	Clé gérée par le client

* Nous ajouterons des ressources supplémentaires au modèle de bootstrap selon les besoins.

Un environnement qui a été amorcé à l'aide de l'ancien modèle doit être mis à niveau pour utiliser le modèle moderne de CDK v2 en redémarrant. Redéployez toutes les AWS CDK applications de l'environnement au moins une fois avant de supprimer l'ancien compartiment.

Synthétiseurs Stack

Votre AWS CDK application doit connaître les ressources d'amorçage dont elle dispose afin de synthétiser avec succès une pile pouvant être déployée. Le synthétiseur de pile est une AWS CDK classe qui contrôle la façon dont le modèle de la pile est synthétisé. Cela inclut la manière dont il utilise les ressources d'amorçage (par exemple, la manière dont il fait référence aux actifs stockés dans le compartiment bootstrap).

Les AWS CDK synthétiseurs à pile intégrés s'appellent`DefaultStackSynthesizer`. Il inclut des fonctionnalités pour les déploiements entre comptes et les déploiements de CDK Pipelines.

Vous pouvez passer un synthétiseur de pile à une pile lorsque vous l'instanciez à l'aide de la propriété. `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 vous ne fournissez pas la `synthesizer` propriété, elle `DefaultStackSynthesizer` est utilisée.

Personnalisation de la synthèse

En fonction des modifications que vous avez apportées au modèle de bootstrap, vous devrez peut-être également personnaliser la synthèse. Ils `DefaultStackSynthesizer` peuvent être personnalisés à l'aide des propriétés décrites ci-dessous.

Si aucune de ces propriétés ne fournit les personnalisations dont vous avez besoin, vous pouvez écrire votre synthétiseur sous la forme d'une classe qui implémente `IStackSynthesizer` (peut-être dérivée de). `DefaultStackSynthesizer`

Modifier le qualificatif

Le qualificatif est ajouté au nom des ressources bootstrap pour distinguer les ressources dans des piles de bootstrap distinctes. Pour déployer deux versions différentes de la pile bootstrap dans le même environnement (AWS compte et région), les piles doivent avoir des qualificatifs différents.

Cette fonctionnalité est destinée à isoler les noms entre les tests automatisés du CDK lui-même. À moins que vous ne puissiez définir très précisément les autorisations IAM accordées au rôle AWS CloudFormation d'exécution, le fait d'avoir deux piles de bootstrap différentes dans un seul compte ne présente aucun avantage en termes d'isolation des autorisations. Par conséquent, il n'est généralement pas nécessaire de modifier cette valeur.

Pour modifier le qualificatif, configurez l'un `DefaultStackSynthesizer` ou l'autre en instanciant le synthétiseur avec la propriété :

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Ou en configurant le qualificatif en tant que clé de contexte dans`cdk.json`.

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

Modification du nom des ressources

Toutes les autres `DefaultStackSynthesizer` propriétés concernent les noms des ressources dans le modèle d'amorçage. Vous ne devez fournir l'une de ces propriétés que si vous avez modifié le modèle de bootstrap et modifié le nom des ressources ou le schéma de dénomination.

Toutes les propriétés acceptent les espaces réservés spéciaux`${Qualifier}`, `${AWS::Partition}${AWS::AccountId}`, et`${AWS::Region}`. Ces espaces réservés sont remplacés par les valeurs du `qualifier` paramètre et par les valeurs de AWS partition, d'ID de compte et de région pour l'environnement de la pile, respectivement.

L'exemple suivant montre les propriétés les plus couramment utilisées `DefaultStackSynthesizer` ainsi que leurs valeurs par défaut, comme si vous instanciez le synthétiseur. Pour obtenir la liste complète, consultez 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,
})
```

Le modèle de contrat de démarrage

Les exigences de la pile d'amorçage dépendent du synthétiseur de pile utilisé. Si vous écrivez votre propre synthétiseur à pile, vous avez le contrôle total des ressources d'amorçage dont votre synthétiseur a besoin et de la manière dont il les trouve.

Cette section décrit les attentes qu'ils `DefaultStackSynthesizer` ont à l'égard du modèle d'amorçage.

Gestion des versions

Le modèle doit contenir une ressource permettant de créer un paramètre SSM avec un nom connu et une sortie reflétant la version du modèle.

```
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]
```

Rôles

`DefaultStackSynthesizer`Cela nécessite cinq rôles IAM pour cinq objectifs différents. Si vous n'utilisez pas les rôles par défaut, vous devez indiquer au synthétiseur les ARN des rôles que vous souhaitez utiliser.

Les rôles sont les suivants :

- Le rôle de déploiement est assumé par le AWS CDK Toolkit et par AWS CodePipeline le déploiement dans un environnement. Il `AssumeRolePolicy` contrôle qui peut être déployé dans l'environnement. Dans le modèle, vous pouvez voir les autorisations dont ce rôle a besoin.
- Le rôle de recherche est assumé par le AWS CDK Toolkit pour effectuer des recherches contextuelles dans un environnement. Il `AssumeRolePolicy` contrôle qui peut être déployé dans l'environnement. Les autorisations dont ce rôle a besoin sont indiquées dans le modèle.
- Le rôle de publication de fichiers et le rôle de publication d'images sont assumés par le AWS CDK Toolkit et par les AWS CodeBuild projets visant à publier des actifs dans un environnement. Ils sont utilisés pour écrire dans le compartiment S3 et dans le référentiel ECR, respectivement. Ces rôles nécessitent un accès en écriture à ces ressources.
- Le rôle AWS CloudFormation d'exécution est transmis AWS CloudFormation pour effectuer le déploiement proprement dit. Ses autorisations sont les autorisations sous lesquelles le déploiement s'exécutera. Les autorisations sont transmises à la pile sous forme de paramètre répertoriant les ARN des politiques gérées.

Outputs

Le AWS CDK Toolkit nécessite que les CloudFormation sorties suivantes existent sur la pile bootstrap.

- `BucketName`: le nom du compartiment de ressources du fichier
- `BucketDomainName`: le compartiment des actifs de fichiers au format de nom de domaine
- `BootstrapVersion`: la version actuelle de la pile bootstrap

Historique du modèle

Le modèle bootstrap est versionné et évolue au fil du temps avec lui-même. AWS CDK Si vous fournissez votre propre modèle de bootstrap, maintenez-le à jour avec le modèle canonique par défaut. Vous devez vous assurer que votre modèle continue de fonctionner avec toutes les fonctionnalités du CDK.

> **Note**
>
> Les versions antérieures du modèle bootstrap créaient un environnement AWS KMS key dans chaque environnement bootstrap par défaut. Pour éviter de payer la clé KMS, redémarrez ces environnements à l'aide de. `--no-bootstrap-customer-key` La valeur par défaut actuelle est l'absence de clé KMS, ce qui permet d'éviter ces frais.

Cette section contient une liste des modifications apportées dans chaque version.

Version du modèle	AWS CDK version	Modifications
1	1.40.0	Version initiale du modèle avec compartiment, clé, référentiel et rôles.
2	1,45,0	Divisez le rôle de publicati on de ressources en rôles de publication de fichiers et d'images distincts.
3	1.46,0	Ajoutez `FileAssetKeyArn` l'exportation pour pouvoir ajouter des autorisations de déchiffrement aux consommat eurs d'actifs.
4	1,61,0	AWS KMS les autorisations sont désormais implicites via Amazon S3 et ne sont plus nécessaires`FileAsetK`

Version du modèle	AWS CDK version	Modifications
		`eyArn` . Ajoutez le paramètre `CdkBootstrapVersion` SSM afin que la version de la pile bootstrap puisse être vérifiée sans connaître le nom de la pile.
5	1,87,0	Le rôle de déploiement peut lire le paramètre SSM.
6	1,108,0	Ajoutez un rôle de recherche distinct du rôle de déploieme nt.
6	1,109,0	Attachez une `aws-cdk:b ootstrap-role` balise aux rôles de déploiement, de publication de fichiers et de publication d'images.
7	1,110,0	Le rôle de déploiement ne peut plus lire directement les buckets dans le compte cible. (Cependant, ce rôle est en fait un administrateur et peut toujours utiliser ses AWS CloudFormation autorisations pour rendre le bucket lisible de toute façon).
8	1,114,0	Le rôle de recherche dispose d'autorisations complètes en lecture seule sur l'environ nement cible et possède également une `aws-cdk:b ootstrap-role` balise.

Version du modèle	AWS CDK version	Modifications
9	2.1.0	Empêche les téléchargements de ressources Amazon S3 d'être rejetés par le SCP de chiffrement couramment référencé.
10	2.4.0	Amazon ECR ScanOnPush est désormais activé par défaut.
11	2.18.0	Ajoute une politique permettant à Lambda d'extraire des dépôts Amazon ECR afin de survivre au redémarrage.
12	2.20.0	Ajoute le support pour les expériencescdk import.
13	2.25.0	Rend les images de conteneur dans les référentiels Amazon ECR créés par bootstrap immuables.
14	2.34.0	Désactive la numérisation d'images Amazon ECR au niveau du référentiel par défaut pour autoriser le démarrage des régions qui ne prennent pas en charge la numérisation d'images.
15	2,60,0	Les clés KMS ne peuvent pas être étiquetées.
16	2,69,0	Adresses du Security Hub détectant KMS.2.

Version du modèle	AWS CDK version	Modifications
17	2,72,0	Adresses au Security Hub détectant l'ECR.3.
18	2,80,0	Les modifications apportées à la version 16 ont été annulées car elles ne fonctionnent pas dans toutes les partitions et ne sont donc pas recommandées.
19	2,106,1	Annulation des modificat ions apportées à la version 18 où AccessControl la propriété avait été supprimée du modèle. (#27964)
20	2,119,0	Ajoutez une `ssm:GetPa rameters` action au rôle IAM de AWS CloudForm ation déploiement. Pour plus d'informations, voir #28336.

Conclusions du Security Hub

Si vous utilisez AWS Security Hub, vous pouvez voir des résultats publiés sur certaines des ressources créées par le processus d' AWS CDK amorçage. Les résultats du Security Hub vous aident à trouver des configurations de ressources dont vous devez vérifier l'exactitude et la sécurité. Nous avons examiné ces configurations de ressources spécifiques avec AWS Security et nous sommes convaincus qu'elles ne constituent pas un problème de sécurité.

[KMS.2] Les principaux IAM ne devraient pas avoir de politiques IAM en ligne autorisant les actions de déchiffrement sur toutes les clés KMS

Le rôle de déploiement (nom par défaut`cdk-hnb659fds-deploy-role-ACCOUNT-REGION`) est autorisé à lire les données chiffrées stockées dans Amazon S3. La politique n'autorise aucune donnée en elle-même : seules les données lues depuis Amazon S3 peuvent être déchiffrées, et

uniquement depuis les compartiments qui autorisent explicitement le rôle de déploiement à les lire via leur politique de compartiment, et les clés qui autorisent explicitement le rôle de déploiement à les déchiffrer à l'aide de leur politique de clé. Cette instruction est utilisée pour permettre à AWS CDK Pipelines d'effectuer des déploiements entre comptes.

Pourquoi Security Hub signale-t-il cela ? La politique contient une clause `Resource: *` combinée à une `Condition` clause ; Security Hub signale le`*`. `*`C'est nécessaire car au moment du démarrage du compte, la AWS KMS clé créée par AWS CDK Pipelines pour le bucket CodePipeline Artifact n'existe pas encore, nous ne pouvons donc pas référencer son ARN. En outre, Security Hub n'inclut pas la `Condition` clause de la déclaration de politique dans son raisonnement.

Et si je voulais corriger cette constatation ? Tant que les politiques de ressources de vos AWS KMS clés ne sont pas inutilement permissives, la politique de rôle actuelle n'autorise pas le rôle de déploiement à accéder à plus de données qu'il ne le devrait. Si vous souhaitez toujours vous débarrasser de cette découverte, vous pouvez le faire en personnalisant la pile bootstrap (en utilisant le processus décrit ci-dessus) de l'une des deux manières suivantes :

- Si vous n'utilisez pas de AWS CDK pipelines pour les déploiements entre comptes : supprimez l'instruction avec `Sid: PipelineCrossAccountArtifactsBucket` du rôle de déploiement ; ou
- Si vous utilisez des AWS CDK pipelines pour des déploiements entre comptes : après avoir déployé votre AWS CDK pipeline, recherchez le Key AWS KMS ARN du bucket Artifact et remplacez l'instruction `Resource: *` de l'`Sid: PipelineCrossAccountArtifactsBucket`instruction par le Key ARN réel.

Ressources

Les ressources sont celles que vous configurez pour être utilisées Services AWS dans vos applications. Les ressources sont une fonctionnalité de AWS CloudFormation. En configurant les ressources et leurs propriétés dans un AWS CloudFormation modèle, vous pouvez les déployer AWS CloudFormation pour provisionner vos ressources. Avec le AWS Cloud Development Kit (AWS CDK), vous pouvez configurer les ressources par le biais de constructions. Vous déployez ensuite votre application CDK, ce qui implique de synthétiser un AWS CloudFormation modèle et de le déployer pour AWS CloudFormation provisionner vos ressources.

Rubriques

- Configuration des ressources à l'aide de constructions

- Référencement des ressources
- Noms physiques des ressources
- Transmission d'identifiants de ressources uniques
- Octroi d'autorisations entre les ressources
- Indicateurs de ressources et alarmes
- Trafic réseau
- Gestion des événements
- Politiques de suppression

Configuration des ressources à l'aide de constructions

Comme décrit dansthe section called “Constructions”, il AWS CDK fournit une riche bibliothèque de classes de constructions, appelées AWS constructions, qui représentent toutes les AWS ressources.

Pour créer une instance d'une ressource à l'aide de la structure correspondante, transmettez le scope comme premier argument, l'ID logique de la construction et un ensemble de propriétés de configuration (accessoires). Par exemple, voici comment créer une file d'attente Amazon SQS AWS KMS chiffrée à l'aide de la construction SQS.Queue de la bibliothèque 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
});
```

Certains accessoires de configuration sont facultatifs et, dans de nombreux cas, ont des valeurs par défaut. Dans certains cas, tous les accessoires sont facultatifs et le dernier argument peut être totalement omis.

Attributs de ressource

La plupart des ressources de la bibliothèque AWS Construct exposent des attributs, qui sont résolus au moment du déploiement par AWS CloudFormation. Les attributs sont exposés sous forme de propriétés sur les classes de ressources avec le nom du type comme préfixe. L'exemple suivant montre comment obtenir l'URL d'une file d'attente Amazon SQS à l'aide de la propriété (`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
```

Consultez the section called “Jetons” pour plus d'informations sur la façon dont les attributs du AWS CDK moment du déploiement sont codés sous forme de chaînes.

Référencement des ressources

Lorsque vous configurez des ressources, vous devez souvent référencer les propriétés d'une autre ressource. Voici quelques exemples :

- Une ressource Amazon Elastic Container Service (Amazon ECS) nécessite une référence au cluster sur lequel elle s'exécute.
- Une CloudFront distribution Amazon nécessite une référence au bucket Amazon Simple Storage Service (Amazon S3) contenant le code source.

Vous pouvez référencer les ressources de l'une des manières suivantes :

- En transmettant une ressource définie dans votre application CDK, soit dans la même pile, soit dans une autre
- En transmettant un objet proxy faisant référence à une ressource définie dans votre AWS compte, créé à partir d'un identifiant unique de la ressource (tel qu'un ARN)

Si la propriété d'une construction représente une construction pour une autre ressource, son type est celui du type d'interface de la construction. Par exemple, la construction Amazon ECS utilise une propriété `cluster` de type`ecs.ICluster`. Un autre exemple est la construction de CloudFront distribution qui prend une propriété `sourceBucket` (Python :`source_bucket`) de type`s3.IBucket`.

Vous pouvez transmettre directement n'importe quel objet de ressource du type approprié défini dans la même AWS CDK application. L'exemple suivant définit un cluster Amazon ECS, puis l'utilise pour définir un service 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 });
```

Référencement de ressources dans une pile différente

Vous pouvez faire référence à des ressources d'une pile différente à condition qu'elles soient définies dans la même application et qu'elles se trouvent dans le même AWS environnement. Le schéma suivant est généralement utilisé :

- Stockez une référence à la construction en tant qu'attribut de la pile qui produit la ressource. (Pour obtenir une référence à la pile de la construction actuelle, utilisez`Stack.of(this)`.)
- Transmettez cette référence au constructeur de la pile qui consomme la ressource en tant que paramètre ou propriété. La pile consommatrice la transmet ensuite en tant que propriété à toute construction qui en a besoin.

L'exemple suivant définit une pile`stack1`. Cette pile définit un compartiment Amazon S3 et stocke une référence à la structure du compartiment en tant qu'attribut de la pile. Ensuite, l'application définit une deuxième pile`stack2`, qui accepte un bucket lors de l'instanciation. `stack2`peut, par exemple, définir une AWS Glue table qui utilise le bucket pour le stockage des données.

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 le AWS CDK détermine que la ressource se trouve dans le même environnement, mais dans une pile différente, il synthétise automatiquement les AWS CloudFormation exportations dans la pile productrice et un Fn : : ImportValue dans la pile consommatrice pour transférer ces informations d'une pile à l'autre.

Résoudre les blocages de dépendance

Le fait de référencer une ressource d'une pile dans une pile différente crée une dépendance entre les deux piles. Cela permet de s'assurer qu'ils sont déployés dans le bon ordre. Une fois les piles déployées, cette dépendance est concrète. Ensuite, la suppression de l'utilisation de la ressource partagée de la pile consommatrice peut entraîner un échec de déploiement inattendu. Cela se produit s'il existe une autre dépendance entre les deux piles qui les oblige à être déployées dans le même ordre. Cela peut également se produire sans dépendance si la pile productrice est simplement choisie par le CDK Toolkit pour être déployée en premier. L' AWS CloudFormation exportation est supprimée de la pile productrice car elle n'est plus nécessaire, mais la ressource exportée est toujours utilisée dans la pile consommatrice car sa mise à jour n'est pas encore déployée. Par conséquent, le déploiement de la pile de producteurs échoue.

Pour sortir de cette impasse, supprimez l'utilisation de la ressource partagée de la pile consommatrice. (Cela supprime l'exportation automatique de la pile de production.) Ajoutez ensuite manuellement la même exportation à la pile de production en utilisant exactement le même identifiant logique que l'exportation générée automatiquement. Supprimez l'utilisation de la ressource partagée dans la pile consommatrice et déployez les deux piles. Supprimez ensuite l'exportation manuelle (et la ressource partagée si elle n'est plus nécessaire) et déployez à nouveau les deux piles. La `exportValue()` méthode de la pile est un moyen pratique de créer l'exportation manuelle à cette fin. (Voir l'exemple dans la référence de méthode liée.)

Référencement des ressources de votre compte AWS

Supposons que vous souhaitiez utiliser une ressource déjà disponible dans votre AWS compte dans votre AWS CDK application. Il peut s'agir d'une ressource définie via la console, un AWS SDK, directement avec AWS CloudFormation ou dans une autre AWS CDK application. Vous pouvez transformer l'ARN de la ressource (ou un autre attribut d'identification, ou groupe d'attributs) en un objet proxy. L'objet proxy sert de référence à la ressource en appelant une méthode d'usine statique sur la classe de la ressource.

Lorsque vous créez un tel proxy, la ressource externe ne fait pas partie de votre AWS CDK application. Par conséquent, les modifications que vous apportez au proxy dans votre AWS CDK application n'affectent pas la ressource déployée. Le proxy peut toutefois être transmis à n'importe quelle AWS CDK méthode nécessitant une ressource de ce type.

L'exemple suivant montre comment référencer un bucket sur la base d'un bucket existant avec l'ARN arn:aws:s3 : my-bucket-name

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"
});
```

Regardons la `Vpc.fromLookup()`méthode de plus près. La `ec2.Vpc` construction étant complexe, vous pouvez sélectionner le VPC à utiliser avec votre application CDK de nombreuses manières. Pour résoudre ce problème, la construction VPC utilise une méthode `fromLookup` statique (Python :`from_lookup`) qui vous permet de rechercher le VPC Amazon souhaité en interrogeant votre AWS compte au moment de la synthèse.

Pour être utilisé`Vpc.fromLookup()`, le système qui synthétise la pile doit avoir accès au compte propriétaire de l'Amazon VPC. Cela est dû au fait que le CDK Toolkit interroge le compte pour trouver le bon Amazon VPC au moment de la synthèse.

De plus, ne `Vpc.fromLookup()` fonctionne que dans les piles définies avec un compte et une région explicites (voirthe section called “Environnements”). S'il AWS CDK essaie de rechercher un Amazon VPC à partir d'une pile indépendante de l'environnement, le CDK Toolkit ne sait pas quel environnement interroger pour trouver le VPC.

Vous devez fournir des `Vpc.fromLookup()` attributs suffisants pour identifier de manière unique un VPC dans votre AWS compte. Par exemple, il ne peut y avoir qu'un seul VPC par défaut, il suffit donc de spécifier le VPC comme VPC par défaut.

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 });
```

Vous pouvez également utiliser cette `tags` propriété pour rechercher des VPC par balise.
Vous pouvez ajouter des balises à l'Amazon VPC au moment de sa création en utilisant AWS CloudFormation ou le. AWS CDK Vous pouvez modifier les balises à tout moment après leur création à l'aide du SDK AWS Management Console AWS CLI, du ou d'un AWS SDK. Outre les balises que vous ajoutez vous-même, les balises suivantes AWS CDK sont automatiquement ajoutées à tous les VPC qu'il crée.

- Nom — Le nom du VPC.
- aws-cdk:subnet-name — Nom du sous-réseau.
- aws-cdk:subnet-type — Type du sous-réseau : public, privé ou isolé.

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" });
```

Les résultats de `Vpc.fromLookup()` sont mis en cache dans le `cdk.context.json` fichier du projet. (Consultez the section called “Contexte”.) Passez ce fichier au contrôle de version afin que votre application continue de faire référence au même Amazon VPC. Cela fonctionne même si vous modifiez ultérieurement les attributs de vos VPC de manière à sélectionner un autre VPC. Cela est particulièrement important si vous déployez la pile dans un environnement qui n'a pas accès au AWS compte qui définit le VPC, tel que CDK Pipelines.

Bien que vous puissiez utiliser une ressource externe partout où vous utiliseriez une ressource similaire définie dans votre AWS CDK application, vous ne pouvez pas la modifier. Par exemple, appeler `addToResourcePolicy` (Python :`add_to_resource_policy`) sur un appareil externe `s3.Bucket` ne fait rien.

Noms physiques des ressources

Les noms logiques des ressources dans AWS CloudFormation sont différents des noms des ressources qui apparaissent AWS Management Console après leur déploiement par AWS CloudFormation. Les AWS CDK appels à ces noms finaux sont des noms physiques.

Par exemple, AWS CloudFormation vous pouvez créer le compartiment Amazon S3 avec l'ID logique `Stack2MyBucket4DD88B4F` de l'exemple précédent et le nom physique`stack2mybucket4dd88b4f-iuv1rbv9z3to`.

Vous pouvez spécifier un nom physique lors de la création de constructions représentant des ressources à l'aide de la propriété <resourceType>Name. L'exemple suivant crée un compartiment Amazon S3 avec le nom physique`my-bucket-name`.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

L'attribution de noms physiques aux ressources présente certains inconvénients. AWS CloudFormation Plus important encore, toute modification des ressources déployées nécessitant le remplacement d'une ressource, telle que la modification des propriétés d'une ressource qui sont immuables après sa création, échouera si un nom physique est attribué à une ressource. Si vous vous retrouvez dans cet état, la seule solution est de supprimer la AWS CloudFormation pile, puis de déployer à nouveau l' AWS CDK application. Consultez la AWS CloudFormation documentation pour plus de détails.

Dans certains cas, par exemple lors de la création d'une AWS CDK application avec des références interenvironnementales, des noms physiques sont nécessaires pour AWS CDK qu'elle fonctionne correctement. Dans ces cas, si vous ne voulez pas vous embêter à trouver vous-même un nom physique, vous pouvez laisser le AWS CDK nom pour vous. Pour ce faire, utilisez la valeur spéciale`PhysicalName.GENERATE_IF_NEEDED`, comme suit.

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 });
```

Transmission d'identifiants de ressources uniques

Dans la mesure du possible, vous devez transmettre les ressources par référence, comme décrit dans la section précédente. Cependant, dans certains cas, vous n'avez pas d'autre choix que de faire référence à une ressource par l'un de ses attributs. Les exemples de cas d'utilisation incluent les suivants :

- Lorsque vous utilisez des AWS CloudFormation ressources de bas niveau.
- Lorsque vous devez exposer des ressources aux composants d'exécution d'une AWS CDK application, par exemple lorsque vous faites référence à des fonctions Lambda par le biais de variables d'environnement.

Ces identifiants sont disponibles sous forme d'attributs sur les ressources, tels que les suivants.

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

La AWS CDK liaison Java utilise des méthodes getter pour les attributs.

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

C#

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

L'exemple suivant montre comment transmettre le nom d'un bucket généré à une AWS Lambda fonction.

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
    }
});
```

Octroi d'autorisations entre les ressources

Les constructions de niveau supérieur permettent d'obtenir des autorisations avec le moindre privilège en proposant des API simples et basées sur l'intention pour exprimer les exigences en matière d'autorisation. Par exemple, de nombreuses constructions L2 proposent des méthodes d'octroi que vous pouvez utiliser pour accorder à une entité (telle qu'un rôle ou un utilisateur IAM) l'autorisation d'utiliser la ressource, sans avoir à créer manuellement des déclarations d'autorisation IAM.

L'exemple suivant crée les autorisations permettant au rôle d'exécution d'une fonction Lambda de lire et d'écrire des objets dans un compartiment Amazon S3 particulier. Si le compartiment Amazon S3 est chiffré à l'aide d'une AWS KMS clé, cette méthode autorise également le rôle d'exécution de la fonction Lambda à déchiffrer avec la clé.

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)
```

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

Les méthodes de subvention renvoient un `iam.Grant` objet. Utilisez l'`success`attribut de l'`Grant`objet pour déterminer si la subvention a été appliquée efficacement (par exemple, elle n'a peut-être pas été appliquée à des ressources externes). Vous pouvez également utiliser la méthode `assertSuccess` (Python :`assert_success`) de l'`Grant`objet pour vérifier que la subvention a été correctement appliquée.

Si aucune méthode d'autorisation spécifique n'est disponible pour le cas d'utilisation en question, vous pouvez utiliser une méthode d'autorisation générique pour définir une nouvelle autorisation avec une liste d'actions spécifiée.

L'exemple suivant montre comment accorder à une fonction Lambda l'accès à l'action 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");
```

De nombreuses ressources, telles que les fonctions Lambda, nécessitent qu'un rôle soit assumé lors de l'exécution du code. Une propriété de configuration vous permet de spécifier un`iam.IRole`. Si aucun rôle n'est spécifié, la fonction crée automatiquement un rôle spécifiquement pour cet usage. Vous pouvez ensuite utiliser des méthodes d'attribution sur les ressources pour ajouter des instructions au rôle.

Les méthodes de subvention sont conçues à l'aide d'API de niveau inférieur pour la gestion des politiques IAM. Les politiques sont modélisées sous forme d'[PolicyDocument](#)objets. Ajoutez des instructions directement aux rôles (ou au rôle attaché à une construction) à l'aide de la `addToRolePolicy` méthode (Python :`add_to_role_policy`), ou à la politique d'une ressource (telle qu'une `Bucket` politique) à l'aide de la méthode `addToResourcePolicy` (Python :`add_to_resource_policy`).

Indicateurs de ressources et alarmes

De nombreuses ressources émettent CloudWatch des métriques qui peuvent être utilisées pour configurer des tableaux de bord de surveillance et des alarmes. Les constructions de niveau supérieur utilisent des méthodes métriques qui vous permettent d'accéder aux métriques sans rechercher le nom correct à utiliser.

L'exemple suivant montre comment définir une alarme lorsque le nombre d'une file `ApproximateNumberOfMessagesNotVisible` d'attente Amazon SQS dépasse 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,
    // ..
});
```

S'il n'existe aucune méthode pour une métrique particulière, vous pouvez utiliser la méthode métrique générale pour spécifier le nom de la métrique manuellement.

Les métriques peuvent également être ajoutées aux CloudWatch tableaux de bord. Consultez CloudWatch.

Trafic réseau

Dans de nombreux cas, vous devez activer les autorisations sur un réseau pour qu'une application fonctionne, par exemple lorsque l'infrastructure informatique doit accéder à la couche de persistance. Les ressources qui établissent ou écoutent les connexions exposent les méthodes qui permettent les flux de trafic, notamment la définition de règles de groupe de sécurité ou de listes d'accès réseau.

Les ressources IConnectable ont une `connections` propriété qui constitue la passerelle vers la configuration des règles de trafic réseau.

Vous permettez aux données de circuler sur un chemin réseau donné à l'aide de `allow` méthodes. L'exemple suivant active les connexions HTTPS vers le Web et les connexions entrantes depuis le groupe Amazon EC2 Auto Scaling. `fleet2`

TypeScript

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

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

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

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

JavaScript

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

const fleet1 = asg.AutoScalingGroup();

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

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

Python

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

fleet1 = asg.AutoScalingGroup( ... )

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

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

Java

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

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

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

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

C#

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

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

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

Des ports par défaut sont associés à certaines ressources. Les exemples incluent l'écouteur d'un équilibreur de charge sur le port public et les ports sur lesquels le moteur de base de données accepte les connexions pour les instances d'une base de données Amazon RDS. Dans de tels cas, vous pouvez renforcer le contrôle du réseau sans avoir à spécifier manuellement le port. Pour ce faire, utilisez les `allowToDefaultPort` méthodes `allowDefaultPortFrom` et (Python :`allow_default_port_from`,`allow_to_default_port`).

L'exemple suivant montre comment activer les connexions depuis n'importe quelle adresse IPV4 et une connexion depuis un groupe Auto Scaling pour accéder à une base de données.

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");
```

Gestion des événements

Certaines ressources peuvent servir de sources d'événements. Utilisez la `addEventNotification` méthode (Python :`add_event_notification`) pour enregistrer une cible d'événement pour un type d'événement particulier émis par la ressource. En outre, les `addXxxNotification` méthodes offrent un moyen simple d'enregistrer un gestionnaire pour les types d'événements courants.

L'exemple suivant montre comment déclencher une fonction Lambda lorsqu'un objet est ajouté à un compartiment 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));
```

Politiques de suppression

Les ressources qui conservent des données persistantes, telles que les bases de données, les compartiments Amazon S3 et les registres Amazon ECR, sont soumises à une politique de suppression. La politique de suppression indique s'il faut supprimer les objets persistants lorsque la AWS CDK pile qui les contient est détruite. Les valeurs spécifiant la politique de suppression sont disponibles via l'`RemovalPolicy`énumération du AWS CDK `core` module.

> **Note**
>
> Les ressources autres que celles qui stockent des données de manière persistante peuvent également `removalPolicy` avoir une utilisation à des fins différentes. Par exemple, une version de fonction Lambda utilise un `removalPolicy` attribut pour déterminer si une version donnée est conservée lors du déploiement d'une nouvelle version. Elles ont des significations et des valeurs par défaut différentes de celles de la politique de suppression d'un bucket Amazon S3 ou d'une table DynamoDB.

Valeur	sens
`RemovalPolicy.CONSERVER`	Keep the contents of the resource when destroying the stack (default). The resource is orphaned from the stack and must be deleted manually. If you attempt to re-deploy the stack while the resource still exists, you will receive an error message due to a name conflict.
`RemovalPolicy.DÉTRUIRE`	The resource will be destroyed along with the stack.

AWS CloudFormation ne supprime pas les compartiments Amazon S3 contenant des fichiers, même si leur politique de suppression est définie sur. `DESTROY` Tenter de le faire est une AWS CloudFormation erreur. Pour AWS CDK supprimer tous les fichiers du bucket avant de le détruire, définissez la `autoDeleteObjects` propriété du bucket sur`true`.

Vous trouverez ci-dessous un exemple de création d'un compartiment Amazon S3 avec `RemovalPolicy` of `DESTROY` et `autoDeleteOjbects` défini sur`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
    });
}
```

Vous pouvez également appliquer une politique de suppression directement à la AWS CloudFormation ressource sous-jacente via la `applyRemovalPolicy()` méthode. Cette méthode est disponible sur certaines ressources dynamiques qui n'ont pas de `removalPolicy` propriété dans les accessoires de leur ressource L2. Voici quelques exemples :

- AWS CloudFormation piles
- Groupes d’utilisateurs Amazon Cognito
- Instances de base de données Amazon DocumentDB
- Volumes Amazon EC2
- Domaines Amazon OpenSearch Service
- Systèmes de fichiers Amazon FSx
- Files d'attente 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**
>
> Le « AWS CDK s » `RemovalPolicy` se AWS CloudFormation traduit par « `DeletionPolicy` ». Cependant, la valeur par défaut AWS CDK est de conserver les données, ce qui est le contraire de la AWS CloudFormation valeur par défaut.

Identifiants

Lorsque vous créez AWS Cloud Development Kit (AWS CDK) des applications, vous utiliserez de nombreux types d'identifiants et de noms. Pour les utiliser AWS CDK efficacement et éviter les erreurs, il est important de comprendre les types d'identifiants.

Les identifiants doivent être uniques dans le cadre dans lequel ils ont été créés ; ils n'ont pas besoin d'être uniques globalement dans votre AWS CDK application.

Si vous tentez de créer un identifiant avec la même valeur dans la même portée, une AWS CDK exception est générée.

Rubriques

Construire des identifiants

L'identifiant le plus courant est l'identifiant passé comme deuxième argument lors de l'instanciation d'un objet de construction. `id` Cet identifiant, comme tous les identifiants, doit uniquement être unique dans le cadre dans lequel il a été créé, ce qui est le premier argument lors de l'instanciation d'un objet de construction.

> **Note**
>
> Le `id` d'une pile est également l'identifiant que vous utilisez pour y faire référence dans lethe section called “AWS CDK Boîte à outils”.

Regardons un exemple où nous avons deux constructions avec l'identifiant `MyBucket` dans notre application. Le premier est défini dans le périmètre de la pile avec l'identifiant`Stack1`. Le second est défini dans le périmètre d'une pile avec l'identifiant`Stack2`. Comme ils sont définis dans des étendues différentes, cela ne provoque aucun conflit et ils peuvent coexister dans la même application sans problème.

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");
    }
}
```

Chemins

Les constructions d'une AWS CDK application forment une hiérarchie ancrée dans la App classe. Nous appelons chemin la collection d'identifiants d'une construction donnée, de sa construction parente, de son grand-parent, etc. jusqu'à la racine de l'arbre de construction.

Affiche AWS CDK généralement les chemins dans vos modèles sous forme de chaîne. Les identifiants des niveaux sont séparés par des barres obliques, en commençant par le nœud situé juste sous l'Appinstance racine, qui est généralement une pile. Par exemple, les chemins des deux ressources du compartiment Amazon S3 dans l'exemple de code précédent sont `Stack1/MyBucket` et`Stack2/MyBucket`.

Vous pouvez accéder au chemin de n'importe quelle construction par programmation, comme indiqué dans l'exemple suivant. Cela permet d'obtenir le chemin de `myConstruct` (ou`my_construct`, comme l'écriraient les développeurs Python). Étant donné que les identifiants doivent être uniques dans le cadre de leur création, leurs chemins sont toujours uniques au sein d'une AWS CDK application.

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 uniques

AWS CloudFormation exige que tous les identifiants logiques d'un modèle soient uniques. Pour cette raison, ils AWS CDK doivent être en mesure de générer un identifiant unique pour chaque construction d'une application. Les ressources ont des chemins globalement uniques (les noms de toutes les étendues de la pile vers une ressource spécifique). Par conséquent, il AWS CDK génère les identifiants uniques nécessaires en concaténant les éléments du chemin et en ajoutant un hachage à 8 chiffres. (Le hachage est nécessaire pour distinguer des chemins distincts, tels que `A/B/C` et`A/BC`, qui donneraient le même AWS CloudFormation identifiant. AWS CloudFormation les identifiants sont alphanumériques et ne peuvent pas contenir de barres obliques ou d'autres caractères de séparation.) Cette AWS CDK chaîne est appelée l'identifiant unique de la construction.

En général, votre AWS CDK application ne devrait pas avoir besoin de connaître les identifiants uniques. Vous pouvez toutefois accéder à l'identifiant unique de n'importe quelle construction par programmation, comme indiqué dans l'exemple suivant.

TypeScript

```
const uid: string = Names.uniqueId(myConstruct);
```

JavaScript

```
const uid = Names.uniqueId(myConstruct);
```

Python

```
uid = Names.unique_id(my_construct)
```

Java

```
String uid = Names.uniqueId(myConstruct);
```

C#

```
string uid = Names.Uniqueid(myConstruct);
```

L'adresse est un autre type d'identifiant unique qui distingue de manière unique les ressources CDK. Dérivé du hachage SHA-1 du chemin, il n'est pas lisible par l'homme. Cependant, sa longueur constante et relativement courte (toujours 42 caractères hexadécimaux) le rend utile dans les situations où l'identifiant unique « traditionnel » peut être trop long. Certaines constructions peuvent utiliser l'adresse du AWS CloudFormation modèle synthétisé au lieu de l'identifiant unique. Encore une fois, votre application ne devrait généralement pas avoir besoin de connaître les adresses de ses constructions, mais vous pouvez récupérer l'adresse d'une construction comme suit.

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;
```

Identifiants logiques

Les identifiants uniques servent d'identifiants logiques (ou de noms logiques) des ressources dans les AWS CloudFormation modèles générés pour les constructions qui représentent AWS les ressources.

Par exemple, le compartiment Amazon S3 créé dans l'exemple précédent `Stack2` génère une `AWS::S3::Bucket` ressource. L'ID logique de la ressource se trouve `Stack2MyBucket4DD88B4F` dans le AWS CloudFormation modèle obtenu. (Pour plus de détails sur la façon dont cet identifiant est généré, voirthe section called “ID uniques”.)

Stabilité de l'identifiant logique

Évitez de modifier l'ID logique d'une ressource après sa création. AWS CloudFormation identifie les ressources par leur identifiant logique. Par conséquent, si vous modifiez l'identifiant logique d'une ressource, AWS CloudFormation créez une nouvelle ressource avec le nouvel identifiant logique, puis supprimez l'identifiant existant. Selon le type de ressource, cela peut entraîner une interruption de service, une perte de données, ou les deux.

Jetons

Les jetons représentent des valeurs qui ne peuvent être résolues que plus tard dans le cycle de vie de l'application. Par exemple, le nom d'un bucket Amazon Simple Storage Service (Amazon S3) que vous définissez dans votre application CDK n'est attribué que lorsque AWS CloudFormation le modèle est synthétisé. Si vous imprimez l'`bucket.bucketName`attribut, qui est une chaîne, vous verrez qu'il contient quelque chose comme ceci :

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

C'est ainsi que l'on AWS CDK code un jeton dont la valeur n'est pas encore connue au moment de la construction, mais qui sera disponible ultérieurement. Ils AWS CDK appellent ces placeholders des jetons. Dans ce cas, il s'agit d'un jeton codé sous forme de chaîne.

Vous pouvez transmettre cette chaîne comme s'il s'agissait du nom du bucket. Dans l'exemple suivant, le nom du compartiment est spécifié en tant que variable d'environnement pour une AWS Lambda fonction.

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
    }
});
```

Lorsque le AWS CloudFormation modèle est finalement synthétisé, le jeton est rendu en tant qu' AWS CloudFormation intrinsèque`{ "Ref": "MyBucket" }`. Au moment du déploiement, AWS CloudFormation remplace ce nom intrinsèque par le nom réel du bucket créé.

Rubriques

- Jetons et encodages de jetons
- Jetons codés par chaîne
- Jetons codés sous forme de liste
- Jetons numérotés
- Valeurs paresseuses
- Conversion au format JSON

Jetons et encodages de jetons

Les jetons sont des objets qui implémentent l'interface IResolvable, qui contient une seule `resolve` méthode. Il AWS CDK appelle cette méthode pendant la synthèse pour produire la valeur finale du AWS CloudFormation modèle. Les jetons participent au processus de synthèse pour produire des valeurs arbitraires de tout type.

> **Note**
>
> Il est rare que vous travailliez directement avec l'`IResolvable`interface. Vous ne verrez probablement que des versions de jetons codées par chaîne.

Les autres fonctions n'acceptent généralement que des arguments de type basique, tels que `string` ou`number`. Pour utiliser des jetons dans ces cas, vous pouvez les encoder dans l'un des trois types suivants en utilisant des méthodes statiques sur la classe CDK.Token.

- `Token.asString`pour générer un codage de chaîne (ou appeler `.toString()` l'objet jeton)
- `Token.asList`pour générer un codage de liste
- `Token.asNumber`pour générer un codage numérique

Ils prennent une valeur arbitraire, qui peut être un`IResolvable`, et les encodent en une valeur primitive du type indiqué.

> **Important**
>
> Étant donné que l'un des types précédents peut potentiellement être un jeton codé, soyez prudent lorsque vous analysez ou essayez de lire leur contenu. Par exemple, si vous essayez d'analyser une chaîne pour en extraire une valeur et que la chaîne est un jeton codé, votre analyse échoue. De même, si vous essayez d'interroger la longueur d'un tableau ou d'effectuer des opérations mathématiques avec un nombre, vous devez d'abord vérifier qu'il ne s'agit pas de jetons codés.

Pour vérifier si une valeur contient un jeton non résolu, appelez la méthode `Token.isUnresolved` (Python :`is_unresolved`).

L'exemple suivant confirme qu'une valeur de chaîne, qui peut être un jeton, ne comporte pas plus de 10 caractères.

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 le nom est un jeton, la validation n'est pas effectuée et une erreur peut encore se produire à un stade ultérieur du cycle de vie, par exemple lors du déploiement.

> **Note**
>
> Vous pouvez utiliser des codages de jetons pour échapper au système de types. Par exemple, vous pouvez coder en chaîne un jeton qui produit une valeur numérique au moment de la synthèse. Si vous utilisez ces fonctions, il est de votre responsabilité de vous assurer que votre modèle passe à un état utilisable après la synthèse.

Jetons codés par chaîne

Les jetons codés sous forme de chaîne ressemblent à ce qui suit.

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

Ils peuvent être transmis comme des chaînes ordinaires et peuvent être concaténés, comme le montre l'exemple suivant.

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";
```

Vous pouvez également utiliser l'interpolation de chaînes, si votre langue le permet, comme indiqué dans l'exemple suivant.

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";
```

Évitez de manipuler la chaîne d'une autre manière. Par exemple, la prise d'une sous-chaîne d'une chaîne risque de casser le jeton de chaîne.

Jetons codés sous forme de liste

Les jetons codés sous forme de liste ressemblent à ce qui suit :

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

La seule chose sûre à faire avec ces listes est de les transmettre directement à d'autres constructions. Les jetons sous forme de liste de chaînes ne peuvent pas être concaténés, et aucun élément ne peut être extrait du jeton. Le seul moyen sûr de les manipuler est d'utiliser des fonctions AWS CloudFormation intrinsèques telles que FN.select.

Jetons numérotés

Les jetons codés par des nombres sont un ensemble de minuscules nombres négatifs à virgule flottante qui ressemblent à ce qui suit.

```
-1.8881545897087626e+289
```

Comme pour les jetons de liste, vous ne pouvez pas modifier la valeur numérique, car cela risque de casser le jeton numérique. La seule opération autorisée est de transmettre la valeur à une autre construction.

Valeurs paresseuses

En plus de représenter les valeurs de temps de déploiement, telles que les AWS CloudFormation paramètres, les jetons sont également couramment utilisés pour représenter les valeurs paresseuses au moment de la synthèse. Il s'agit de valeurs pour lesquelles la valeur finale sera déterminée avant la fin de la synthèse, mais pas au moment où la valeur est construite. Utilisez des jetons pour transmettre une chaîne littérale ou une valeur numérique à une autre construction, tandis que la valeur réelle au moment de la synthèse peut dépendre d'un calcul qui n'a pas encore été effectué.

Vous pouvez créer des jetons représentant des valeurs paresseuses au moment de la synthèse à l'aide de méthodes statiques appliquées à la `Lazy` classe, telles que Lazy.String et Lazy.Number. Ces méthodes acceptent un objet dont la `produce` propriété est une fonction qui accepte un argument de contexte et renvoie la valeur finale lorsqu'elle est appelée.

L'exemple suivant crée un groupe Auto Scaling dont la capacité est déterminée après sa création.

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;
```

Conversion au format JSON

Parfois, vous souhaitez produire une chaîne JSON contenant des données arbitraires, sans savoir si les données contiennent des jetons. Pour encoder correctement en JSON toute structure de données, qu'elle contienne ou non des jetons, utilisez la pile de méthodes. toJsonString, comme le montre l'exemple suivant.

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
});
```

Paramètres

Les paramètres sont des valeurs personnalisées fournies au moment du déploiement. Les paramètres sont une fonctionnalité de AWS CloudFormation. Comme il AWS Cloud Development Kit (AWS CDK) synthétise les AWS CloudFormation modèles, il prend également en charge les paramètres de temps de déploiement.

Rubriques

À propos des paramètres

À l'aide de AWS CDK, vous pouvez définir des paramètres, qui peuvent ensuite être utilisés dans les propriétés des constructions que vous créez. Vous pouvez également déployer des piles contenant des paramètres.

Lorsque vous déployez le AWS CloudFormation modèle à l'aide du AWS CDK Toolkit, vous fournissez les valeurs des paramètres sur la ligne de commande. Si vous déployez le modèle via la AWS CloudFormation console, vous êtes invité à saisir les valeurs des paramètres.

En général, nous vous déconseillons d'utiliser AWS CloudFormation des paramètres avec le AWS CDK. Les méthodes habituelles pour transmettre des valeurs aux AWS CDK applications sont les valeurs contextuelles et les variables d'environnement. Comme elles ne sont pas disponibles au moment de la synthèse, les valeurs des paramètres ne peuvent pas être facilement utilisées pour le contrôle du flux ou à d'autres fins dans votre application CDK.

> **Note**
>
> Pour contrôler le flux avec des paramètres, vous pouvez utiliser `CfnCondition`des constructions, bien que cela soit gênant par rapport aux `if` instructions natives.

L'utilisation de paramètres nécessite que vous soyez attentif au comportement du code que vous écrivez au moment du déploiement, ainsi qu'au moment de la synthèse. Il est donc plus difficile de comprendre et de raisonner votre AWS CDK candidature, dans de nombreux cas pour peu d'avantages.

En général, il est préférable que votre application CDK accepte les informations nécessaires d'une manière bien définie et qu'elle les utilise directement pour déclarer des constructions dans votre application CDK. Un AWS CloudFormation modèle AWS CDK idéal généré est concret et il ne reste aucune valeur à spécifier au moment du déploiement.

Il existe toutefois des cas d'utilisation pour lesquels AWS CloudFormation les paramètres sont particulièrement adaptés. Si vous avez des équipes distinctes qui définissent et déploient l'infrastructure, par exemple, vous pouvez utiliser des paramètres pour rendre les modèles générés plus largement utiles. De plus, étant donné qu'il AWS CDK prend en charge AWS CloudFormation les paramètres, vous pouvez les utiliser AWS CDK avec AWS des services utilisant AWS CloudFormation des modèles (tels que Service Catalog). Ces AWS services utilisent des paramètres pour configurer le modèle en cours de déploiement.

Définition des paramètres

Utilisez la `CfnParameter`classe pour définir un paramètre. Vous devez spécifier au moins un type et une description pour la plupart des paramètres, bien que les deux soient techniquement facultatifs. La description apparaît lorsque l'utilisateur est invité à saisir la valeur du paramètre dans la AWS CloudFormation console. Pour plus d'informations sur les types disponibles, consultez la section Types.

> **Note**
>
> Vous pouvez définir des paramètres dans n'importe quelle étendue. Cependant, nous vous recommandons de définir les paramètres au niveau de la pile afin que leur identifiant logique ne change pas lorsque vous refactorisez votre code.

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"
});
```

Utilisation de paramètres

Une `CfnParameter` instance expose sa valeur à votre AWS CDK application via un jeton. Comme tous les jetons, le jeton du paramètre est résolu au moment de la synthèse. Mais il s'agit d'une référence au paramètre défini dans le AWS CloudFormation modèle (qui sera résolu au moment du déploiement), plutôt que d'une valeur concrète.

Vous pouvez récupérer le jeton sous forme d'instance de la `Token` classe, ou sous forme de chaîne, de liste de chaînes ou de codage numérique. Votre choix dépend du type de valeur requis par la classe ou la méthode avec laquelle vous souhaitez utiliser le paramètre.

TypeScript

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

JavaScript

Property	kind of value
`value`	`Jeton` 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
`value`	`Jeton` class instance
`valeur_en tant que liste`	The token represented as a string list
`valeur_en tant que nombre`	The token represented as a number
`valeur_en tant que chaîne`	The token represented as a string

Java

Property	kind of value
`Obtenir une valeur ()`	`Jeton` class instance
`getValueAsListe ()`	The token represented as a string list
`getValueAsNuméro ()`	The token represented as a number
`getValueAsChaîne ()`	The token represented as a string

C#

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

Par exemple, pour utiliser un paramètre dans une `Bucket` définition :

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
};
```

Déploiement avec paramètres

Un modèle généré contenant des paramètres peut être déployé de la manière habituelle via la AWS CloudFormation console. Vous êtes invité à saisir les valeurs de chaque paramètre.

Le AWS CDK Toolkit (outil de ligne de `cdk` commande) permet également de spécifier des paramètres lors du déploiement. Vous les fournissez sur la ligne de commande après le `--parameters` drapeau. Vous pouvez déployer une pile qui utilise le `uploadBucketName` paramètre, comme dans l'exemple suivant.

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

Pour définir plusieurs paramètres, utilisez plusieurs `--parameters` indicateurs.

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

Si vous déployez plusieurs piles, vous pouvez spécifier une valeur différente pour chaque paramètre pour chaque pile. Pour ce faire, préfixez le nom du paramètre avec le nom de la pile et deux points.

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

Par défaut, le AWS CDK conserve les valeurs des paramètres des déploiements précédents et les utilise dans les déploiements suivants si elles ne sont pas spécifiées explicitement. Utilisez l'`--no-previous-parameters`indicateur pour exiger que tous les paramètres soient spécifiés.

Identification

Les balises sont des éléments clé-valeur informatifs que vous pouvez ajouter aux constructions de votre application. AWS CDK Une balise appliquée à une construction donnée s'applique également à tous ses enfants étiquetables. Les balises sont incluses dans le AWS CloudFormation modèle synthétisé à partir de votre application et sont appliquées aux AWS ressources qu'elle déploie. Vous pouvez utiliser des balises pour identifier et classer les ressources aux fins suivantes :

- Simplification de la gestion
- Répartition des coûts
- Contrôle d’accès
- Tout autre objectif que vous concevez

> **Tip**
>
> Pour plus d'informations sur la manière dont vous pouvez utiliser les balises avec vos AWS ressources, consultez les meilleures pratiques en matière de balisage AWS des ressources dans le AWS livre blanc.

Rubriques

Utilisation de balises

La `Tags`classe inclut la méthode statique`of()`, grâce à laquelle vous pouvez ajouter ou supprimer des balises à la construction spécifiée.

- `Tags.of(SCOPE).add()`applique une nouvelle balise à la construction donnée et à tous ses enfants.
- `Tags.of(SCOPE).remove()`supprime une balise de la construction donnée et de tous ses enfants, y compris les balises qu'une construction enfant peut s'être appliquée à elle-même.

> **Note**
>
> Le balisage est implémenté à l'aide dethe section called “Aspects”. Les aspects permettent d'appliquer une opération (telle que le balisage) à toutes les constructions d'une portée donnée.

L'exemple suivant applique la clé de balise avec la valeur de valeur à une construction.

TypeScript

```
Tags.of(myConstruct).add('key', 'value');
```

JavaScript

```
Tags.of(myConstruct).add('key', 'value');
```

Python

```
Tags.of(my_construct).add("key", "value")
```

Java

```
Tags.of(myConstruct).add("key", "value");
```

C#

```
Tags.Of(myConstruct).Add("key", "value");
```

L'exemple suivant supprime la clé de balise d'une construction.

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 vous utilisez `Stage` des constructions, appliquez la balise au `Stage` niveau ou en dessous. Les balises ne sont pas appliquées au-delà `Stage` des limites.

Priorités des tags

AWS CDK Applique et supprime les balises de manière récursive. En cas de conflit, l'opération de balisage ayant la priorité la plus élevée l'emporte. (Les priorités sont définies à l'aide de la `priority`

propriété facultative.) Si les priorités de deux opérations sont les mêmes, l'opération de balisage la plus proche du bas de l'arbre de construction gagne. Par défaut, l'application d'une balise a une priorité de 100 (sauf pour les balises ajoutées directement à une AWS CloudFormation ressource, qui ont une priorité de 50). La priorité par défaut pour supprimer un tag est 200.

Ce qui suit applique une balise avec une priorité de 300 à une construction.

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 });
```

Propriétés facultatives

Les balises `properties`permettent d'affiner la manière dont les balises sont appliquées ou supprimées des ressources. Toutes les propriétés sont facultatives.

`applyToLaunchedInstances`(Python :`apply_to_launched_instances`)

Disponible uniquement pour add (). Par défaut, les balises sont appliquées aux instances lancées dans un groupe Auto Scaling. Définissez cette propriété sur false pour ignorer les instances lancées dans un groupe Auto Scaling.

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

Utilisez-les pour manipuler les balises uniquement sur un sous-ensemble de ressources, en fonction des types de AWS CloudFormation ressources. Par défaut, l'opération est appliquée à toutes les ressources du sous-arbre de construction, mais cela peut être modifié en incluant ou en excluant certains types de ressources. L'exclusion a priorité sur l'inclusion, si les deux sont spécifiées.

`priority`

Utilisez-le pour définir la priorité de cette opération par rapport aux autres `Tags.remove()` opérations `Tags.add()` et. Les valeurs les plus élevées ont priorité sur les valeurs faibles. La valeur par défaut est de 100 pour les opérations d'ajout (50 pour les balises appliquées directement aux AWS CloudFormation ressources) et de 200 pour les opérations de suppression.

L'exemple suivant applique le tag tagname avec la valeur valeur et la priorité 100 aux ressources de type AWS::Xxx::Yyydans la construction. Il n'applique pas le tag aux instances lancées dans un groupe Amazon EC2 Auto Scaling ni aux ressources de ce type. AWS::Xxx::Zzz (Ce sont des espaces réservés pour deux types de AWS CloudFormation ressources arbitraires mais différents.)

TypeScript

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

JavaScript

```
Tags.of(myConstruct).add('tagname', 'value', {
  applyToLaunchedInstances: false,
  includeResourceTypes: ['AWS::Xxx::Yyy'],
```

```
  excludeResourceTypes: ['AWS::Xxx::Zzz'],
  priority: 100
});
```

Python

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

Java

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

C#

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

L'exemple suivant supprime le tag tagname avec la priorité 200 des ressources de type AWS::Xxx::Yyydans la construction, mais pas des ressources de type 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
});
```

Exemple

L'exemple suivant ajoute la clé de balise StackTypeavec valeur TheBestà toute ressource créée dans le `Stack` nom`MarketingSystem`. Il le supprime ensuite à nouveau de toutes les ressources, à l'exception des sous-réseaux Amazon EC2 VPC. Le résultat est que seuls les sous-réseaux ont la balise appliquée.

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"]
});
```

Le code suivant permet d'obtenir le même résultat. Déterminez quelle approche (inclusion ou exclusion) rend votre intention plus claire.

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"]
});
```

Marquage de constructions individuelles

`Tags.of(scope).add(key, value)`est la méthode standard pour ajouter des balises aux constructions du AWS CDK. Son comportement arborescent, qui balise de manière récursive toutes les ressources étiquetables dans le cadre d'une portée donnée, correspond presque toujours à ce que vous recherchez. Cependant, vous devez parfois étiqueter une ou plusieurs constructions arbitraires spécifiques.

L'un de ces cas implique l'application de balises dont la valeur est dérivée d'une propriété de la construction étiquetée. L'approche de balisage standard applique de manière récursive la même clé et la même valeur à toutes les ressources correspondantes du périmètre. Cependant, ici, la valeur peut être différente pour chaque construction étiquetée.

Les balises sont implémentées à l'aide d'aspects, et le CDK appelle la `visit()` méthode de la balise pour chaque construction dans le cadre que vous avez spécifié en utilisant`Tags.of(scope)`. Nous pouvons appeler `Tag.visit()` directement pour appliquer une balise à une seule construction.

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);
```

Vous pouvez étiqueter toutes les constructions d'une portée, mais laisser les valeurs des balises dériver des propriétés de chaque construction. Pour ce faire, écrivez un aspect et appliquez la balise dans la `visit()` méthode de l'aspect, comme indiqué dans l'exemple précédent. Ajoutez ensuite l'aspect à la portée souhaitée à l'aide de`Aspects.of(scope).add(aspect)`.

L'exemple suivant applique une balise à chaque ressource d'une pile contenant le chemin de la ressource.

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 logique du balisage conditionnel, y compris les priorités, les types de ressources, etc., est intégrée à la `Tag` classe. Vous pouvez utiliser ces fonctionnalités lorsque vous appliquez des balises à des ressources arbitraires ; la balise n'est pas appliquée si les conditions ne sont pas remplies. De plus, la `Tag` classe balise uniquement les ressources balisables, vous n'avez donc pas besoin de tester si une construction est balisable avant d'appliquer une balise.

Assets

Les actifs sont des fichiers locaux, des répertoires ou des images Docker qui peuvent être regroupés dans des AWS CDK bibliothèques et des applications. Par exemple, un actif peut être un répertoire contenant le code du gestionnaire d'une AWS Lambda fonction. Les actifs peuvent représenter n'importe quel artefact dont l'application a besoin pour fonctionner.

Le didacticiel vidéo suivant fournit un aperçu complet des actifs du CDK et explique comment les utiliser dans votre infrastructure en tant que code (iAc).

Explication des actifs CDK

Vous ajoutez des actifs via des API exposées par des AWS constructions spécifiques. Par exemple, lorsque vous définissez une construction Lambda.Function, la propriété code vous permet de transmettre un actif (répertoire). `Function`utilise des actifs pour regrouper le contenu du répertoire et l'utiliser pour le code de la fonction. De même, ecs. ContainerImage.fromAsset utilise une image Docker créée à partir d'un répertoire local lors de la définition d'une tâche Amazon ECS.

Les actifs en détail

Lorsque vous faites référence à un actif dans votre application, l'assemblage cloud synthétisé à partir de votre application inclut des informations de métadonnées ainsi que des instructions pour la AWS CDK CLI. Les instructions indiquent où trouver la ressource sur le disque local et quel type de

regroupement effectuer en fonction du type de ressource, par exemple un répertoire à compresser (zip) ou une image Docker à créer.

AWS CDK Génère un hachage source pour les actifs. Cela peut être utilisé au moment de la construction pour déterminer si le contenu d'un actif a changé.

Par défaut, AWS CDK crée une copie de la ressource dans le répertoire d'assemblage du cloud, qui est par défaut`cdk.out`, sous le hachage source. Ainsi, l'assemblage du cloud est autonome. Ainsi, s'il est déplacé vers un autre hôte pour le déploiement, il peut toujours être déployé. Consultez the section called “Assemblages cloud” pour plus de détails.

Lors du AWS CDK déploiement d'une application qui référence des actifs (soit directement par le code de l'application, soit par le biais d'une bibliothèque), la AWS CDK CLI prépare et publie d'abord les actifs dans un compartiment Amazon S3 ou un référentiel Amazon ECR. (Le compartiment ou le référentiel S3 est créé pendant le démarrage.) Ce n'est qu'alors que les ressources définies dans la pile sont déployées.

Cette section décrit les API de bas niveau disponibles dans le framework.

Types d'actifs

Il AWS CDK prend en charge les types d'actifs suivants :

Ressources Amazon S3

Il s'agit de fichiers et de répertoires locaux qu'ils AWS CDK téléchargent sur Amazon S3.

Image Docker

Il s'agit d'images Docker qu'ils AWS CDK téléchargent sur Amazon ECR.

Ces types d'actifs sont expliqués dans les sections suivantes.

Ressources Amazon S3

Vous pouvez définir les fichiers et les répertoires locaux en tant qu'actifs, ainsi que les AWS CDK packages et les télécharger sur Amazon S3 via le module aws-s3-assets.

L'exemple suivant définit un actif de répertoire local et un actif de fichier.

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")),
})
```

Dans la plupart des cas, il n'est pas nécessaire d'utiliser directement les API du `aws-s3-assets` module. Les modules qui prennent en charge des actifs, tels que`aws-lambda`, disposent de méthodes pratiques qui vous permettent d'utiliser des actifs. Pour les fonctions Lambda, la méthode statique fromAsset () vous permet de spécifier un répertoire ou un fichier .zip dans le système de fichiers local.

Exemple de fonction Lambda

Un cas d'utilisation courant consiste à créer des fonctions Lambda avec le code du gestionnaire en tant que ressource Amazon S3.

L'exemple suivant utilise une ressource Amazon S3 pour définir un gestionnaire Python dans le répertoire `handler` local. Il crée également une fonction Lambda avec l'actif du répertoire local comme propriété. `code` Voici le code Python du gestionnaire.

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

Le code de l' AWS CDK application principale doit ressembler à ce qui suit.

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
}
```

La `Function` méthode utilise des actifs pour regrouper le contenu du répertoire et l'utiliser pour le code de la fonction.

> **Tip**
>
> Les `.jar` fichiers Java sont des fichiers ZIP dotés d'une extension différente. Ils sont chargés tels quels sur Amazon S3, mais lorsqu'ils sont déployés en tant que fonction Lambda, les fichiers qu'ils contiennent sont extraits, ce que vous ne souhaitez peut-être pas. Pour éviter cela, placez le `.jar` fichier dans un répertoire et spécifiez ce répertoire comme ressource.

Exemple d'attributs relatifs au moment du déploiement

Les types d'actifs Amazon S3 présentent également des attributs de temps de déploiement qui peuvent être référencés dans AWS CDK les bibliothèques et les applications. La commande AWS CDK CLI cdk synth affiche les propriétés des actifs sous forme de AWS CloudFormation paramètres.

L'exemple suivant utilise des attributs de temps de déploiement pour transmettre l'emplacement d'une ressource d'image à une fonction Lambda sous forme de variables d'environnement. (Le type de fichier n'a pas d'importance ; l'image PNG utilisée ici n'est qu'un exemple.)

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(),
  },
})
```

Autorisations

Si vous utilisez des actifs Amazon S3 directement via le module aws-s3-assets, des rôles, des utilisateurs ou des groupes IAM, et que vous devez lire des actifs pendant l'exécution, accordez-leur des autorisations IAM via la méthode Asset.grantRead.

L'exemple suivant accorde à un groupe IAM des autorisations de lecture sur une ressource de fichier.

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)
```

Ressources d'image Docker

AWS CDK Supporte le regroupement d'images Docker locales en tant que ressources via le aws-ecr-assetsmodule.

L'exemple suivant définit une image Docker créée localement et transmise à Amazon ECR.
Les images sont créées à partir d'un répertoire contextuel Docker local (avec un Dockerfile) et téléchargées sur Amazon ECR par la AWS CDK CLI ou le pipeline CI/CD de votre application. Les images peuvent être naturellement référencées dans votre AWS CDK application.

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")),
})
```

Le `my-image` répertoire doit inclure un Dockerfile. La AWS CDK CLI crée une image Docker à partir de`my-image`, la transmet vers un référentiel Amazon ECR et spécifie le nom du référentiel en tant que AWS CloudFormation paramètre de votre pile. Les types de ressources d'image Docker exposent des attributs de temps de déploiement qui peuvent être référencés dans les AWS CDK bibliothèques et les applications. La commande AWS CDK CLI cdk synth affiche les propriétés des actifs sous forme de AWS CloudFormation paramètres.

Exemple de définition de tâche Amazon ECS

Un cas d'utilisation courant consiste à créer un Amazon ECS TaskDefinitionpour exécuter des conteneurs Docker. L'exemple suivant indique l'emplacement d'une ressource d'image Docker créée localement AWS CDK et transmise à 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),
})
```

Exemple d'attributs relatifs au moment du déploiement

L'exemple suivant montre comment utiliser les attributs de temps de déploiement `repository` et comment `imageUri` créer une définition de tâche Amazon ECS avec le type de AWS Fargate lancement. Notez que la recherche du dépôt Amazon ECR nécessite la balise de l'image, et non son URI. Nous l'extrayons donc à la fin de l'URI de la ressource.

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()),
})
```

Exemple d'arguments de construction

Vous pouvez fournir des arguments de construction personnalisés pour l'étape de construction de Docker via l'option de propriété `buildArgs` (Python :`build_args`) lorsque la AWS CDK CLI crée l'image pendant le déploiement.

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"),
  },
})
```

Autorisations

Si vous utilisez un module qui prend en charge les actifs d'image Docker, tel que aws-ecs, il AWS CDK gère les autorisations pour vous lorsque vous utilisez les actifs directement ou via. ContainerImage fromEcrRepository(Python :`from_ecr_repository`). Si vous utilisez directement les ressources d'image Docker, assurez-vous que le principal utilisateur est autorisé à extraire l'image.

Dans la plupart des cas, vous devez utiliser la méthode Asset.repository.GrantPull (Python :. `grant_pull` Cela modifie la politique IAM du principal pour lui permettre d'extraire des images de ce référentiel. Si le principal qui extrait l'image n'appartient pas au même compte, ou s'il s'agit d'un AWS service qui n'assume aucun rôle dans votre compte (par exemple AWS CodeBuild), vous devez accorder des autorisations d'extraction conformément à la politique de ressources et non à la politique du principal. Utilisez le fichier asset.repository. addToResourceMéthode de politique (Python :`add_to_resource_policy`) pour accorder les autorisations principales appropriées.

AWS CloudFormation métadonnées des ressources

> **Note**
>
> Cette section ne concerne que les auteurs de constructions. Dans certaines situations, les outils doivent savoir qu'une certaine ressource CFN utilise un actif local. Par exemple, vous pouvez utiliser la AWS SAM CLI pour appeler des fonctions Lambda localement à des fins de débogage. Consultez the section called “AWS SAM intégration” pour plus de détails.

Pour permettre de tels cas d'utilisation, les outils externes consultent un ensemble d'entrées de métadonnées sur les AWS CloudFormation ressources :

- `aws:asset:path`— Indique le chemin local de la ressource.
- `aws:asset:property`— Le nom de la propriété de ressource dans laquelle l'actif est utilisé.

À l'aide de ces deux entrées de métadonnées, les outils peuvent identifier que les actifs sont utilisés par une certaine ressource et permettre des expériences locales avancées.

Pour ajouter ces entrées de métadonnées à une ressource, utilisez la méthode `asset.addResourceMetadata` (Python :`add_resource_metadata`).

Autorisations

La bibliothèque AWS Construct utilise quelques expressions idiomatiques courantes et largement implémentées pour gérer les accès et les autorisations. Le module IAM fournit les outils dont vous avez besoin pour utiliser ces expressions idiomatiques.

AWS CDK utilise AWS CloudFormation pour déployer les modifications. Chaque déploiement implique un acteur (un développeur ou un système automatisé) qui lance un AWS CloudFormation déploiement. Au cours de cette opération, l'acteur assumera une ou plusieurs identités IAM (utilisateur ou rôles) et transmettra éventuellement un rôle à AWS CloudFormation.

Si vous vous authentifiez AWS IAM Identity Center en tant qu'utilisateur, le fournisseur d'authentification unique fournit des informations d'identification de session de courte durée qui vous autorisent à agir en tant que rôle IAM prédéfini. Pour savoir comment AWS CDK obtenir les AWS informations d'identification grâce à l'authentification IAM Identity Center, voir Comprendre l'authentification IAM Identity Center dans le Guide de référence AWS des SDK et des outils.

Principaux

Un principal IAM est une AWS entité authentifiée représentant un utilisateur, un service ou une application qui peut appeler AWS des API. La bibliothèque AWS Construct permet de spécifier les principes de plusieurs manières flexibles pour leur permettre d'accéder à vos AWS ressources.

Dans les contextes de sécurité, le terme « principal » fait spécifiquement référence aux entités authentifiées telles que les utilisateurs. Les objets tels que les groupes et les rôles ne représentent pas les utilisateurs (ni les autres entités authentifiées) mais les identifient indirectement dans le but d'accorder des autorisations.

Par exemple, si vous créez un groupe IAM, vous pouvez accorder au groupe (et donc à ses membres) un accès en écriture à une table Amazon RDS. Cependant, le groupe lui-même n'est pas un principal car il ne représente pas une entité unique (vous ne pouvez pas non plus vous connecter à un groupe).

Dans la bibliothèque IAM du CDK, les classes qui identifient directement ou indirectement les principaux implémentent l'`IPrincipal`interface, ce qui permet d'utiliser ces objets de manière interchangeable dans les politiques d'accès. Cependant, ils ne sont pas tous des principes essentiels au sens de la sécurité. Ces objets incluent :

1. des ressources IAM telles que `Role`, `User`, et `Group`

2. Principaux de service () `new iam.ServicePrincipal('service.amazonaws.com')`
3. Principaux fédérés () `new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`
4. Principaux du compte (`new iam.AccountPrincipal('0123456789012')`)
5. Principaux utilisateurs canoniques () `new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`
6. AWS Organizations principes () `new iam.OrganizationPrincipal('org-id')`
7. Principaux ARN arbitraires () `new iam.ArnPrincipal(res.arn)`
8. Et de `iam.CompositePrincipal(principal1, principal2, ...)` faire confiance à plusieurs principes

Octrois

Chaque construction qui représente une ressource accessible, telle qu'un compartiment Amazon S3 ou une table Amazon DynamoDB, possède des méthodes qui accordent l'accès à une autre entité. Toutes ces méthodes ont des noms commençant par grant.

Par exemple, les compartiments Amazon S3 disposent des méthodes `grantRead` et `grantReadWrite` (Python :`grant_read`,`grant_read_write`) nécessaires pour permettre l'accès en lecture et en lecture/écriture, respectivement, d'une entité au compartiment. L'entité n'a pas besoin de savoir exactement quelles autorisations IAM Amazon S3 sont requises pour effectuer ces opérations.

Le premier argument d'une méthode d'octroi est toujours de type iGrantable. Cette interface représente les entités auxquelles des autorisations peuvent être accordées. C'est-à-dire qu'il représente des ressources dotées de rôles, tels que les objets IAM `RoleUser`, et`Group`.

D'autres entités peuvent également obtenir des autorisations. Par exemple, plus loin dans cette rubrique, nous expliquons comment accorder à un CodeBuild projet l'accès à un compartiment Amazon S3. Généralement, le rôle associé est obtenu via une `role` propriété de l'entité à laquelle l'accès est accordé.

Les ressources qui utilisent des rôles d'exécution, telles que`lambda.Function`, sont également mises en œuvre`IGrantable`, afin que vous puissiez leur accorder l'accès directement au lieu d'accorder l'accès à leur rôle. Par exemple, s'il s'`bucket`agit d'un compartiment Amazon S3 et `function` d'une fonction Lambda, le code suivant accorde à la fonction un accès en lecture au compartiment.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Parfois, des autorisations doivent être appliquées pendant le déploiement de votre stack. C'est le cas lorsque vous accordez à une ressource AWS CloudFormation personnalisée l'accès à une autre ressource. La ressource personnalisée sera invoquée pendant le déploiement. Elle doit donc disposer des autorisations spécifiées au moment du déploiement.

Un autre cas est celui où un service vérifie que les bonnes politiques sont appliquées au rôle que vous lui transmettez. (Un certain nombre de AWS services le font pour s'assurer que vous n'avez pas oublié de définir les politiques.) Dans ces cas, le déploiement risque d'échouer si les autorisations sont appliquées trop tard.

Pour forcer l'application des autorisations de la subvention avant la création d'une autre ressource, vous pouvez ajouter une dépendance à la subvention elle-même, comme indiqué ici. Bien que la valeur de retour des méthodes de subvention soit généralement ignorée, chaque méthode de subvention renvoie en fait un `iam.Grant` objet.

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);
```

Rôles

Le package IAM contient une `Role` construction qui représente les rôles IAM. Le code suivant crée un nouveau rôle, faisant confiance au service 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
});
```

Vous pouvez ajouter des autorisations à un rôle en appelant la `addToPolicy` méthode du rôle (Python :`add_to_policy`), en transmettant un `PolicyStatement` qui définit la règle à ajouter. L'instruction est ajoutée à la politique par défaut du rôle ; si elle n'en possède aucune, elle est créée.

L'exemple suivant ajoute une déclaration de `Deny` politique au rôle pour les actions `ec2:SomeAction` et `s3:AnotherAction` sur les ressources `bucket` et `otherRole` (Python :`other_role`), à condition que le service autorisé soit 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"
        }
    }
}));
```

Dans l'exemple précédent, nous avons créé une nouvelle entrée `PolicyStatement` en ligne avec l'appel `addToPolicy` (Python :`add_to_policy`). Vous pouvez également transmettre une déclaration de politique existante ou une déclaration que vous avez modifiée. L'PolicyStatementobjet dispose de nombreuses méthodes pour ajouter des principes, des ressources, des conditions et des actions.

Si vous utilisez une construction qui nécessite un rôle pour fonctionner correctement, vous pouvez effectuer l'une des opérations suivantes :

- Transmettez un rôle existant lors de l'instanciation de l'objet de construction.
- Laissez le concept créer un nouveau rôle pour vous, en faisant confiance au directeur de service approprié. L'exemple suivant utilise une telle construction : un CodeBuild projet.

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
});
```

Une fois l'objet créé, le rôle (qu'il s'agisse du rôle transmis ou du rôle par défaut créé par la construction) est disponible en tant que propriété`role`. Toutefois, cette propriété n'est pas disponible sur les ressources externes. Par conséquent, ces constructions ont une méthode `addToRolePolicy` (Python :`add_to_role_policy`).

La méthode ne fait rien si la construction est une ressource externe, sinon elle appelle la méthode `addToPolicy` (Python :`add_to_policy`) de la `role` propriété. Cela vous évite d'avoir à traiter explicitement le cas non défini.

L'exemple suivant illustre ce qui suit :

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
}));
```

Politiques basées sur une ressource

Certaines ressources AWS, telles que les compartiments Amazon S3 et les rôles IAM, sont également soumises à une politique de ressources. Ces constructions ont une `addToResourcePolicy` méthode (Python :`add_to_resource_policy`), qui prend a `PolicyStatement` comme argument. Chaque déclaration de politique ajoutée à une politique de ressources doit spécifier au moins un principal.

Dans l'exemple suivant, le compartiment Amazon S3 `bucket` octroie un rôle doté de cette `s3:SomeAction` autorisation.

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 }
}));
```

Utilisation d'objets IAM externes

Si vous avez défini un utilisateur, un principal, un groupe ou un rôle IAM en dehors de votre AWS CDK application, vous pouvez utiliser cet objet IAM dans votre AWS CDK application. Pour ce faire, créez une référence à celui-ci en utilisant son ARN ou son nom. (Utilisez le nom pour les utilisateurs, les groupes et les rôles.) La référence renvoyée peut ensuite être utilisée pour accorder des autorisations ou pour élaborer des déclarations de politique, comme expliqué précédemment.

- Pour les utilisateurs, appelez `User.fromUserArn()` ou`User.fromUserName()`. `User.fromUserAttributes()`est également disponible, mais fournit actuellement les mêmes fonctionnalités que`User.fromUserArn()`.
- Pour les principes, instanciez un objet. `ArnPrincipal`
- Pour les groupes, appelez `Group.fromGroupArn()` ou`Group.fromGroupName()`.
- Pour les rôles, appelez `Role.fromRoleArn()` ou`Role.fromRoleName()`.

Les politiques (y compris les politiques gérées) peuvent être utilisées de la même manière en utilisant les méthodes suivantes. Vous pouvez utiliser des références à ces objets partout où une politique IAM est requise.

- `Policy.fromPolicyName`
- `ManagedPolicy.fromManagedPolicyArn`

- `ManagedPolicy.fromManagedPolicyName`
- `ManagedPolicy.fromAwsManagedPolicyName`

> **Note**
>
> Comme pour toutes les références à AWS des ressources externes, vous ne pouvez pas modifier les objets IAM externes dans votre application CDK.

Contexte d'exécution

Les valeurs de contexte sont des paires clé-valeur qui peuvent être associées à une application, une pile ou une construction. Ils peuvent être fournis à votre application à partir d'un fichier (généralement dans le répertoire de votre projet `cdk.json` ou `cdk.context.json` dans le répertoire de votre projet) ou sur la ligne de commande.

Le kit d'outils CDK utilise le contexte pour mettre en cache les valeurs extraites de votre AWS compte lors de la synthèse. Les valeurs incluent les zones de disponibilité de votre compte ou les identifiants Amazon Machine Image (AMI) actuellement disponibles pour les instances Amazon EC2. Comme ces valeurs sont fournies par votre AWS compte, elles peuvent changer entre les exécutions de votre application CDK. Cela en fait une source potentielle de changements involontaires. Le comportement de mise en cache du kit CDK « bloque » ces valeurs pour votre application CDK jusqu'à ce que vous décidiez d'accepter les nouvelles valeurs.

Imaginez le scénario suivant sans mise en cache du contexte. Supposons que vous ayez spécifié la « dernière version d'Amazon Linux » comme AMI pour vos instances Amazon EC2 et qu'une nouvelle version de cette AMI ait été publiée. Ensuite, la prochaine fois que vous déploierez votre pile CDK, vos instances déjà déployées utiliseront l'ancienne (« mauvaise ») AMI et devront être mises à niveau. La mise à niveau entraînerait le remplacement de toutes vos instances existantes par de nouvelles, ce qui serait probablement inattendu et indésirable.

Le CDK enregistre plutôt les AMI disponibles pour votre compte dans le `cdk.context.json` fichier de votre projet et utilise la valeur stockée pour les futures opérations de synthèse. Ainsi, la liste des AMI n'est plus une source potentielle de changement. Vous pouvez également être sûr que vos piles seront toujours synthétisées selon les mêmes AWS CloudFormation modèles.

Les valeurs de contexte ne sont pas toutes des valeurs mises en cache dans votre AWS environnement. the section called “Drapeaux caractéristiques”sont également des valeurs

contextuelles. Vous pouvez également créer vos propres valeurs de contexte à utiliser par vos applications ou constructions.

Les clés de contexte sont des chaînes. Les valeurs peuvent être de n'importe quel type pris en charge par JSON : nombres, chaînes, tableaux ou objets.

> **Tip**
>
> Si vos constructions créent leurs propres valeurs de contexte, intégrez le nom du package de votre bibliothèque dans ses clés afin qu'il n'entre pas en conflit avec les valeurs de contexte des autres packages.

De nombreuses valeurs de contexte sont associées à un AWS environnement particulier, et une application CDK donnée peut être déployée dans plusieurs environnements. La clé de ces valeurs inclut le AWS compte et la région afin que les valeurs issues de différents environnements n'entrent pas en conflit.

La clé de contexte suivante illustre le format utilisé par le AWS CDK, y compris le compte et la région.

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

> **Important**
>
> Les valeurs de contexte mises en cache sont gérées par le AWS CDK et ses constructions, y compris les constructions que vous pouvez écrire. N'ajoutez ni ne modifiez les valeurs de contexte mises en cache en modifiant manuellement les fichiers. Il peut toutefois être utile de vérifier de `cdk.context.json` temps en temps quelles valeurs sont mises en cache. Les valeurs de contexte qui ne représentent pas les valeurs mises en cache doivent être stockées sous la `context` clé `decdk.json`. De cette façon, elles ne seront pas effacées lorsque les valeurs mises en cache seront effacées.

Sources des valeurs contextuelles

Les valeurs de contexte peuvent être fournies à votre AWS CDK application de six manières différentes :

- Automatiquement depuis le AWS compte courant.

- Grâce à l'--contextoption de cdk commande. (Ces valeurs sont toujours des chaînes.)
- Dans le `cdk.context.json` dossier du projet.
- Dans la `context` clé du `cdk.json` fichier du projet.
- Dans la `context` clé de votre `~/.cdk.json` dossier.
- Dans votre AWS CDK application en utilisant la `construct.node.setContext()` méthode.

Le fichier de projet `cdk.context.json` est l'endroit où sont mises en AWS CDK cache les valeurs de contexte extraites de votre AWS compte. Cette pratique permet d'éviter des modifications inattendues de vos déploiements lorsque, par exemple, une nouvelle zone de disponibilité est introduite. Le AWS CDK n'écrit de données contextuelles dans aucun des autres fichiers répertoriés.

> **Important**
>
> Parce qu'ils font partie de l'état de votre application `cdk.json` et `cdk.context.json` doivent être affectés au contrôle de source avec le reste du code source de votre application. Sinon, les déploiements dans d'autres environnements (par exemple, un pipeline CI) risquent de produire des résultats incohérents.

Les valeurs de contexte sont limitées à la structure qui les a créées ; elles sont visibles pour les structures enfants, mais pas pour les parents ou les frères et sœurs. Les valeurs de contexte définies par le AWS CDK Toolkit (la cdk commande) peuvent être définies automatiquement, à partir d'un fichier ou à partir de l'--contextoption. Les valeurs de contexte issues de ces sources sont définies implicitement dans la App construction. Ils sont donc visibles pour toutes les constructions de chaque pile de l'application.

Votre application peut lire une valeur de contexte à l'aide de `construct.node.tryGetContext` cette méthode. Si l'entrée demandée ne se trouve pas dans le build actuel ou dans l'un de ses parents, le résultat est`undefined`. (Le résultat peut également être l'équivalent de votre langage, comme `None` en Python.)

Méthodes de contexte

Il AWS CDK prend en charge plusieurs méthodes contextuelles qui permettent aux AWS CDK applications d'obtenir des informations contextuelles à partir de l' AWS environnement. Par exemple, vous pouvez obtenir une liste des zones de disponibilité disponibles dans un AWS compte et une région donnés à l'aide de la méthode Stack.AvailabilityZones.

Les méthodes contextuelles sont les suivantes :

HostedZone. Depuis Lookup

Récupère les zones hébergées dans votre compte.

Zones de disponibilité Stack.

Obtient les zones de disponibilité prises en charge.

StringParameter.valueFromLookup

Obtient une valeur depuis le magasin de paramètres Amazon EC2 Systems Manager de la région actuelle.

VPC. From Lookup

Permet d'intégrer les Amazon Virtual Private Clouds existants à vos comptes.

LookupMachineImage

Recherche une image de machine à utiliser avec une instance NAT dans un Amazon Virtual Private Cloud.

Si aucune valeur de contexte requise n'est disponible, l' AWS CDK application indique au kit d'outils CDK que les informations de contexte sont manquantes. Ensuite, la CLI interroge le AWS compte courant pour obtenir les informations et stocke les informations contextuelles obtenues dans le `cdk.context.json` fichier. Ensuite, il exécute à nouveau l' AWS CDK application avec les valeurs de contexte.

Affichage et gestion du contexte

Utilisez la cdk context commande pour afficher et gérer les informations de votre `cdk.context.json` fichier. Pour voir ces informations, utilisez la cdk context commande sans aucune option. Le résultat devrait ressembler à ce qui suit.

```
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.
```

Pour supprimer une valeur de contextecdk context --reset, exécutez en spécifiant la clé ou le numéro correspondant à la valeur. L'exemple suivant supprime la valeur correspondant à la deuxième clé de l'exemple précédent. Cette valeur représente la liste des zones de disponibilité de la région Europe (Irlande).

```
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.
```

Par conséquent, si vous souhaitez effectuer une mise à jour vers la dernière version de l'AMI Amazon Linux, utilisez l'exemple précédent pour effectuer une mise à jour contrôlée de la valeur de contexte et la réinitialiser. Ensuite, synthétisez et déployez à nouveau votre application.

```
cdk synth
```

Pour effacer toutes les valeurs de contexte stockées pour votre applicationcdk context --clear, exécutez comme suit.

```
cdk context --clear
```

Seules les valeurs de contexte stockées `cdk.context.json` peuvent être réinitialisées ou effacées. Ne touche AWS CDK pas les autres valeurs de contexte. Par conséquent, pour empêcher la réinitialisation d'une valeur de contexte à l'aide de ces commandes, vous pouvez copier la valeur dans`cdk.json`.

AWS CDK**`--context`**Drapeau Toolkit

Utilisez l'option `--context` (en abrégé) `-c` pour transmettre les valeurs de contexte d'exécution à votre application CDK lors de la synthèse ou du déploiement.

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

Pour spécifier plusieurs valeurs de contexte, répétez l'--contextoption autant de fois que vous le souhaitez, en fournissant une paire clé-valeur à chaque fois.

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

Lors de la synthèse de plusieurs piles, les valeurs de contexte spécifiées sont transmises à toutes les piles. Pour fournir différentes valeurs de contexte à des piles individuelles, utilisez des touches différentes pour les valeurs ou utilisez plusieurs cdk synth cdk deploy commandes.

Les valeurs de contexte transmises depuis la ligne de commande sont toujours des chaînes. Si une valeur est généralement d'un autre type, votre code doit être prêt à la convertir ou à l'analyser. Des valeurs de contexte autres que des chaînes peuvent être fournies d'une autre manière (par exemple, dans`cdk.context.json`). Pour vous assurer que ce type de valeur fonctionne comme prévu, vérifiez qu'il s'agit d'une chaîne avant de la convertir.

Exemple

Voici un exemple d'utilisation d'un Amazon VPC existant à l'aide AWS CDK du contexte.

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()
        });
    }
}
```

Vous pouvez utiliser cdk diff pour voir les effets de la transmission d'une valeur de contexte sur la ligne de commande :

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

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

Les valeurs de contexte résultantes peuvent être visualisées comme indiqué ici.

```
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"
    ]
  }
}
```

Drapeaux caractéristiques

AWS CDK Utilise des indicateurs de fonctionnalité pour activer des comportements potentiellement perturbateurs dans une version. Les drapeaux sont stockés sous forme de the section called “Contexte” valeurs dans `cdk.json` (ou`~/.cdk.json`). Ils ne sont pas supprimés par les cdk context --clear commandes cdk context --reset or.

Les indicateurs de fonctionnalité sont désactivés par défaut. Les projets existants qui ne spécifient pas le drapeau continueront de fonctionner comme avant dans les AWS CDK versions ultérieures. cdk initLes nouveaux projets créés à l'aide d'indicateurs d'inclusion activant toutes les fonctionnalités

disponibles dans la version qui a créé le projet. Modifiez `cdk.json` pour désactiver tous les indicateurs pour lesquels vous préférez le comportement antérieur. Vous pouvez également ajouter des indicateurs pour activer de nouveaux comportements après la mise à niveau du AWS CDK.

Une liste de tous les indicateurs de fonctionnalités actuels se trouve dans le AWS CDK GitHub référentiel dans `FEATURE_FLAGS.md`. Consultez le `CHANGELOG` dans une version donnée pour une description de toutes les nouvelles fonctionnalités ajoutées dans cette version.

Revenir au comportement de la version 1

Dans CDK v2, les valeurs par défaut de certains indicateurs de fonctionnalité ont été modifiées par rapport à la v1. Vous pouvez les redéfinir sur pour revenir `false` à un comportement spécifique de la AWS CDK v1. Utilisez la `cdk diff` commande pour inspecter les modifications apportées à votre modèle synthétisé afin de déterminer si l'un de ces indicateurs est nécessaire.

`@aws-cdk/core:newStyleStackSynthesis`

Utilisez la nouvelle méthode de synthèse des piles, qui suppose des ressources bootstrap portant des noms connus. Nécessite un amorçage moderne, mais permet à son tour le CI/CD via CDK Pipelines et les déploiements entre comptes prêts à l'emploi.

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

Si votre application utilise plusieurs clés d'API Amazon API Gateway et les associe à des plans d'utilisation.

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

Si votre application utilise une instance de base de données Amazon RDS ou des clusters de base de données et qu'elle spécifie explicitement l'identifiant de ces derniers.

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

Si votre application utilise la politique de sécurité TLS_V1_2_2019 avec les distributions. Amazon CloudFront Le CDK v2 utilise la politique de sécurité TLSv1.2_2021 par défaut.

`@aws-cdk/core:stackRelativeExports`

Si votre application utilise plusieurs piles et que vous faites référence aux ressources d'une pile à l'autre, cela détermine si le chemin absolu ou relatif est utilisé pour créer AWS CloudFormation les exportations.

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

S'il est défini sur`false`, le CDK inclut des métadonnées lorsqu'il détecte si une fonction Lambda a changé. Cela peut entraîner des échecs de déploiement lorsque seules les métadonnées ont changé, car les versions dupliquées ne sont pas autorisées. Il n'est pas nécessaire d'annuler cet indicateur si vous avez apporté au moins une modification à toutes les fonctions Lambda de votre application.

La syntaxe permettant de rétablir ces drapeaux `cdk.json` est indiquée ici.

```
{
  "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
  }
}
```

Aspects

Les aspects permettent d'appliquer une opération à toutes les constructions d'une portée donnée. L'aspect peut modifier les constructions, par exemple en ajoutant des balises. Il peut également vérifier l'état des constructions, par exemple s'assurer que tous les compartiments sont chiffrés.

Pour appliquer un aspect à une construction et à toutes les constructions de la même portée, appelez `Aspects.of(SCOPE).add()`avec un nouvel aspect, comme indiqué dans l'exemple suivant.

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(...))
```

Il AWS CDK utilise des aspects pour étiqueter les ressources, mais le framework peut également être utilisé à d'autres fins. Par exemple, vous pouvez l'utiliser pour valider ou modifier les AWS CloudFormation ressources définies pour vous par des constructions de niveau supérieur.

Aspects en détail

Les aspects utilisent le modèle des visiteurs. Un aspect est une classe qui implémente l'interface suivante.

TypeScript

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

JavaScript

JavaScript n'a pas d'interface en tant que fonctionnalité de langage. Par conséquent, un aspect est simplement une instance d'une classe ayant une `visit` méthode qui accepte le nœud à opérer.

Python

Python n'a pas d'interface en tant que fonctionnalité de langage. Par conséquent, un aspect est simplement une instance d'une classe ayant une `visit` méthode qui accepte le nœud à opérer.

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)
}
```

Lorsque vous appelez`Aspects.of(`*SCOPE*`).add(...)`, la construction ajoute l'aspect à une liste interne d'aspects. Vous pouvez obtenir la liste avec`Aspects.of(`*SCOPE*`)`.

Pendant la phase de préparation, il AWS CDK appelle la `visit` méthode de l'objet pour la construction et chacun de ses enfants dans l'ordre du haut vers le bas.

La `visit` méthode est libre de changer quoi que ce soit dans la construction. Dans les langages fortement typés, convertissez la construction reçue en un type plus spécifique avant d'accéder aux propriétés ou méthodes spécifiques à la construction.

Les aspects ne se propagent pas au-delà des limites de la `Stage` construction, car ils `Stages` sont autonomes et immuables une fois définis. Appliquez des aspects à la `Stage` construction elle-même (ou à une valeur inférieure) si vous souhaitez qu'ils visitent les constructions situées à l'intérieur du`Stage`.

Exemple

L'exemple suivant confirme que le versionnement est activé pour tous les compartiments créés dans la pile. L'aspect ajoute une annotation d'erreur aux constructions qui échouent à la validation. Cela entraîne l'échec de l'synthopération et empêche le déploiement de l'assemblage cloud obtenu.

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());
```

Commencer à utiliser le AWS CDK

Commencez avec le AWS Cloud Development Kit (AWS CDK) en installant AWS CDK CLI et en créant votre première application CDK.

Rubriques

Prérequis

Ressources recommandées

Avant de commencer à utiliser le AWS CDK, nous vous recommandons de bien comprendre les éléments suivants :

- Une introduction au AWS CDK. Pour en savoir plus, consultez Qu'est-ce que c'est AWS CDK ?
- Concepts fondamentaux qui sous-tendent le AWS CDK. Pour en savoir plus, veuillez consulter la section AWS CDK concepts.
- Le Services AWS que vous souhaitez gérer avec le AWS CDK.
- AWS Identity and Access Management. Pour plus d'informations, voir Qu'est-ce que l'IAM ? et qu'est-ce que IAM Identity Center ?
- AWS CloudFormation car il AWS CDK utilise le AWS CloudFormation service pour fournir des ressources créées dans le CDK. Pour en savoir plus, consultez Qu'est-ce qu' AWS CloudFormation ?
- Le langage de programmation pris en charge que vous prévoyez d'utiliser avec AWS CDK.

Préparez votre environnement local

Tous les AWS CDK développeurs, quelle que soit votre langue préférée, ont besoin de la version Node.js14.15.0 ou ultérieure. Tous les langages de programmation pris en charge utilisent le même backend, qui s'exécute surNode.js. Nous recommandons une version avec support actif à long terme. Il se peut que votre organisation ait une recommandation différente.

> **Important**
>
> Les versions 13.0.0 à 13.6.0 de Node.js ne sont pas compatibles avec le AWS CDK en raison de problèmes de compatibilité liés à ses dépendances.

Les autres prérequis dépendent de la langue dans laquelle vous développez des AWS CDK applications et sont les suivants.

TypeScript

- TypeScript 3.8 ou version ultérieure (`npm -g install typescript`)

JavaScript

Aucune exigence supplémentaire

Python

- Python 3.7 ou version ultérieure, y compris `pip` et `virtualenv`

Java

- Kit de développement Java (JDK) 8 (alias 1.8) ou version ultérieure
- Apache Maven 3.5 ou version ultérieure

L'IDE Java est recommandé (nous utilisons Eclipse dans certains exemples de ce guide). L'IDE doit être capable d'importer des projets Maven. Vérifiez que votre projet est configuré pour utiliser Java 1.8. Définissez la variable d'environnement JAVA_HOME sur le chemin où vous avez installé le JDK.

C#

.NET Core 3.1 ou version ultérieure, ou .NET 6.0 ou version ultérieure.

Visual Studio 2019 (n'importe quelle édition) ou Visual Studio Code sont recommandés.

Go

Go 1.1.8 ou version ultérieure.

Pour des informations plus détaillées, consultez la section Conditions requises pour votre langue :

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

> **Obsolète d'un langage tiers**
>
> Chaque version linguistique n'est prise en charge que jusqu'à ce qu'elle le soit EOL (fin de vie) et peut être modifiée avec préavis.

Étape 1 : Création d'un Compte AWS

Si vous êtes nouveau AWS, vous devez vous inscrire à un Compte AWS et créer un utilisateur administratif. Pour plus d'informations, consultez la section Configuration avec IAM dans le Guide de l'utilisateur d'IAM.

Lorsque vous interagissez avec AWS, vous spécifiez vos informations de AWS sécurité pour vérifier qui vous êtes et si vous êtes autorisé à accéder aux ressources que vous demandez. AWS utilise les informations d'identification de sécurité pour authentifier et autoriser vos demandes. Pour en savoir plus, consultez les informations d'identification AWS de sécurité dans le guide de l'utilisateur IAM.

Étape 2 : Configuration de l'accès par programmation

Lorsque vous développez avec le AWS CDK dans votre environnement local, vous compterez sur le AWS CDK CLI pour interagir avec vos AWS ressources Services AWS et les gérer. Pour utiliser le AWS CDK CLI, vous devez configurer l'accès par programmation. Pour en savoir plus sur les différentes manières de configurer l'accès par programmation, consultez Authentification et accès dans le Guide de référence AWS des SDK et des outils.

Pour les nouveaux utilisateurs qui n'ont pas reçu de méthode d'authentification de la part de leur employeur, nous vous recommandons d'utiliser AWS IAM Identity Center. Cette méthode consiste à installer le AWS Command Line Interface (AWS CLI) et à l'utiliser pour la configuration et la connexion au portail AWS d'accès. Pour configurer l'accès par programmation à l'aide d'IAM Identity

Center, consultez la section Authentification IAM Identity Center dans le Guide de référence AWS des SDK et des outils. Une fois terminé, votre environnement doit contenir les éléments suivants :

- Le AWS CLI, que vous utilisez pour démarrer une session de portail d' AWS accès avant d'exécuter votre application.
- AWS`config`Fichier partagé comportant un `[default]` profil avec un ensemble de valeurs de configuration pouvant être référencées à partir du AWS CDK. Pour connaître l'emplacement de ce fichier, consultez Location of the shared files dans le manuel AWS SDKs and Tools Reference Guide.
- Le `config` fichier partagé définit le `region`paramètre. Cela définit les AWS CDK utilisations par défaut Région AWS pour les AWS demandes.
- AWS CDK Utilise la configuration du fournisseur de jetons SSO du profil pour obtenir des informations d'identification avant d'envoyer des demandes à AWS. La `sso_role_name` valeur, qui est un rôle IAM connecté à un ensemble d'autorisations IAM Identity Center, doit autoriser l'accès à l'utilisateur dans Services AWS votre application.

  Le `config` fichier d'exemple suivant montre un profil par défaut configuré avec la configuration du fournisseur de jetons SSO. Le `sso_session` paramètre du profil fait référence à la `sso-session`section nommée. La `sso-session` section contient les paramètres permettant de lancer une session sur le portail AWS d'accès.

```
[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
```

Démarrer une session sur le portail AWS d'accès

Avant d'y accéder Services AWS, vous devez disposer d'une session de portail AWS d'accès active AWS CDK pour utiliser l'authentification IAM Identity Center afin de résoudre les informations d'identification. En fonction de la durée de votre session configurée, votre accès finira par expirer et

une erreur d'authentification AWS CDK se produira. Exécutez la commande suivante dans le AWS CLI pour vous connecter au portail AWS d'accès.

```
aws sso login
```

Si la configuration de votre fournisseur de jetons SSO utilise un profil nommé au lieu du profil par défaut, la commande estaws sso login --profile *NAME*. Spécifiez également ce profil lors de l'émission de cdk commandes à l'aide de l'--profileoption ou de la variable d'AWS_PROFILEenvironnement.

Pour vérifier si vous avez déjà une session active, exécutez la AWS CLI commande suivante.

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

La réponse à cette commande doit indiquer le compte IAM Identity Center et l'ensemble d'autorisations configurés dans le fichier partagé config.

> **Note**
>
> Si vous disposez déjà d'une session active sur le portail AWS d'accès et que vous l'exécutezaws sso login, il ne vous sera pas demandé de fournir d'informations d'identification.
> Le processus de connexion peut vous demander d'autoriser l' AWS CLI accès à vos données. Étant donné que le AWS CLI est construit au-dessus du SDK pour Python, les messages d'autorisation peuvent contenir des variantes du botocore nom.

Étape 3 : installez le AWS CDKCLI

Installez le AWS CDK CLI globalement à l'aide de la commande Node Package Manager suivante.

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

> **Note**
>
> Si une erreur d'autorisation s'affiche et que vous disposez d'un accès administrateur sur votre système, essayezsudo npm install -g aws-cdk.

Exécutez la commande suivante pour vérifier la réussite de l'installation. Le numéro de version AWS CDK CLI doit être affiché :

```
cdk --version
```

Si vous recevez un message d'erreur, essayez de le désinstaller AWS CDK CLI en exécutant ce qui suit :

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

Répétez ensuite les étapes pour réinstaller le AWS CDK CLI.

Si le message d'erreur persiste, supprimez le `node-modules` dossier du projet en cours ainsi que du `node-modules` dossier global. Pour localiser ce dossier, exécuteznpm `config get prefix`.

Vous AWS CDK CLI obtiendrez les informations d'identification de sécurité à partir des sources que vous avez configurées lors des étapes précédentes.

> **Note**
>
> Le CDK Toolkit v2 fonctionne avec les projets CDK v1 existants. Cependant, il ne peut pas initialiser de nouveaux projets CDK v1. Voyez the section called “Nouveaux prérequis” si vous devez être capable de le faire.

Étape 4 : Bootstrap votre environnement

Chaque AWS environnement dans lequel vous prévoyez de déployer des ressources doit être amorcé.

Pour démarrer, exécutez ce qui suit :

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

> **Tip**
>
> Si vous n'avez pas votre numéro de AWS compte à portée de main, vous pouvez l'obtenir auprès du AWS Management Console. Ou, si vous l'avez AWS CLI installé, la commande suivante affiche les informations de votre compte par défaut, y compris le numéro de compte.

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

Si vous avez créé des profils nommés dans votre AWS configuration locale, vous pouvez utiliser `--profile` cette option pour afficher les informations de compte pour un profil spécifique. L'exemple suivant montre comment afficher les informations de compte pour le profil de production.

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

Pour afficher la région par défaut, utilisez`aws configure get`.

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

AWS CDK Outils optionnels

Il s'AWS Toolkit for Visual Studio Codeagit d'un plug-in open source pour Visual Studio Code qui vous permet de créer, de déboguer et de déployer des applications sur AWS. La boîte à outils fournit une expérience intégrée pour le développement AWS CDK d'applications. Il inclut la fonctionnalité AWS CDK Explorer pour répertorier vos AWS CDK projets et parcourir les différents composants de l'application CDK. Installez le plug-in et apprenez-en plus sur l'utilisation de l' AWS CDK explorateur.

Étapes suivantes

Maintenant que vous l'avez installé AWS CDK CLI, utilisez-le pour créer votre première AWS CDK application.

Pour en savoir plus sur l'utilisation du AWS CDK dans votre langage de programmation préféré, consultezUtilisation des langages AWS CDK de programmation pris en charge.

AWS CDK Il s'agit d'un projet open source. Pour contribuer, voir Contribuer au AWS Cloud Development Kit (AWS CDK).

En savoir plus

Pour en savoir plus à ce sujet AWS CDK, consultez les rubriques suivantes :

- Atelier CDK — Atelier pratique approfondi.
- Référence d'API — Explorez les constructions disponibles pour celles Services AWS que vous utiliserez.
- Construct Hub — Trouvez des constructions de la communauté CDK.
- AWS CDK exemples — Explorez des exemples de code de AWS CDK projets.

Votre première AWS CDK application

Commencez à utiliser le en AWS Cloud Development Kit (AWS CDK) créant votre première application CDK.

Avant de commencer ce didacticiel, nous vous recommandons de suivre les étapes suivantes :

- Voir Qu'est-ce que c'est AWS CDK ? pour une introduction au AWS CDK.
- Consultez AWS CDK concepts pour découvrir les concepts de base du AWS CDK.
- Passez en revue les prérequis et les étapes AWS CDK de configuration surCommencer à utiliser le AWS CDK.

Rubriques

À propos de ce didacticiel

Dans ce didacticiel, vous allez créer et déployer une AWS CDK application simple. Cette application contient une pile avec une seule ressource de bucket Amazon Simple Storage Service (Amazon S3). Grâce à ce didacticiel, vous allez apprendre ce qui suit :

- Structure d'un AWS CDK projet.
- Comment créer une AWS CDK application.
- Comment utiliser la bibliothèque AWS Construct pour définir des applications, des piles et AWS des ressources.
- Comment utiliser le CDK CLI pour synthétiser, différencier, déployer et supprimer votre application CDK.
- Comment modifier et redéployer votre application CDK pour mettre à jour vos ressources déployées.

Le flux AWS CDK de développement standard comprend les étapes suivantes :

1. Créez votre AWS CDK application — Ici, vous utiliserez un modèle fourni par le AWS CDK CLI.
2. Définissez vos piles et vos ressources : utilisez des constructions pour définir vos piles et vos AWS ressources au sein de votre application.
3. Créez votre application : cette étape est facultative. Exécute AWS CDK CLI automatiquement cette étape si nécessaire. Il est recommandé d'effectuer cette étape pour identifier les erreurs de syntaxe et de type.
4. Synthétisez vos piles : cette étape crée un AWS CloudFormation modèle pour chaque pile de votre application. Cette étape est utile pour identifier les erreurs logiques dans les AWS ressources que vous avez définies.
5. Déployez votre application : déployez-la AWS dans votre environnement en utilisant AWS CloudFormation pour provisionner vos ressources. Au cours du déploiement, vous identifierez tout problème d'autorisation lié à votre application.

Dans le cadre d'un flux de travail classique, vous allez revenir en arrière et répéter les étapes précédentes pour modifier ou déboguer votre application.

Nous vous recommandons d'utiliser le contrôle de version pour vos AWS CDK projets.

Étape 1 : créer l'application

Une application CDK doit se trouver dans son propre répertoire, avec ses propres dépendances de module locales. Sur votre machine de développement, créez un nouveau répertoire. Voici un exemple de création d'un nouveau `hello-cdk` répertoire :

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

> **Important**
>
> Assurez-vous de nommer le répertoire de votre projet`hello-cdk`, exactement comme indiqué ici. Le modèle de AWS CDK projet utilise le nom du répertoire pour nommer les éléments du code généré. Si vous utilisez un autre nom, le code de ce didacticiel ne fonctionnera pas.

Ensuite, depuis votre nouveau répertoire, initialisez l'application à l'aide de la cdk init commande. Spécifiez le `app` modèle et votre langage de programmation préféré à l'aide de l'`--language`option. Voici un exemple :

TypeScript

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

JavaScript

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

Python

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

Une fois l'application créée, entrez également les deux commandes suivantes. Ils activent l'environnement virtuel Python de l'application et installent les dépendances AWS CDK principales.

```
source .venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

Java

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

Si vous utilisez un IDE, vous pouvez désormais ouvrir ou importer le projet. Dans Eclipse, par exemple, choisissez Fichier > Importer > Maven > Projets Maven existants. Assurez-vous que les paramètres du projet sont définis pour utiliser Java 8 (1.8).

C#

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

Si vous utilisez Visual Studio, ouvrez le fichier de solution dans le `src` répertoire.

Go

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

Une fois l'application créée, entrez également la commande suivante pour installer les modules AWS Construct Library dont l'application a besoin.

```
go get
```

La cdk init commande crée un certain nombre de fichiers et de dossiers dans le `hello-cdk` répertoire pour vous aider à organiser le code source de votre AWS CDK application. Collectivement, c'est ce que l'on appelle votre AWS CDK projet. Prenez un moment pour découvrir le projet CDK.

Si vous l'avez Git installé, chaque projet que vous créez à l'aide cdk init est également initialisé en tant que Git référentiel.

Étape 2 : créer l'application

Dans la plupart des environnements de programmation, vous créez ou compilez du code après avoir apporté des modifications. Cela n'est pas nécessaire AWS CDK puisque le CDK CLI exécutera automatiquement cette étape. Cependant, vous pouvez toujours créer manuellement lorsque vous souhaitez détecter des erreurs de syntaxe et de type. Voici un exemple :

TypeScript

```
npm run build
```

JavaScript

Aucune étape de construction n'est nécessaire.

Python

Aucune étape de construction n'est nécessaire.

Java

```
mvn compile -q
```

Ou appuyez sur Ctrl-B dans Eclipse (les autres IDE Java peuvent varier)

C#

```
dotnet build src
```

Ou appuyez sur F6 dans Visual Studio

Go

```
go build
```

Étape 3 : Répertoriez les piles dans l'application

Vérifiez que votre application a été correctement créée en répertoriant les piles qu'elle contient. Exécutez les commandes suivantes :

```
cdk ls
```

La sortie doit s'afficher`HelloCdkStack`. Si ce résultat ne s'affiche pas, vérifiez que vous vous trouvez dans le bon répertoire de travail de votre projet et réessayez. Si vous ne voyez toujours pas votre pile, répétez l'opération the section called “Étape 1 : créer l'application” et réessayez.

Étape 4 : ajouter un compartiment Amazon S3

À ce stade, votre application CDK contient une seule pile. Vous allez ensuite définir une ressource de bucket Amazon Simple Storage Service (Amazon S3) au sein de votre stack. Pour ce faire, vous allez importer et utiliser la construction `Bucket` L2 depuis la bibliothèque de AWS constructions.

Modifiez votre application CDK en important la `Bucket` construction et en définissant la ressource de votre compartiment Amazon S3. Voici un exemple :

TypeScript

Dans `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

Dans `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

Dans `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

Dans `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#

Dans `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

Dans `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
}
```

Regardons la `Bucket` construction de plus près. Comme toutes les constructions, la `Bucket` classe prend trois paramètres :

- scope — Définit la `Stack` classe en tant que parent de la `Bucket` construction. Toutes les constructions qui définissent les AWS ressources sont créées dans le cadre d'une pile. Vous pouvez définir des constructions à l'intérieur des constructions, en créant une hiérarchie (arbre). Ici, et dans la plupart des cas, le champ d'application est `this` (`self`finPython).
- Id : ID logique du contenu `Bucket` dans votre AWS CDK application. Cet identifiant, associé à un hachage basé sur l'emplacement du bucket dans la pile, identifie de manière unique le bucket lors du déploiement. AWS CDK Il fait également référence à cet ID lorsque vous mettez à jour la structure dans votre application et que vous la redéployez pour mettre à jour la ressource déployée. Ici, votre identifiant logique est`MyFirstBucket`. Les compartiments peuvent également avoir un nom, spécifié avec la `bucketName` propriété. Ceci est différent de l'identifiant logique.
- props — Ensemble de valeurs qui définissent les propriétés du compartiment. Ici, vous avez défini la `versioned` propriété comme`true`, qui permet le versionnement des fichiers du bucket.

Les accessoires sont représentés différemment dans les langues prises en charge par le AWS CDK.

- Dans TypeScript et JavaScript, `props` est un argument unique et vous transmettez un objet contenant les propriétés souhaitées.
- En Python, les accessoires sont transmis sous forme d'arguments de mots clés.
- En Java, un générateur est fourni pour transmettre les accessoires. Il y en a deux : une pour`BucketProps`, et une seconde `Bucket` pour vous permettre de construire la construction et ses accessoires en une seule étape. Ce code utilise ce dernier.
- En C#, vous instanciez un `BucketProps` objet à l'aide d'un initialiseur d'objet et vous le transmettez comme troisième paramètre.

Si les accessoires d'une construction sont facultatifs, vous pouvez omettre complètement le `props` paramètre.

Toutes les constructions utilisent ces trois mêmes arguments, il est donc facile de rester orienté lorsque vous en découvrez de nouveaux. Et comme vous pouvez vous y attendre, vous pouvez sous-classer n'importe quelle construction pour l'étendre en fonction de vos besoins, ou si vous souhaitez modifier ses valeurs par défaut.

Étape 5 : Synthétiser un modèle AWS CloudFormation

Synthétisez un AWS CloudFormation modèle pour l'application, comme suit :

```
cdk synth
```

Si votre application contient plusieurs piles, vous devez spécifier les piles à synthétiser. Comme votre application ne contient qu'une seule pile, le CDK détecte CLI automatiquement la pile à synthétiser.

Si vous ne l'exécutez pas`cdk synth`, le CDK CLI exécutera automatiquement cette étape lors du déploiement. Nous vous recommandons toutefois d'exécuter cette étape avant chaque déploiement.

> **Tip**
>
> Si vous recevez un message d'erreur tel que`--app is required ...`, vérifiez le répertoire à partir duquel vous exécutez CLI les commandes CDK. Vous devriez être dans le répertoire principal de votre application.

La `cdk synth` commande exécute votre application. Cela crée un AWS CloudFormation modèle pour chaque pile de votre application. Le CDK CLI affichera une version au format YAML de votre modèle sur la ligne de commande et enregistrera une version au format JSON de votre modèle dans le répertoire. `cdk.out` Voici un extrait de la sortie de ligne de commande qui montre le bucket défini dans le AWS CloudFormation modèle :

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

> **Note**
>
> Chaque modèle généré contient une `AWS::CDK::Metadata` ressource par défaut. L' AWS CDK équipe utilise ces métadonnées pour mieux comprendre AWS CDK l'utilisation et trouver des moyens de l'améliorer. Pour plus de détails, notamment pour savoir comment désactiver les rapports de version, consultezRapport sur les versions.

Le modèle généré peut être déployé via la AWS CloudFormation console ou tout autre outil AWS CloudFormation de déploiement. Vous pouvez également utiliser le CDK CLI pour le déploiement. À l'étape suivante, vous utiliserez le CDK CLI pour le déploiement.

Étape 6 : Déployez votre stack

Pour déployer votre pile CDK à AWS CloudFormation l'aide du CDKCLI, exécutez ce qui suit :

```
cdk deploy
```

> **Important**
>
> Vous devez effectuer un démarrage unique de votre AWS environnement avant le déploiement. Pour obtenir des instructions, consultez Bootstrap your environment.

De mêmecdk synth, il n'est pas nécessaire de spécifier la AWS CDK pile puisque l'application ne contient qu'une seule pile.

Si votre code a des implications en termes de sécurité, le CDK en CLI produira un résumé. Vous devrez les confirmer pour poursuivre le déploiement. L'application présentée dans ce didacticiel n'a pas ces implications.

Après l'exécutioncdk deploy, le CDK CLI affiche les informations de progression au fur et à mesure du déploiement de votre stack. Lorsque vous avez terminé, vous pouvez accéder à la AWS CloudFormation console pour voir votre HelloCdkStack pile. Vous pouvez également accéder à la console Amazon S3 pour consulter vos MyFirstBucket ressources.

Félicitations ! Vous avez déployé votre première pile à l'aide du AWS CDK. Vous allez ensuite modifier votre application et la redéployer pour mettre à jour votre ressource.

Étape 7 : modifiez votre application

Au cours de cette étape, vous allez modifier votre compartiment Amazon S3 en le configurant pour qu'il soit automatiquement supprimé lorsque votre pile est supprimée. Cette modification implique de modifier les RemovalPolicy propriétés du bucket. Vous allez également configurer la autoDeleteObjects propriété pour configurer le CDK CLI afin de supprimer des objets du compartiment avant de le détruire. Par défaut, AWS CloudFormation ne supprime pas les compartiments Amazon S3 contenant des objets.

Utilisez l'exemple suivant pour modifier votre ressource :

TypeScript

Mettre à jour lib/hello-cdk-stack.ts.

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

JavaScript

Mettre à jour lib/hello-cdk-stack.js.

```
new s3.Bucket(this, 'MyFirstBucket', {
```

```
  versioned: true,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  autoDeleteObjects: true
});
```

Python

Mettre à jour `hello_cdk/hello_cdk_stack.py`.

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

Java

Mettre à jour `src/main/java/com/myorg/HelloCdkStack.java`.

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

C#

Mettre à jour `src/HelloCdk/HelloCdkStack.cs`.

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

Go

Mettre à jour `hello-cdk.go`.

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

```
  })
```

À l'heure actuelle, vos modifications de code n'ont entraîné aucune mise à jour directe de la ressource de compartiment Amazon S3 déployée. Votre code définit l'état souhaité pour votre ressource. Pour modifier votre ressource déployée, vous allez utiliser le CDK CLI pour synthétiser l'état souhaité dans un nouveau AWS CloudFormation modèle. Vous déploierez ensuite votre nouveau AWS CloudFormation modèle sous forme d'ensemble de modifications. Les ensembles de modifications apportent uniquement les modifications nécessaires pour atteindre le nouvel état souhaité.

Pour voir ces modifications, utilisez la `cdk diff` commande. Exécutez les commandes suivantes :

```
cdk diff
```

Le CDK CLI interroge votre Compte AWS compte pour obtenir le dernier AWS CloudFormation modèle de la `HelloCdkStack` pile. Ensuite, il compare le dernier modèle avec le modèle qu'il vient de synthétiser à partir de votre application. Le résultat doit ressembler à ce qui suit.

```
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
```

Ce diff comporte quatre sections :

- Modifications de la déclaration IAM et modifications de la politique IAM — Ces modifications d'autorisation sont apportées parce que vous définissez la `AutoDeleteObjects` propriété dans votre compartiment Amazon S3. La fonction de suppression automatique utilise une ressource personnalisée pour supprimer les objets du compartiment avant que le compartiment lui-même ne soit supprimé. Les objets IAM accordent au code de la ressource personnalisée l'accès au bucket.
- Paramètres — AWS CDK Utilise ces entrées pour localiser l'actif AWS Lambda fonctionnel de la ressource personnalisée.
- Ressources : ressources nouvelles et modifiées de cette pile. Nous pouvons voir les objets IAM mentionnés précédemment, la ressource personnalisée et sa fonction Lambda associée être ajoutés. Nous pouvons également constater que le bucket `DeletionPolicy` et les `UpdateReplacePolicy` attributs sont en cours de mise à jour. Ils permettent de supprimer le bucket en même temps que la pile et de le remplacer par un nouveau.

Vous remarquerez peut-être que nous l'avons spécifié `RemovalPolicy` dans notre AWS CDK application mais que nous avons obtenu une `DeletionPolicy` propriété dans le AWS CloudFormation modèle obtenu. Cela est dû au fait AWS CDK que le nom de la propriété est différent. La AWS CDK valeur par défaut est de conserver le compartiment lorsque la pile est supprimée, tandis que la AWS CloudFormation valeur par défaut est de le supprimer. Pour plus d'informations, consultez the section called “Politiques de suppression”.

Pour voir votre nouveau AWS CloudFormation modèle, vous pouvez exécutercdk synth. En apportant quelques modifications à votre application CDK, votre nouveau AWS CloudFormation modèle inclut désormais de nombreuses lignes de code supplémentaires par rapport au AWS CloudFormation modèle d'origine.

Déployez ensuite votre application en exécutant ce qui suit :

```
cdk deploy
```

Ils vous AWS CDK informeront des modifications de politique de sécurité que nous avons déjà constatées dans le diff. Entrez y pour approuver les modifications et déployer la pile mise à jour.

Le CDK CLI déploiera votre stack pour apporter les modifications souhaitées. Voici un exemple de résultat :

```
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
```

Étape 8 : Détruire les ressources de l'application

Maintenant que vous avez terminé ce didacticiel, vous pouvez supprimer la AWS CloudFormation pile déployée et toutes les ressources qui y sont associées. Il s'agit d'une bonne pratique pour minimiser les coûts inutiles et préserver la propreté de votre environnement. Exécutez les commandes suivantes :

```
cdk destroy
```

Entrez y pour approuver les modifications et supprimer votre pile.

> **Note**
>
> Si vous ne modifiiez pas celui du compartiment`RemovalPolicy`, la suppression de la pile se terminerait avec succès, mais le compartiment deviendrait orphelin (il ne serait plus associé à la pile).

Étapes suivantes

Félicitations ! Vous avez terminé ce didacticiel et vous avez utilisé le AWS CDK pour créer, modifier et supprimer avec succès des ressources dans le AWS Cloud. Vous êtes maintenant prêt à commencer à utiliser le AWS CDK.

Pour en savoir plus sur l'utilisation du AWS CDK dans votre langage de programmation préféré, consultezUtilisation des langages AWS CDK de programmation pris en charge.

Pour des ressources supplémentaires, consultez les rubriques suivantes :

- Essayez le CDK Workshop pour une visite plus approfondie impliquant un projet plus complexe.
- Approfondissez des concepts tels que the section called “Environnements” the section called “Assets”the section called “Autorisations”,the section called “Contexte”,,the section called “Paramètres”, etthe section called “Personnalisation des constructions”.
- Consultez la référence de l'API pour commencer à explorer les constructions CDK disponibles pour vos services préférés AWS .
- Visitez Construct Hub pour découvrir les constructions créées par AWS et d'autres.
- Découvrez des exemples d'utilisation du AWS CDK.

AWS CDK Il s'agit d'un projet open source. Pour contribuer, voir Contribuer au AWS Cloud Development Kit (AWS CDK).

Migration de la AWS CDK v1 vers la v2 AWS CDK

La version 2 AWS Cloud Development Kit (AWS CDK) est conçue pour faciliter l'écriture de l'infrastructure sous forme de code dans votre langage de programmation préféré. Cette rubrique décrit les modifications entre la v1 et la v2 du AWS CDK.

> **Tip**
>
> Pour identifier les piles déployées avec la version AWS CDK 1, utilisez l'utilitaire awscdk-v1-stack-finder.

Les principaux changements entre AWS CDK v1 et CDK v2 sont les suivants.

- AWS CDK v2 consolide les parties stables de la bibliothèque AWS Construct, y compris la bibliothèque principale, dans un seul package,`aws-cdk-lib`. Les développeurs n'ont plus besoin d'installer de packages supplémentaires pour les différents AWS services qu'ils utilisent. Cette approche à package unique signifie également que vous n'avez pas à synchroniser les versions des différents packages de bibliothèques CDK.

  Les constructions L1 (CFNxxxx), qui représentent les ressources exactes disponibles dans AWS CloudFormation, sont toujours considérées comme stables et sont donc incluses. `aws-cdk-lib`
- Les modules expérimentaux, dans lesquels nous travaillons toujours avec la communauté pour développer de nouvelles constructions L2 ou L3, ne sont pas inclus. `aws-cdk-lib` Ils sont plutôt distribués sous forme de packages individuels. Les packages expérimentaux sont nommés avec un `alpha` suffixe et un numéro de version sémantique. Le numéro de version sémantique correspond à la première version de la bibliothèque AWS Construct avec laquelle ils sont compatibles, également avec un `alpha` suffixe. Les constructions sont déplacées `aws-cdk-lib` après avoir été désignées stables, ce qui permet à la bibliothèque de constructions principale de respecter un versionnage sémantique strict.

  La stabilité est spécifiée au niveau du service. Par exemple, si nous commençons à créer une ou plusieurs constructions L2 pour Amazon AppFlow, qui, au moment d'écrire ces lignes, ne comporte que des constructions L1, elles apparaissent d'abord dans un module nommé. `@aws-cdk/aws-appflow-alpha` Ensuite, ils passent au `aws-cdk-lib` moment où nous pensons que les nouvelles constructions répondent aux besoins fondamentaux des clients.

Une fois qu'un module a été désigné comme stable et intégréaws-cdk-lib, de nouvelles API sont ajoutées en utilisant la convention « beTAN » décrite dans le bullet suivant.

Une nouvelle version de chaque module expérimental est publiée à chaque sortie du AWS CDK. Cependant, dans la plupart des cas, ils n'ont pas besoin d'être synchronisés. Vous pouvez mettre à jour aws-cdk-lib le module expérimental quand vous le souhaitez. L'exception est que lorsque deux ou plusieurs modules expérimentaux connexes dépendent les uns des autres, ils doivent être de la même version.

- Pour les modules stables auxquels de nouvelles fonctionnalités sont ajoutées, les nouvelles API (qu'il s'agisse de constructions entièrement nouvelles ou de nouvelles méthodes ou propriétés sur une construction existante) reçoivent un Beta1 suffixe pendant que le travail est en cours. (Suivi par Beta2Beta3, et ainsi de suite lorsque des modifications importantes sont nécessaires.) Une version de l'API sans suffixe est ajoutée lorsque l'API est désignée stable. Toutes les méthodes sauf la plus récente (qu'elle soit bêta ou finale) sont alors déconseillées.

  Par exemple, si nous ajoutons une nouvelle méthode grantPower() à une construction, elle apparaît initialement sous la formegrantPowerBeta1(). Si des modifications importantes sont nécessaires (par exemple, un nouveau paramètre ou une nouvelle propriété obligatoire), la version suivante de la méthode sera nomméegrantPowerBeta2(), et ainsi de suite. Lorsque le travail est terminé et que l'API est finalisée, la méthode grantPower() (sans suffixe) est ajoutée et les méthodes betAN sont déconseillées.

  Toutes les API bêta restent dans la bibliothèque Construct jusqu'à la prochaine version majeure (3.0), et leurs signatures ne changeront pas. Vous verrez des avertissements d'obsolescence si vous les utilisez. Vous devez donc passer à la version finale de l'API dès que possible. Cependant, aucune future version AWS CDK 2.x n'interrompra votre application.
- La Construct classe a été extraite de la bibliothèque AWS CDK dans une bibliothèque séparée, avec les types associés. Ceci est fait pour soutenir les efforts visant à appliquer le modèle de programmation Construct à d'autres domaines. Si vous écrivez vos propres constructions ou utilisez des API associées, vous devez déclarer le constructs module en tant que dépendance et apporter des modifications mineures à vos importations. Si vous utilisez des fonctionnalités avancées, telles que l'intégration au cycle de vie de l'application CDK, d'autres modifications peuvent être nécessaires. Pour plus de détails, consultez la RFC.
- Les propriétés, méthodes et types obsolètes de la AWS CDK version v1.x et de sa bibliothèque de construction ont été complètement supprimés de l'API CDK v2. Dans la plupart des langues prises en charge, ces API génèrent des avertissements sous la version v1.x. Vous avez donc peut-être

déjà migré vers les API de remplacement. Une liste complète des API obsolètes dans CDK v1.x est disponible sur. GitHub

- Le comportement qui était contrôlé par des indicateurs de fonctionnalité dans la AWS CDK version v1.x est activé par défaut dans CDK v2. Les indicateurs de fonctionnalité antérieurs ne sont plus nécessaires et, dans la plupart des cas, ils ne sont pas pris en charge. Quelques-uns sont encore disponibles pour vous permettre de revenir au comportement de CDK v1 dans des circonstances très spécifiques. Pour plus d'informations, consultez the section called "Mise à jour des indicateurs de fonctionnalités".
- Avec CDK v2, les environnements dans lesquels vous déployez doivent être amorcés à l'aide de la pile bootstrap moderne. L'ancienne pile bootstrap (par défaut sous v1) n'est plus prise en charge. Le CDK v2 nécessite en outre une nouvelle version de la pile moderne. Pour mettre à niveau vos environnements existants, redémarrez-les. Il n'est plus nécessaire de définir des indicateurs de fonctionnalité ou des variables d'environnement pour utiliser la pile bootstrap moderne.

> **Important**
>
> Le modèle bootstrap moderne accorde efficacement les autorisations implicites `--cloudformation-execution-policies` à n'importe quel AWS compte de la `--trust` liste. Par défaut, cela étend les autorisations de lecture et d'écriture à n'importe quelle ressource du compte bootstrap. Assurez-vous de configurer la pile d'amorçage avec des politiques et des comptes fiables avec lesquels vous êtes à l'aise.

Nouveaux prérequis

La plupart des exigences pour la AWS CDK v2 sont les mêmes que pour la AWS CDK v1.x. Les exigences supplémentaires sont répertoriées ici.

- Pour TypeScript les développeurs, la version TypeScript 3.8 ou ultérieure est requise.
- Une nouvelle version du kit d'outils CDK est requise pour être utilisée avec CDK v2. Maintenant que le CDK v2 est généralement disponible, la version v2 est la version par défaut lors de l'installation du CDK Toolkit. Il est rétrocompatible avec les projets CDK v1, vous n'avez donc pas besoin de conserver la version précédente installée, sauf si vous souhaitez créer des projets CDK v1. Pour effectuer une mise à niveau, problèmen`pm install -g aws-cdk`.

Mise à niveau depuis AWS CDK v2 Developer Preview

Si vous utilisez le CDK v2 Developer Preview, votre projet dépend d'une version Release Candidate du AWS CDK, telle que`2.0.0-rc1`. Mettez-les à jour`2.0.0`, puis mettez à jour les modules installés dans votre projet.

TypeScript

`npm install` ou `yarn install`

JavaScript

`npm install` ou `yarn install`

Python

```
python -m pip install -r requirements.txt
```

Java

```
mvn package
```

C#

```
dotnet restore
```

Go

```
go get
```

Après avoir mis à jour vos dépendances, lancez la mise `npm update -g aws-cdk` à jour du kit d'outils CDK vers la version finale.

Migration de la AWS CDK v1 vers le CDK v2

Pour migrer votre application vers la AWS CDK version v2, mettez d'abord à jour les indicateurs de fonctionnalité dans`cdk.json`. Mettez ensuite à jour les dépendances et les importations de votre application selon les besoins du langage de programmation dans lequel elle est écrite.

Mise à jour vers une version v1 récente

Nous constatons qu'un certain nombre de clients passent d'une ancienne version de AWS CDK v1 à la version la plus récente de v2 en une seule étape. Bien que cela soit certainement possible, vous devrez à la fois effectuer une mise à niveau après plusieurs années de changements (qui, malheureusement, n'ont peut-être pas tous fait l'objet du même nombre de tests d'évolution que ceux d'aujourd'hui) et effectuer une mise à niveau entre des versions avec de nouveaux paramètres par défaut et une organisation du code différente.

Pour une expérience de mise à niveau la plus sûre et pour diagnostiquer plus facilement les sources de modifications inattendues, nous vous recommandons de séparer ces deux étapes : passez d'abord à la dernière version v1, puis passez ensuite à la version v2.

Mise à jour des indicateurs de fonctionnalités

Supprimez les indicateurs de fonctionnalité v1 suivants `cdk.json` s'ils existent, car ils sont tous actifs par défaut dans la AWS CDK version v2. Si leur effet antérieur est important pour votre infrastructure, vous devrez apporter des modifications au code source. Consultez la liste des drapeaux GitHub pour plus d'informations.

- `@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`

Quelques indicateurs de fonctionnalités de la version 1 peuvent être définis pour `false` revenir à des comportements spécifiques de la version AWS CDK 1 ; voir the section called “Revenir au comportement de la version 1” la liste ci-dessous GitHub pour une référence complète.

Pour les deux types d'indicateurs, utilisez la `cdk diff` commande pour inspecter les modifications apportées à votre modèle synthétisé afin de voir si les modifications apportées à l'un de ces indicateurs affectent votre infrastructure.

Compatibilité avec le kit CDK

Le CDK v2 nécessite la version v2 ou ultérieure du kit d'outils CDK. Cette version est rétrocompatible avec les applications CDK v1. Par conséquent, vous pouvez utiliser une seule version installée dans le monde entier de CDK Toolkit avec tous vos AWS CDK projets, qu'ils utilisent la v1 ou la v2. Une exception est que CDK Toolkit v2 ne crée que des projets CDK v2.

Si vous devez créer des projets CDK v1 et v2, n'installez pas CDK Toolkit v2 globalement. (Supprimez-le si vous l'avez déjà installé :`npm remove -g aws-cdk`.) Pour appeler le kit d'outils CDK, utilisez la version v1 ou v2 du kit d'outils CDK comme npx vous le souhaitez.

```
npx aws-cdk@1.x init app --language typescript
npx aws-cdk@2.x init app --language typescript
```

> **Tip**
>
> Configurez des alias de ligne de commande afin de pouvoir utiliser les cdk1 commandes cdk et pour appeler la version souhaitée du kit d'outils 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 $*
> ```

Mettre à jour les dépendances et les importations

Mettez à jour les dépendances de votre application, puis installez les nouveaux packages. Enfin, mettez à jour les importations dans votre code.

TypeScript

Applications

Pour les applications CDK, procédez à la mise à jour `package.json` comme suit. Supprimez les dépendances sur les modules stables individuels de style v1 et établissez la version la plus basse dont `aws-cdk-lib` vous avez besoin pour votre application (2.0.0 ici).

Les constructions expérimentales sont fournies dans des packages séparés, versionnés indépendamment, dont les noms se terminent par `alpha` et un numéro de version alpha. Le numéro de version alpha correspond à la première version `aws-cdk-lib` avec laquelle ils sont compatibles. Ici, nous nous sommes concentrés sur la version `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"
  }
}
```

Construire des bibliothèques

Pour les bibliothèques de construction, définissez la version la plus basse dont `aws-cdk-lib` vous avez besoin pour votre application (2.0.0 ici) et mettez-la à jour `package.json` comme suit.

Notez que cela `aws-cdk-lib` apparaît à la fois comme une dépendance entre pairs et comme une dépendance de développement.

```
{
  "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**
>
> Vous devez effectuer une augmentation de version majeure sur le numéro de version de votre bibliothèque lorsque vous publiez une bibliothèque compatible avec la version 2, car

il s'agit d'un changement radical pour les utilisateurs de bibliothèques. Il n'est pas possible de prendre en charge à la fois le CDK v1 et le CDK v2 avec une seule bibliothèque. Pour continuer à soutenir les clients qui utilisent toujours la version 1, vous pouvez maintenir la version précédente en parallèle ou créer un nouveau package pour la version 2.
C'est à vous de décider combien de temps vous souhaitez continuer à soutenir les clients de la version AWS CDK 1. Vous pourriez vous inspirer du cycle de vie du CDK v1 lui-même, qui est entré en maintenance le 1er juin 2022 et arrivera à expiration end-of-life le 1er juin 2023. Pour plus de détails, consultez la politique de AWS CDK maintenance.

Bibliothèques et applications

Installez les nouvelles dépendances en exécutant `npm install` ou`yarn install`.

Modifiez vos importations pour importer `Construct` depuis le nouveau `constructs` module, les types de base tels que `App` et `Stack` depuis le niveau supérieur de`aws-cdk-lib`, et les modules stables de Construct Library pour les services que vous utilisez à partir des espaces de noms situés sous`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

Mettez à jour `package.json` comme suit. Supprimez les dépendances sur les modules stables individuels de style v1 et établissez la version la plus basse dont `aws-cdk-lib` vous avez besoin pour votre application (2.0.0 ici).

Les constructions expérimentales sont fournies dans des packages séparés, versionnés indépendamment, dont les noms se terminent par `alpha` et un numéro de version alpha. Le numéro de version alpha correspond à la première version `aws-cdk-lib` avec laquelle ils sont compatibles. Ici, nous nous sommes concentrés sur la version `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"
  }
}
```

Installez les nouvelles dépendances en exécutant `npm install` ou`yarn install`.

Modifiez les importations de votre application pour effectuer les opérations suivantes :

- Importer `Construct` depuis le nouveau `constructs` module
- Importez des types de base, tels que `App` et`Stack`, depuis le niveau supérieur de `aws-cdk-lib`
- Importez les modules de la bibliothèque AWS Construct à partir des espaces de noms sous `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

Mettez à jour `requirements.txt` la `install_requires` définition `setup.py` comme suit. Supprimez les dépendances sur les modules stables individuels de style v1.

Les constructions expérimentales sont fournies dans des packages séparés, versionnés indépendamment, dont les noms se terminent par `alpha` et un numéro de version alpha. Le numéro de version alpha correspond à la première version `aws-cdk-lib` avec laquelle ils sont compatibles. Ici, nous avons épinglé la `aws-codestar` v2.0.0alpha1.

```
install_requires=[
     "aws-cdk-lib>=2.0.0",
     "constructs>=10.0.0",
     "aws-cdk.aws-codestar-alpha>=2.0.0alpha1",
     # ...
],
```

> **Tip**
>
> Désinstallez toutes les autres versions des AWS CDK modules déjà installés dans l'environnement virtuel de votre application à l'aide de`pip uninstall`. Installez ensuite les nouvelles dépendances avec`python -m pip install -r requirements.txt`.

Modifiez les importations de votre application pour effectuer les opérations suivantes :

- Importer `Construct` depuis le nouveau `constructs` module
- Importez des types de base, tels que App et`Stack`, depuis le niveau supérieur de `aws_cdk`
- Importez les modules de la bibliothèque AWS Construct à partir des espaces de noms sous `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

Dans`pom.xml`, supprimez toutes les `software.amazon.awscdk` dépendances des modules stables et remplacez-les par des dépendances sur `software.constructs` (for`Construct`) et`software.amazon.awscdk`.

Les constructions expérimentales sont fournies dans des packages séparés, versionnés indépendamment, dont les noms se terminent par `alpha` et un numéro de version alpha. Le numéro de version alpha correspond à la première version `aws-cdk-lib` avec laquelle ils sont compatibles. Ici, nous nous sommes concentrés sur la version `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>
```

Installez les nouvelles dépendances en exécutant`mvn package`.

Modifiez votre code pour effectuer les opérations suivantes :

- Importer `Construct` depuis la nouvelle `software.constructs` bibliothèque
- Importez des classes de base, telles que `Stack` etApp, depuis `software.amazon.awscdk`
- Importer des constructions de service depuis `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#

Le moyen le plus simple de mettre à niveau les dépendances d'une application CDK C# consiste à modifier le `.csproj` fichier manuellement. Supprimez toutes les références aux `Amazon.CDK.*` packages stables et remplacez-les par des références aux `Constructs` packages `Amazon.CDK.Lib` et.

Les constructions expérimentales sont fournies dans des packages séparés, versionnés indépendamment, dont les noms se terminent par `alpha` et un numéro de version alpha. Le

numéro de version alpha correspond à la première version `aws-cdk-lib` avec laquelle ils sont compatibles. Ici, nous nous sommes concentrés sur la version `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" />
```

Installez les nouvelles dépendances en exécutant`dotnet restore`.

Modifiez les importations dans vos fichiers source comme suit.

```
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

Problème go get pour mettre à jour vos dépendances vers la dernière version et mettre à jour le `.mod` fichier de votre projet.

Tester votre application migrée avant de la déployer

Avant de déployer vos piles, utilisez-le `cdk diff` pour vérifier l'absence de modifications inattendues des ressources. Aucune modification des identifiants logiques (entraînant le remplacement des ressources) n'est attendue.

Les changements attendus incluent, sans toutefois s'y limiter :

- Modifications apportées à la `CDKMetadata` ressource.
- Hashs d'actifs mis à jour.
- Changements liés au nouveau style de synthèse des piles. S'applique si votre application utilisait l'ancien synthétiseur Stack dans la version 1. (Le CDK v2 ne prend pas en charge l'ancien synthétiseur Stack.)
- L'ajout d'une `CheckBootstrapVersion` règle.

Les modifications inattendues ne sont généralement pas causées par la mise à niveau vers la AWS CDK version v2 en elle-même. Ils sont généralement le résultat d'un comportement obsolète qui a

été précédemment modifié par des indicateurs de fonctionnalité. Il s'agit d'un symptôme de mise à niveau à partir d'une version du CDK antérieure à environ 1.85.x. Vous constaterez les mêmes modifications lors de la mise à niveau vers la dernière version v1.x. Vous pouvez généralement résoudre ce problème en procédant comme suit :

1. Mettez à jour votre application vers la dernière version v1.x
2. Supprimer les drapeaux de fonctionnalités
3. Révisez votre code si nécessaire
4. Déploiement
5. Passez à la version v2

> **Note**
>
> Si votre application mise à niveau n'est pas déployable après la mise à niveau en deux étapes, signalez le problème.

Lorsque vous êtes prêt à déployer les piles dans votre application, envisagez d'abord d'en déployer une copie afin de pouvoir la tester. Le moyen le plus simple de le faire est de le déployer dans une autre région. Cependant, vous pouvez également modifier les identifiants de vos piles. Après le test, assurez-vous de détruire la copie de test aveccdk destroy.

Résolution des problèmes

TypeScript **`'from' expected`**ou **`';' expected`** erreur lors des importations

Passez à la version TypeScript 3.8 ou ultérieure.

Exécutez « cdk bootstrap »

Si le message d'erreur suivant s'affiche :

```
#  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 v2 nécessite une pile de bootstrap mise à jour, et en outre, tous les déploiements de la v2 nécessitent des ressources de bootstrap. (Avec la version 1, vous pouvez déployer des piles simples sans amorçage.) Consultez the section called “Action d'amorçage” pour plus de détails.

Trouver des piles v1

Lors de la migration de votre application CDK de la v1 à la v2, vous souhaiterez peut-être identifier les AWS CloudFormation piles déployées créées à l'aide de la version v1. Pour ce faire, exécutez la commande suivante :

```
npx awscdk-v1-stack-finder
```

Pour plus de détails sur l'utilisation, consultez le fichier README d'awscdk-v1-stack-finder.

Migrez les ressources et les AWS CloudFormation modèles existants vers le AWS CDK

La fonctionnalité CDK Migrate est en version préliminaire AWS CDK et est sujette à modification.

Utilisez l'interface de ligne de AWS Cloud Development Kit (AWS CDK) commande (AWS CDK CLI) pour migrer les AWS ressources déployées, les AWS CloudFormation piles déployées et les AWS CloudFormation modèles locaux vers AWS CDK.

Rubriques

Comment fonctionne la migration

Utilisez la AWS CDK CLI `cdk migrate` commande pour effectuer une migration depuis les sources suivantes :

- AWS Ressources déployées.
- AWS CloudFormation Stacks déployés.
- AWS CloudFormation Modèles locaux.

AWS Ressources déployées

Vous pouvez migrer AWS des ressources déployées depuis un environnement spécifique (Compte AWS et Région AWS) qui ne sont pas associées à une AWS CloudFormation pile.

Il AWS CDK CLI utilise le service de générateur IaC pour rechercher les ressources de votre AWS environnement afin de recueillir des informations détaillées sur les ressources. Pour en savoir plus sur le générateur IaC, consultez la section Génération de modèles pour les ressources existantes dans le guide de AWS CloudFormation l'utilisateur.

Après avoir rassemblé les détails des ressources, AWS CDK CLI crée une nouvelle application CDK qui inclut une pile unique contenant vos ressources migrées.

AWS CloudFormation Stacks déployés

Vous pouvez migrer une seule AWS CloudFormation pile vers une nouvelle AWS CDK application. Cela AWS CDK CLI récupérera le AWS CloudFormation modèle de votre pile et créera une nouvelle application CDK. L'application CDK sera composée d'une seule pile contenant votre pile migrée. AWS CloudFormation

AWS CloudFormation Modèles locaux

Vous pouvez effectuer la migration à partir d'un AWS CloudFormation modèle local. Les modèles locaux peuvent contenir ou non des ressources déployées. Cela AWS CDK CLI créera une nouvelle application CDK contenant une pile unique contenant vos ressources.

Après la migration, vous pouvez gérer, modifier et déployer votre application CDK pour approvisionner ou mettre AWS CloudFormation à jour vos ressources.

Avantages de CDK Migrate

La migration des ressources vers ce AWS CDK site a toujours été un processus manuel qui nécessite du temps et de l'expertise, voire même AWS CDK pour commencer. AWS CloudFormation Avec CDK Migrate, vous pouvez réaliser la majeure partie des efforts de migration en un rien de temps. AWS CDK CLI CDK Migrate vous permettra de commencer rapidement à utiliser le AWS CDK pour développer et gérer des applications nouvelles et existantes sur AWS.

Considérations

Considérations d’ordre général

CDK Migrate ou CDK Import

La `cdk import` commande peut importer des ressources déployées dans une application CDK nouvelle ou existante. Lors de l'importation, chaque ressource devra être définie manuellement en tant que construction L1 dans votre application. Nous vous recommandons d'`cdk import`utiliser cette option pour importer une ou plusieurs ressources à la fois dans une application CDK nouvelle ou existante. Pour en savoir plus, veuillez consulter la section Importation de ressources existantes dans une pile.

La `cdk migrate` commande migre des ressources déployées, des AWS CloudFormation piles déployées ou des AWS CloudFormation modèles locaux vers une nouvelle application CDK. Pendant la migration, les AWS CDK CLI utilisateurs `cdk import` importent vos ressources dans la nouvelle application CDK. AWS CDK CLIIl génère également des constructions L1 pour chaque ressource pour vous. Nous vous recommandons de l'utiliser `cdk migrate` lors de l'importation depuis une source de migration prise en charge vers une nouvelle AWS CDK application.

CDK Migrate crée uniquement des constructions L1

L'application CDK nouvellement créée inclura uniquement les constructions L1. Vous pouvez ajouter des constructions de niveau supérieur à votre application après la migration.

CDK Migrate crée des applications CDK contenant une seule pile

L'application CDK nouvellement créée contiendra une seule pile.

Lors de la migration des ressources déployées, toutes les ressources migrées seront contenues dans une pile unique dans la nouvelle application CDK.

Lors de la migration de AWS CloudFormation piles, vous ne pouvez migrer qu'une seule AWS CloudFormation pile vers une seule pile dans la nouvelle application CDK.

Migration des actifs

Les actifs du projet, tels que AWS Lambda le code, ne migreront pas directement vers la nouvelle application CDK. Après la migration, vous pouvez spécifier les valeurs des actifs pour les inclure dans l'application CDK.

Migration de ressources dynamiques

Lorsque vous migrez des ressources dynamiques, telles que des bases de données et des compartiments Amazon Simple Storage Service (Amazon S3), vous souhaiterez le plus souvent migrer la ressource existante plutôt que d'en créer une nouvelle.

Pour migrer et préserver les ressources dynamiques, procédez comme suit :

- Vérifiez que votre ressource dynamique prend en charge l'importation. Pour plus d'informations, consultez la section Support des types de ressources dans le Guide de AWS CloudFormation l'utilisateur.
- Après la migration, vérifiez que l'ID logique de la ressource migrée dans la nouvelle application CDK correspond à l'ID logique de la ressource déployée.
- Si vous migrez depuis une AWS CloudFormation pile, vérifiez que le nom de la pile dans la nouvelle application CDK correspond à la AWS CloudFormation pile.
- Déployez l'application CDK à l'aide du même AWS compte et Région AWS de la même ressource migrée.

Considérations relatives à la migration à partir d'un modèle AWS CloudFormation

CDK Migrate prend en charge la migration d'un modèle unique

Lors de la migration AWS CloudFormation de modèles, vous pouvez sélectionner un seul modèle à migrer. Les modèles imbriqués ne sont pas pris en charge.

Migration de modèles dotés de fonctions intrinsèques

Lors de la migration à partir d'un AWS CloudFormation modèle qui utilise des fonctions intrinsèques, ils AWS CDK CLI essaieront de migrer votre logique vers l'application CDK avec la Fn classe. Pour en savoir plus, consultez la classe Fn dans la référence de l'AWS Cloud Development Kit (AWS CDK) API.

Considérations à prendre en compte lors de la migration depuis des ressources déployées

Limites de numérisation

Lorsque vous analysez votre environnement à la recherche de ressources, le générateur iAc est soumis à des limites spécifiques quant aux données qu'il peut récupérer et à des limites de quotas lors de l'analyse. Pour en savoir plus, consultez la section Considérations du guide de AWS CloudFormation l'utilisateur.

Prérequis

Avant d'utiliser la `cdk migrate` commande, procédez comme suit :

1. Établissez l'authentification avec AWS. Pour obtenir des instructions, veuillez consulter Étape 2 : Configuration de l'accès par programmation.
2. Installez ou mettez à niveau le AWS CDK CLI. Pour obtenir des instructions d'installation, consultez Étape 3 : installez le AWS CDKCLI.

Commencez avec CDK Migrate

Pour commencer, exécutez la AWS CDK CLI `cdk migrate` commande depuis le répertoire de votre choix. Fournissez les options obligatoires et facultatives, en fonction du type de migration que vous effectuez.

Pour une liste complète et une description des options que vous pouvez utiliser`cdk migrate`, consultezcdk migrateréférence de commande.

Voici quelques options importantes que vous souhaiterez peut-être proposer.

Nom de la pile

La seule option requise est`--stack-name`. Utilisez cette option pour spécifier le nom de la pile qui sera créée dans l' AWS CDK application après la migration. Le nom de la pile sera également utilisé comme nom de votre AWS CloudFormation pile lors du déploiement.

Langue

`--language`À utiliser pour spécifier le langage de programmation de la nouvelle application CDK.

AWS compte et Région AWS

AWS CDK CLIRécupère le AWS compte et les Région AWS informations à partir de sources par défaut. Pour plus d'informations, consultez Étape 2 : Configuration de l'accès par programmation. Vous pouvez utiliser `--account` des `--region` options `cdk migrate` pour fournir d'autres valeurs.

Répertoire de sortie de votre nouveau projet CDK

Par défaut, un nouveau projet CDK AWS CDK CLI sera créé dans votre répertoire de travail et utilisera la valeur que vous avez fournie `--stack-name` pour nommer le dossier du projet. Si un dossier portant le même nom existe actuellement, il AWS CDK CLI remplacera ce dossier.

Vous pouvez spécifier un chemin de sortie différent pour le nouveau dossier de projet CDK avec l'`--output-path`option.

Source de migration

Fournissez une option pour spécifier la source à partir de laquelle vous effectuez la migration.

- `--from-path`— Migrez à partir d'un AWS CloudFormation modèle local.
- `--from-scan`— Migrez à partir des ressources déployées dans un AWS compte et Région AWS.
- `--from-stack`— Migrez depuis une AWS CloudFormation pile.

En fonction de votre source de migration, vous pouvez proposer des options supplémentaires pour personnaliser la `cdk migrate` commande.

Migrer à partir d'une AWS CloudFormation pile

Pour effectuer une migration à partir d'une AWS CloudFormation pile déployée, offrez l'`--from-stack`option. Indiquez le nom de votre AWS CloudFormation stack déployé avec`--stack-name`. Voici un exemple :

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

Ils AWS CDK CLI effectueront les opérations suivantes :

1. Récupérez le AWS CloudFormation modèle de votre stack déployé.
2. Exécutez `cdk init` pour initialiser une nouvelle application CDK.
3. Créez une pile dans l'application CDK qui contient votre pile migrée. AWS CloudFormation

Lorsque vous migrez depuis une AWS CloudFormation pile déployée, le système AWS CDK CLI tente de faire correspondre les identifiants logiques des ressources déployées et le nom de la AWS CloudFormation pile déployée aux ressources migrées et à la pile dans la nouvelle application CDK.

Après la migration, vous pouvez gérer et modifier votre application CDK normalement. Lorsque vous déployez, AWS CloudFormation identifiera le déploiement comme une mise à jour de AWS CloudFormation pile en raison du nom de AWS CloudFormation pile correspondant. Les ressources dont les identifiants logiques correspondent seront mises à jour. Pour plus d'informations sur le déploiement, consultezGérez et déployez votre application CDK.

Migrer à partir d'un AWS CloudFormation modèle

CDK Migrate prend en charge la migration à partir de AWS CloudFormation modèles formatés en ou. `JSON` `YAML`

Pour effectuer une migration depuis un AWS CloudFormation modèle local, utilisez l'`--from-path`option et indiquez un chemin d'accès au modèle local. Vous devez également fournir l'`--stack-name`option requise. Voici un exemple :

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

Ils AWS CDK CLI effectueront les opérations suivantes :

1. Récupérez votre AWS CloudFormation modèle local.
2. Exécutez `cdk init` pour initialiser une nouvelle application CDK.
3. Créez une pile dans l'application CDK contenant votre modèle migré. AWS CloudFormation

Après la migration, vous pouvez gérer et modifier votre application CDK normalement. Lors du déploiement, vous disposez des options suivantes :

- Mettre à jour une AWS CloudFormation pile : si le AWS CloudFormation modèle local a déjà été déployé, vous pouvez mettre à jour la AWS CloudFormation pile déployée.

- Déployer une nouvelle AWS CloudFormation pile : si le AWS CloudFormation modèle local n'a jamais été déployé, ou si vous souhaitez créer une nouvelle pile à partir d'un modèle précédemment déployé, vous pouvez déployer une nouvelle AWS CloudFormation pile.

Migrer à partir d'un AWS SAM modèle

Pour migrer depuis un modèle AWS Serverless Application Model (AWS SAM), vous devez d'abord le convertir en AWS CloudFormation modèle ou le déployer pour créer une AWS CloudFormation pile.

Pour convertir un AWS SAM modèle en AWS CloudFormation, vous pouvez utiliser la AWS SAM CLI `sam validate --debug` commande. Vous devrez peut-être le `lint` définir `false` dans votre `samconfig.toml` fichier avant d'exécuter cette commande.

Pour le convertir en AWS CloudFormation pile, déployez le AWS SAM modèle à l'aide du AWS SAM CLI. Migrez ensuite depuis la pile déployée.

Migrer à partir des ressources déployées

Pour effectuer une migration à partir de AWS ressources déployées, indiquez l'`--from-scan`option. Vous devez également fournir l'`--stack-name`option requise. Voici un exemple :

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

Ils AWS CDK CLI effectueront les opérations suivantes :

1. Scannez votre compte pour obtenir les détails des ressources et des propriétés — Il AWS CDK CLI utilise le générateur iAc pour scanner votre compte et recueillir des informations.
2. Générer un AWS CloudFormation modèle — Après la numérisation, il AWS CDK CLI utilise le générateur iAc pour créer un AWS CloudFormation modèle.
3. Initialisez une nouvelle application CDK et migrez votre modèle : l'application AWS CDK CLI s'exécute `cdk init` pour initialiser une nouvelle AWS CDK application et migre votre AWS CloudFormation modèle vers l'application CDK en une seule pile.

Utiliser des filtres

Par défaut, il AWS CDK CLI analysera l'ensemble de l' AWS environnement et migrera les ressources jusqu'à la limite de quota maximale du générateur IaC. Vous pouvez fournir des filtres AWS CDK CLI pour spécifier un critère selon lequel les ressources seront migrées de votre compte vers la nouvelle application CDK. Pour en savoir plus, veuillez consulter la section `--filter`.

Recherche de ressources avec le générateur IaC

Selon le nombre de ressources de votre compte, la numérisation peut prendre quelques minutes. Une barre de progression s'affiche pendant le processus de numérisation.

Types de ressources pris en charge

Ils AWS CDK CLI migreront les ressources prises en charge par le générateur iAc. Pour une liste complète, consultez la section Support des types de ressources dans le guide de AWS CloudFormation l'utilisateur.

Résolution des propriétés en écriture seule

Certaines ressources prises en charge contiennent des propriétés en écriture seule. Ces propriétés peuvent être écrites pour configurer la propriété, mais ne peuvent pas être lues par le générateur IaC ou AWS CloudFormation pour obtenir la valeur. Par exemple, une propriété utilisée pour spécifier un mot de passe de base de données peut être en écriture seule pour des raisons de sécurité.

Lors de l'analyse des ressources pendant la migration, le générateur iAC détecte les ressources susceptibles de contenir des propriétés en écriture seule et les classe dans l'un des types suivants :

- `MUTUALLY_EXCLUSIVE_PROPERTIES`— Il s'agit de propriétés en écriture seule pour une ressource spécifique qui sont interchangeables et ont un objectif similaire. L'une des propriétés mutuellement exclusives est requise pour configurer votre ressource. Par exemple, les propriétés `S3BucketImageUri`, et d'une `AWS::Lambda::Function` ressource sont des `ZipFile` propriétés en écriture seule qui s'excluent mutuellement. N'importe lequel d'entre eux peut être utilisé pour spécifier les actifs de votre fonction, mais vous devez en utiliser un.
- `MUTUALLY_EXCLUSIVE_TYPES`— Il s'agit de propriétés en écriture seule obligatoires qui acceptent plusieurs types de configuration. Par exemple, la `Body` propriété d'une `AWS::ApiGateway::RestApi` ressource accepte un type d'objet ou de chaîne.

- UNSUPPORTED_PROPERTIES— Il s'agit de propriétés en écriture seule qui n'entrent pas dans les deux autres catégories. Il s'agit soit de propriétés facultatives, soit de propriétés obligatoires qui acceptent un ensemble d'objets.

Pour plus d'informations sur les propriétés en écriture seule et sur la façon dont le générateur iAC les gère lors de la recherche de ressources déployées et de la création de AWS CloudFormation modèles, consultez les sections Générateur iAc et propriétés en écriture seule dans le Guide de l'utilisateur.AWS CloudFormation

Après la migration, vous devez spécifier des valeurs de propriétés en écriture seule dans la nouvelle application CDK. Une section Warnings AWS CDK CLI sera ajoutée au **ReadMe** fichier du projet CDK pour documenter toutes les propriétés en écriture seule identifiées par le générateur IaC. Voici un exemple :

```
# 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
```

- Les avertissements sont organisés sous des en-têtes qui identifient l'ID logique de la ressource à laquelle ils sont associés.
- Les avertissements sont classés par type. Ces types proviennent directement du générateur IaC.

Pour résoudre les propriétés en écriture seule

1. Identifiez les propriétés en écriture seule à résoudre dans la section Avertissements du fichier de votre projet CDK. `ReadMe` Ici, vous pouvez prendre note des ressources de votre application CDK susceptibles de contenir des propriétés en écriture seule et identifier les types de propriétés en écriture seule découverts.

   a. Pour `MUTUALLY_EXCLUSIVE_PROPERTIES` déterminer la propriété mutuellement exclusive à configurer dans votre AWS CDK application.

   b. Pour`MUTUALLY_EXCLUSIVE_TYPES`, déterminez le type accepté que vous utiliserez pour configurer la propriété.

   c. Pour`UNSUPPORTED_PROPERTIES`, déterminez si la propriété est facultative ou obligatoire. Ensuite, configurez le cas échéant.

2. Utilisez les instructions du générateur IaC et les propriétés en écriture seule pour faire référence à la signification des types d'avertissement.
3. Dans votre application CDK, les valeurs des propriétés en écriture seule à résoudre seront également spécifiées dans la `Props` section de votre application. Indiquez les valeurs correctes ici. Pour obtenir des descriptions et des conseils sur les propriétés, vous pouvez vous référer à la référence de l'AWS CDK API.

   Voici un exemple de `Props` section d'une application CDK migrée avec deux propriétés en écriture seule à résoudre :

```
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;
```

```
}
```

Une fois que vous avez résolu toutes les valeurs de propriété en écriture seule, vous êtes prêt à préparer le déploiement.

Le fichier migrate.json

Il AWS CDK CLI crée un `migrate.json` fichier dans votre AWS CDK projet lors de la migration. Ce fichier contient des informations de référence sur vos ressources déployées. Lorsque vous déployez votre application CDK pour la première fois, elle AWS CDK CLI utilise ce fichier pour référencer vos ressources déployées, associe vos ressources à la nouvelle AWS CloudFormation pile et supprime le fichier.

Gérez et déployez votre application CDK

Lors de la migration vers AWS CDK, la nouvelle application CDK peut ne pas être immédiatement prête à être déployée. Cette rubrique décrit les actions à prendre en compte lors de la gestion et du déploiement de votre nouvelle application CDK.

Préparation au déploiement

Avant le déploiement, vous devez préparer votre application CDK.

Synthétisez votre application

- Utilisez la `cdk synth` commande pour synthétiser la pile de votre application CDK dans un AWS CloudFormation modèle.

  Si vous avez migré à partir d'une AWS CloudFormation pile ou d'un modèle déployé, vous pouvez comparer le modèle synthétisé au modèle migré pour vérifier les valeurs des ressources et des propriétés.

  Pour en savoir plus sur `cdk synth`, consultez Synthétiser des piles.

Effectuez une différence

- Si vous avez migré depuis une AWS CloudFormation pile déployée, vous pouvez utiliser la commande cdk diff pour comparer avec la pile de votre nouvelle application CDK.

Pour en savoir plus sur cdk diff, consultez. Comparaison des piles

Bootstrap votre environnement

Si vous déployez depuis un AWS environnement pour la première fois, utilisez-le `cdk bootstrap` pour préparer votre environnement. Pour en savoir plus, veuillez consulter la section Action d'amorçage.

Déployez votre application CDK

Lorsque vous déployez une application CDK, AWS CDK CLI elle utilise le AWS CloudFormation service pour provisionner vos ressources. Les ressources sont regroupées en une seule pile dans l'application CDK et sont déployées en une seule AWS CloudFormation pile.

Selon l'endroit d'où vous avez migré, vous pouvez effectuer un déploiement pour créer une nouvelle AWS CloudFormation pile ou mettre à jour une AWS CloudFormation pile existante.

Déployer pour créer une nouvelle AWS CloudFormation pile

Si vous avez migré à partir de ressources déployées, une nouvelle AWS CloudFormation pile AWS CDK CLI sera automatiquement créée lors du déploiement. Vos ressources déployées seront incluses dans la nouvelle AWS CloudFormation pile.

Si vous avez migré à partir d'un AWS CloudFormation modèle local qui n'a jamais été déployé, une nouvelle AWS CloudFormation pile AWS CDK CLI sera automatiquement créée lors du déploiement.

Si vous avez migré à partir d'une AWS CloudFormation pile déployée ou d'un AWS CloudFormation modèle local précédemment déployé, vous pouvez effectuer un déploiement pour créer une nouvelle AWS CloudFormation pile. Pour créer une nouvelle pile, procédez comme suit :

- Déployez dans un nouvel AWS environnement. Cela consiste à utiliser un autre AWS compte ou à effectuer un déploiement sur un autre Région AWS.
- Si vous souhaitez déployer une nouvelle pile dans le même AWS environnement que la pile ou le modèle migré, vous devez modifier le nom de la pile dans votre application CDK pour lui attribuer une nouvelle valeur. Vous devez également modifier tous les identifiants logiques des ressources de votre application CDK. Vous pouvez ensuite le déployer dans le même environnement pour créer une nouvelle pile et de nouvelles ressources.

Déployer pour mettre à jour une AWS CloudFormation pile existante

Si vous avez migré à partir d'une AWS CloudFormation pile déployée ou d'un AWS CloudFormation modèle local précédemment déployé, vous pouvez effectuer un déploiement pour mettre à jour la AWS CloudFormation pile existante.

Vérifiez que le nom de la pile dans votre application CDK correspond au nom de la AWS CloudFormation pile déployée et déployez-la dans le même AWS environnement.

Utilisation des langages AWS CDK de programmation pris en charge

Utilisez le AWS Cloud Development Kit (AWS CDK) pour définir votre AWS Cloud infrastructure avec un langage de programmation pris en charge.

Rubriques

Importation de la bibliothèque AWS Construct

AWS CDK Cela inclut la bibliothèque AWS Construct, une collection de constructions organisées par AWS service. Les constructions stables de la bibliothèque sont proposées dans un seul module, appelé par son nom de TypeScript package :`aws-cdk-lib`. Le nom réel du package varie en fonction de la langue.

TypeScript

Installation	`installation de npm aws-cdk-lib`
Importer	`const cdk = require (« aws-cdk-lib ») ;`

JavaScript

Installation	`installation de npm aws-cdk-lib`
Importer	`const cdk = require (« aws-cdk-lib ») ;`

Python

Installation	`installation de python -m pip aws-cdk-lib`
Importer	`importer aws_cdk en tant que cdk`

Java

Ajouter à `pom.xml`	Group `software.amazon.awscdk` ; artifact `aws-cdk-lib`
Importer	`import Software.Amazon.AWSCDK.app ;` (for example)

C#

Installation	`dotnet ajouter le package Amazon.CDK.lib`
Importer	`en utilisant Amazon.CDK ;`

La classe `construct` de base et le code de support se trouvent dans le `constructs` module. Les constructions expérimentales, dont l'API est encore en cours de perfectionnement, sont distribuées sous forme de modules distincts.

La référence de AWS CDK l'API

La référence AWS CDK d'API fournit une documentation détaillée sur les constructions (et autres composants) de la bibliothèque. Une version de la référence d'API est fournie pour chaque langage de programmation pris en charge.

Le matériel de référence de chaque module est divisé dans les sections suivantes.

- Vue d'ensemble : matériel d'introduction que vous devez connaître pour utiliser le service AWS CDK, y compris des concepts et des exemples.
- Constructions : classes de bibliothèque qui représentent une ou plusieurs AWS ressources concrètes. Il s'agit de ressources ou de modèles « sélectionnés » (L2) (ressources L3) qui fournissent une interface de haut niveau avec des valeurs par défaut saines.
- Classes : classes non construites qui fournissent les fonctionnalités utilisées par les constructions du module.
- Structures : structures de données (ensembles d'attributs) qui définissent la structure des valeurs composites telles que les propriétés (`props`argument des constructions) et les options.
- Interfaces : les interfaces, dont les noms commencent tous par « I », définissent les fonctionnalités minimales absolues pour la construction ou autre classe correspondante. Le CDK utilise des interfaces de construction pour représenter les AWS ressources définies en dehors de votre AWS CDK application et référencées par des méthodes telles que`Bucket.fromBucketArn()`.
- Enums : collections de valeurs nommées à utiliser pour spécifier certains paramètres de construction. L'utilisation d'une valeur énumérée permet au CDK de vérifier la validité de ces valeurs lors de la synthèse.
- CloudFormation Ressources : Ces constructions L1, dont les noms commencent par « Cfn », représentent exactement les ressources définies dans la spécification. CloudFormation Ils sont automatiquement générés à partir de cette spécification à chaque version du CDK. Chaque construction L2 ou L3 encapsule une ou plusieurs ressources. CloudFormation
- CloudFormation Types de propriétés : ensemble de valeurs nommées qui définissent les propriétés de chaque CloudFormation ressource.

Interfaces comparées aux classes de construction

Il AWS CDK utilise les interfaces d'une manière spécifique qui peut ne pas être évidente même si vous connaissez les interfaces en tant que concept de programmation.

Les AWS CDK supports utilisent des ressources définies en dehors des applications CDK à l'aide de méthodes telles que`Bucket.fromBucketArn()`. Les ressources externes ne peuvent pas être modifiées et peuvent ne pas disposer de toutes les fonctionnalités disponibles avec les ressources définies dans votre application CDK à l'aide, par exemple, de la `Bucket` classe. Les interfaces représentent donc le strict minimum de fonctionnalités disponibles dans le CDK pour un type de AWS ressource donné, y compris les ressources externes.

Lorsque vous instanciez des ressources dans votre application CDK, vous devez toujours utiliser des classes concrètes telles que. `Bucket` Lorsque vous spécifiez le type d'argument que vous acceptez dans l'une de vos propres constructions, utilisez le type d'interface, par exemple `IBucket` si vous êtes prêt à gérer des ressources externes (c'est-à-dire que vous n'avez pas besoin de les modifier). Si vous avez besoin d'une construction définie par le CDK, spécifiez le type le plus général que vous pouvez utiliser.

Certaines interfaces sont des versions minimales de propriétés ou des ensembles d'options associés à des classes spécifiques, plutôt que des constructions. De telles interfaces peuvent être utiles lors du sous-classement pour accepter des arguments que vous transmettrez à votre classe parent. Si vous avez besoin d'une ou de plusieurs propriétés supplémentaires, vous devez implémenter ou dériver de cette interface, ou d'un type plus spécifique.

> **Note**
>
> Certains langages de programmation pris en charge par le AWS CDK n'ont pas de fonctionnalité d'interface. Dans ces langages, les interfaces ne sont que des classes ordinaires. Vous pouvez les identifier par leurs noms, qui suivent le schéma d'un « I » initial suivi du nom d'une autre construction (par exemple`IBucket`). Les mêmes règles s'appliquent.

Gestion des dépendances

Les dépendances de votre AWS CDK application ou de votre bibliothèque sont gérées à l'aide d'outils de gestion de packages. Ces outils sont couramment utilisés avec les langages de programmation.

Généralement, il AWS CDK prend en charge l'outil de gestion de paquets standard ou officiel du langage, s'il en existe un. Sinon, ils AWS CDK prendront en charge la langue la plus populaire ou la plus largement prise en charge. Vous pouvez également utiliser d'autres outils, en particulier s'ils

fonctionnent avec les outils pris en charge. Cependant, le support officiel pour les autres outils est limité.

AWS CDK prend en charge les gestionnaires de packages suivants :

Langue	Outil de gestion de packages pris en charge
TypeScript/JavaScript	NPM (Node Package Manager) ou Yarn
Python	PIP (Installateur de packages pour Python)
Java	Maven
C#	NuGet
Go	Modules Go

Lorsque vous créez un nouveau projet à l'aide de la AWS CDK CLI `cdk init` commande, les dépendances des bibliothèques principales du CDK et des constructions stables sont automatiquement spécifiées.

Pour plus d'informations sur la gestion des dépendances pour les langages de programmation pris en charge, consultez les rubriques suivantes :

- Gestion des dépendances dans TypeScript.
- Gestion des dépendances dans JavaScript.
- Gestion des dépendances dans Python.
- Gestion des dépendances dans Java.
- Gestion des dépendances dans C#.
- Gestion des dépendances dans Go.

Comparaison AWS CDKTypeScript avec d'autres langues

TypeScript a été le premier langage pris en charge pour le développement AWS CDK d'applications. Par conséquent, une quantité importante d'exemples de code CDK est écrite dedans TypeScript. Si vous développez dans une autre langue, il peut être utile de comparer la façon dont le AWS CDK

code est implémenté TypeScript par rapport à la langue de votre choix. Cela peut vous aider à utiliser les exemples tout au long de la documentation.

Importer un module

TypeScript/JavaScript

TypeScript prend en charge l'importation d'un espace de noms complet ou d'objets individuels à partir d'un espace de noms. Chaque espace de noms inclut des constructions et d'autres classes à utiliser avec un service donné 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

Par exemple TypeScript, Python prend en charge les importations de modules avec espace de noms et les importations sélectives. Les espaces de noms en Python ressemblent à aws_cdk.

xxx, où xxx représente un nom AWS de service, tel que s3 pour Amazon S3. (Amazon S3 est utilisé dans ces exemples).

```
# 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

Les importations de Java fonctionnent différemment de celles TypeScript de Java. Chaque instruction d'importation importe soit un nom de classe unique à partir d'un package donné, soit toutes les classes définies dans ce package (en utilisant*). Les classes sont accessibles en utilisant soit le nom de classe lui-même s'il a été importé, soit le nom de classe qualifié, y compris son package.

Les bibliothèques portent le même nom que `software.amazon.awscdk.services.xxx` la bibliothèque AWS Construct (la bibliothèque principale est`software.amazon.awscdk`). L'ID de groupe Maven pour les AWS CDK packages est`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#, vous importez des types avec la `using` directive. Il existe deux styles. L'un vous donne accès à tous les types de l'espace de noms spécifié en utilisant leurs noms simples. Avec l'autre, vous pouvez faire référence à l'espace de noms lui-même en utilisant un alias.

Les packages sont nommés de la même manière que `Amazon.CDK.AWS.xxx` les packages AWS Construct Library. (Le module de base est`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

Chaque module AWS Construct Library est fourni sous forme de package 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(...)
```

Instanciation d'une construction

AWS CDK les classes de construction portent le même nom dans tous les langages pris en charge. La plupart des langages utilisent le `new` mot-clé pour instancier une classe (Python et Go ne le font pas). De plus, dans la plupart des langues, le mot clé `this` fait référence à l'instance actuelle. (Python l'utilise `self` par convention.) Vous devez transmettre une référence à l'instance actuelle comme `scope` paramètre de chaque construction que vous créez.

Le troisième argument d'une AWS CDK construction est `props` un objet contenant les attributs nécessaires à la construction de la construction. Cet argument peut être facultatif, mais lorsqu'il est requis, les langues prises en charge le gèrent de manière idiomatique. Les noms des attributs sont également adaptés aux modèles de dénomination standard de la langue.

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 n'utilise pas de `new` mot-clé lors de l'instanciation d'une classe. L'argument properties est représenté à l'aide d'arguments de mots clés, et les arguments sont nommés en utilisant`snake_case`.

Si une valeur props est elle-même un ensemble d'attributs, elle est représentée par une classe nommée d'après la propriété, qui accepte les arguments par mot clé pour les sous-propriétés.

En Python, l'instance actuelle est transmise aux méthodes en tant que premier argument, nommé `self` par convention.

```
# 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, l'argument props est représenté par une classe nommée `XxxxProps` (par exemple, `BucketProps` pour les accessoires de la `Bucket` construction). Vous créez l'argument props à l'aide d'un modèle de générateur.

Chaque `XxxxProps` classe possède un constructeur. Il existe également un générateur pratique pour chaque construction qui construit les accessoires et la construction en une seule étape, comme le montre l'exemple suivant.

Les accessoires portent le même nom que dans TypeScript, using`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#, les accessoires sont spécifiés à l'aide d'un initialiseur d'objet pour une classe nommée `XxxxProps` (par exemple, `BucketProps` pour les accessoires de la `Bucket` construction).

Les accessoires sont nommés de la même manière que TypeScript, sauf en utilisant`PascalCase`.

Il est pratique d'utiliser le `var` mot-clé lors de l'instanciation d'une construction, vous n'avez donc pas besoin de taper le nom de la classe deux fois. Cependant, votre guide de style de code local peut varier.

```
// 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

Pour créer une construction dans Go, appelez la fonction `NewXxxxxx` où `Xxxxxxx` est le nom de la construction. Les propriétés des constructions sont définies en tant que structure.

Dans Go, tous les paramètres de construction sont des pointeurs, y compris des valeurs telles que des nombres, des booléens et des chaînes. Utilisez les fonctions pratiques telles que `jsii.String` la création de ces pointeurs.

```
// 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"),
 }})
```

Accès aux membres

Il est courant de faire référence aux attributs ou aux propriétés des constructions et autres AWS CDK classes et d'utiliser ces valeurs, par exemple, comme entrées pour créer d'autres constructions. Les différences de dénomination décrites précédemment pour les méthodes s'appliquent également ici. De plus, en Java, il n'est pas possible d'accéder directement aux membres. Au lieu de cela, une méthode getter est fournie.

TypeScript/JavaScript

Les noms sontcamelCase.

```
bucket.bucketArn
```

Python

Les noms sontsnake_case.

```
bucket.bucket_arn
```

Java

Une méthode getter est fournie pour chaque propriété ; ces noms sontcamelCase.

```
bucket.getBucketArn()
```

C#

Les noms sontPascalCase.

```
bucket.BucketArn
```

Go

Les noms sontPascalCase.

```
bucket.BucketArn
```

Constantes Enum

Les constantes Enum sont limitées à une classe et portent des noms en majuscules avec des traits de soulignement dans toutes les langues (parfois appelées). SCREAMING_SNAKE_CASE Étant donné que les noms de classe utilisent également le même boîtier dans toutes les langues prises en charge à l'exception de Go, les noms d'énumération qualifiés sont également les mêmes dans ces langues.

```
s3.BucketEncryption.KMS_MANAGED
```

Dans Go, les constantes enum sont des attributs de l'espace de noms du module et sont écrites comme suit.

```
awss3.BucketEncryption_KMS_MANAGED
```

Interfaces d'objets

AWS CDK Utilise des interfaces d' TypeScript objets pour indiquer qu'une classe implémente un ensemble attendu de méthodes et de propriétés. Vous pouvez reconnaître une interface d'objet car son nom commence par`I`. Une classe concrète indique les interfaces qu'elle implémente en utilisant le `implements` mot clé.

TypeScript/JavaScript

> **Note**
>
> JavaScript n'a pas de fonctionnalité d'interface. Vous pouvez ignorer le `implements` mot clé et les noms de classe qui le suivent.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';

class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

Python ne possède pas de fonctionnalité d'interface. Cependant, pour le, AWS CDK vous pouvez indiquer l'implémentation de l'interface en décorant votre classe avec`@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

Les structures Go n'ont pas besoin de déclarer explicitement les interfaces qu'elles implémentent. Le compilateur Go détermine l'implémentation en fonction des méthodes et des propriétés disponibles sur la structure. Par exemple, dans le code suivant, `MyAspect` implémente l'`IAspect`interface car elle fournit une `Visit` méthode qui prend une construction.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
 fmt.Println("Visited", *node.Node().Path())
}
```

Travailler avec le AWS CDK in TypeScript

TypeScript est une langue client entièrement prise en charge pour le AWS Cloud Development Kit (AWS CDK) et est considérée comme stable. L'utilisation de l' AWS CDK in TypeScript utilise des outils familiers, notamment le TypeScript compilateur (`tsc`) de Microsoft, Node.js et le Node Package Manager (`npm`). Vous pouvez également utiliser Yarn si vous préférez, bien que les exemples de ce guide utilisent NPM. Les modules composant la bibliothèque AWS Construct sont distribués via le référentiel NPM, npmjs.org.

Vous pouvez utiliser n'importe quel éditeur ou IDE. De nombreux AWS CDK développeurs utilisent Visual Studio Code (ou son équivalent open source VSCodium), qui bénéficie d'un excellent support pour. TypeScript

Rubriques

Commencez avec TypeScript

Pour utiliser le AWS CDK, vous devez disposer d'un AWS compte et d'informations d'identification et avoir installé Node.js et le AWS CDK Toolkit. veuillez consulter Commencer à utiliser le AWS CDK.

Vous avez également besoin de TypeScript lui-même (version 3.8 ou ultérieure). Si vous ne l'avez pas déjà, vous pouvez l'installer en utilisant`npm`.

```
npm install -g typescript
```

> **Note**
>
> Si une erreur d'autorisation s'affiche et que vous disposez d'un accès administrateur sur votre système, essayez`sudo npm install -g typescript`.

TypeScript Tenez-vous au courant avec un habituénpm update -g typescript.

> **Note**
>
> Obsolète linguistique tierce : la version linguistique n'est prise en charge que jusqu'à sa fin de vie (EOL) partagée par le fournisseur ou la communauté et est sujette à modification avec préavis.

Création d'un projet

Vous créez un nouveau AWS CDK projet en l'invoquant `cdk init` dans un répertoire vide. Utilisez l'`--language`option et spécifiez `typescript` :

```
mkdir my-project
cd my-project
cdk init app --language typescript
```

La création d'un projet installe également le `aws-cdk-lib`module et ses dépendances.

`cdk init`utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Toutefois, le nom doit sinon prendre la forme d'un TypeScript identifiant ; par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Utilisation de locaux **`tsc`** et **`cdk`**

Dans l'ensemble, ce guide suppose que vous installez TypeScript le kit d'outils CDK globalement (`npm install -g typescript aws-cdk`), et les exemples de commandes fournis (tels que`cdk synth`) suivent cette hypothèse. Cette approche facilite la mise à jour des deux composants, et comme les deux adoptent une approche stricte en matière de rétrocompatibilité, il y a généralement peu de risques à toujours utiliser les dernières versions.

Certaines équipes préfèrent spécifier toutes les dépendances au sein de chaque projet, y compris des outils tels que le TypeScript compilateur et le kit d'outils CDK. Cette pratique vous permet d'associer ces composants à des versions spécifiques et de vous assurer que tous les développeurs de votre équipe (et de votre environnement CI/CD) utilisent exactement ces versions. Cela élimine toute source potentielle de changement, ce qui contribue à rendre les builds et les déploiements plus cohérents et reproductibles.

Le CDK inclut des dépendances pour les deux TypeScript et pour le kit d'outils CDK dans les modèles de TypeScript projet. Ainsi`package.json`, si vous souhaitez utiliser cette approche, vous n'avez pas besoin d'apporter de modifications à votre projet. Il vous suffit d'utiliser des commandes légèrement différentes pour créer votre application et pour émettre des `cdk` commandes.

Opération	Utiliser des outils mondiaux	Utiliser des outils locaux
Initialiser le projet	`cdk init --language typescript`	`npx aws-cdk init --language typescript`
Génération	`tsc`	`npm run build`
Exécuter la commande CDK Toolkit	`cdk ...`	`npm run cdk ...` or `npx aws-cdk ...`

`npx aws-cdk`exécute la version du kit d'outils CDK installée localement dans le projet en cours, s'il en existe une, en revenant à l'installation globale, le cas échéant. S'il n'existe aucune installation globale, `npx` télécharge une copie temporaire du kit CDK et l'exécute. Vous pouvez spécifier une version arbitraire du kit CDK en utilisant la `@` syntaxe : `npx aws-cdk@2.0 --version` prints`2.0.0`.

> **Tip**
>
> Configurez un alias afin de pouvoir utiliser la `cdk` commande avec une installation locale du CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Gestion des modules AWS de la bibliothèque Construct

Utilisez le Node Package Manager (`npm`) pour installer et mettre à jour les modules AWS Construct Library destinés à être utilisés par vos applications, ainsi que par les autres packages dont vous avez besoin. (Vous pouvez utiliser à la `yarn` place de `npm` si vous préférez.) `npm`installe également automatiquement les dépendances de ces modules.

La plupart AWS CDK des constructions se trouvent dans le package CDK principal, nommé`aws-cdk-lib`, qui est une dépendance par défaut dans les nouveaux projets créés par. cdk init Les modules de la bibliothèque de AWS construction « expérimentaux », où des constructions de niveau supérieur sont encore en cours de développement, sont nommés ainsi. `@aws-cdk/`*`SERVICE-NAME`*`-alpha` Le nom du service comporte un préfixe aws-. Si vous n'êtes pas sûr du nom d'un module, recherchez-le sur NPM.

> **Note**
>
> La référence de l'API CDK indique également les noms des packages.

Par exemple, la commande ci-dessous installe le module expérimental pour AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

La prise en charge de la bibliothèque Construct par certains services se trouve dans plusieurs espaces de noms. Par exemple`aws-route53`, il existe en outre trois espaces de noms Amazon Route 53 supplémentaires, `aws-route53-targets``aws-route53-patterns`, et`aws-route53resolver`.

Les dépendances de votre projet sont conservées dans`package.json`. Vous pouvez modifier ce fichier pour verrouiller certaines ou toutes vos dépendances dans une version spécifique ou pour permettre leur mise à jour vers des versions plus récentes selon certains critères. Pour mettre à jour les dépendances NPM de votre projet vers la dernière version autorisée conformément aux règles que vous avez spécifiées dans `package.json` :

```
npm update
```

Dans TypeScript, vous importez des modules dans votre code sous le même nom que celui que vous utilisez pour les installer à l'aide de NPM. Nous recommandons les pratiques suivantes lors

de l'importation de AWS CDK classes et de modules AWS Construct Library dans vos applications. Le respect de ces directives vous aidera à rendre votre code cohérent avec AWS CDK les autres applications et à le rendre plus facile à comprendre.

- N'utilisez pas les `import` directives de style ES6. `require()`
- En général, importez des classes individuelles depuis`aws-cdk-lib`.

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

- Si vous avez besoin de nombreuses classes`aws-cdk-lib`, vous pouvez utiliser un alias d'espace de noms de `cdk` au lieu d'importer les classes individuelles. Évitez de faire les deux.

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

- En général, importez des structures AWS de service utilisant des alias d'espace de noms courts.

```
import { aws_s3 as s3 } from 'aws-cdk-lib';
```

Gestion des dépendances dans TypeScript

Dans les projets TypeScript CDK, les dépendances sont spécifiées dans le `package.json` fichier du répertoire principal du projet. Les AWS CDK modules principaux se trouvent dans un seul NPM package appelé`aws-cdk-lib`.

Lorsque vous installez un package à l'aide denpm install, NPM enregistre le package `package.json` pour vous.

Si vous préférez, vous pouvez utiliser Yarn à la place de NPM. Cependant, le CDK ne prend pas en charge le plug-and-play mode Yarn, qui est le mode par défaut dans Yarn 2. Ajoutez ce qui suit au `.yarnrc.yml` fichier de votre projet pour désactiver cette fonctionnalité.

```
nodeLinker: node-modules
```

Applications CDK

Voici un exemple de `package.json` fichier généré par la `cdk init --language typescript` commande :

```
{
```

```
  "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"
  }
}
```

Pour les applications CDK déployables, cela `aws-cdk-lib` doit être spécifié dans la `dependencies` section de. `package.json` Vous pouvez utiliser un spécificateur de numéro de version caret (^) pour indiquer que vous accepterez des versions ultérieures à celle spécifiée, à condition qu'elles soient dans la même version majeure.

Pour les constructions expérimentales, spécifiez les versions exactes des modules de la bibliothèque de constructions alpha, dont les API peuvent changer. N'utilisez pas ^ ou ~ car les versions ultérieures de ces modules peuvent entraîner des modifications de l'API susceptibles de perturber votre application.

Spécifiez les versions des bibliothèques et des outils nécessaires pour tester votre application (par exemple, le framework de `jest` test) dans la `devDependencies` section de`package.json`. Utilisez éventuellement ^ pour indiquer que les versions compatibles ultérieures sont acceptables.

Bibliothèques de construction tierces

Si vous développez une bibliothèque de constructions, spécifiez ses dépendances à l'aide d'une combinaison des `devDependencies` sections `peerDependencies` et, comme indiqué dans l'exemple de `package.json` fichier suivant.

```
{
  "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"
  }
}
```

Dans`peerDependencies`, utilisez un curseur (^) pour spécifier la version la plus basse de `aws-cdk-lib` celle avec laquelle votre bibliothèque fonctionne. Cela maximise la compatibilité de votre bibliothèque avec une gamme de versions de CDK. Spécifiez les versions exactes des modules de la bibliothèque de construction alpha, dont les API peuvent changer. L'utilisation `peerDependencies` garantit qu'il n'y a qu'une seule copie de toutes les bibliothèques CDK dans l'`node_modules`arborescence.

Dans`devDependencies`, spécifiez les outils et les bibliothèques dont vous avez besoin pour les tests, éventuellement avec ^ pour indiquer que les versions compatibles ultérieures sont acceptables. Spécifiez exactement (sans ^ ou ~) les versions les plus basses `aws-cdk-lib` et les autres packages CDK avec lesquels vous annoncez que votre bibliothèque doit être compatible. Cette pratique garantit que vos tests s'exécutent par rapport à ces versions. Ainsi, si vous utilisez par inadvertance une fonctionnalité présente uniquement dans les versions les plus récentes, vos tests peuvent la détecter.

> **Warning**
>
> `peerDependencies`sont installés automatiquement uniquement par NPM 7 et versions ultérieures. Si vous utilisez NPM 6 ou une version antérieure, ou si vous utilisez Yarn, vous devez inclure les dépendances de vos dépendances dans`devDependencies`. Dans le cas contraire, ils ne seront pas installés et vous recevrez un avertissement concernant les dépendances non résolues entre pairs.

Installation et mise à jour des dépendances

Exécutez la commande suivante pour installer les dépendances de votre projet.

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
```

Pour mettre à jour les modules installés, les yarn upgrade commandes npm install et précédentes peuvent être utilisées. L'une ou l'autre des commandes met `node_modules` à jour les packages vers les dernières versions conformes aux règles de`package.json`. Cependant, ils ne `package.json` se mettent pas à jour d'eux-mêmes, ce que vous pouvez faire pour définir une nouvelle version minimale. Si vous hébergez votre package sur GitHub, vous pouvez configurer les mises à jour de version de Dependabot pour qu'elles soient mises à jour `package.json` automatiquement. Sinon, utilisez npm-check-updates.

> **Important**
>
> Par conception, lorsque vous installez ou mettez à jour des dépendances, NPM et Yarn choisissent la dernière version de chaque package qui répond aux exigences spécifiées dans. `package.json` Il existe toujours un risque que ces versions soient cassées (accidentellement ou intentionnellement). Effectuez des tests approfondis après avoir mis à jour les dépendances de votre projet.

AWS CDK expressions idiomatiques dans TypeScript

accessoires

Toutes les classes de la bibliothèque de AWS construction sont instanciées à l'aide de trois arguments : la portée dans laquelle la construction est définie (son parent dans l'arbre de construction), un identifiant et des accessoires. L'argument props est un ensemble de paires clé/valeur que la construction utilise pour configurer les AWS ressources qu'elle crée. D'autres classes et méthodes utilisent également le modèle « bundle of attributes » pour les arguments.

Dans TypeScript, la forme de `props` est définie à l'aide d'une interface qui vous indique les arguments obligatoires et facultatifs ainsi que leurs types. Une telle interface est définie pour chaque type d'`props`argument, généralement spécifique à une construction ou à une méthode unique. Par exemple, la construction Bucket (dans le`aws-cdk-lib/aws-s3 module`) spécifie un `props` argument conforme à l'BucketPropsinterface.

Si une propriété est elle-même un objet, par exemple la propriété WebsiteRedirect de`BucketProps`, cet objet aura sa propre interface à laquelle sa forme doit être conforme, dans ce cas. RedirectTarget

Si vous sous-classez une classe AWS Construct Library (ou si vous remplacez une méthode qui prend un argument semblable à des accessoires), vous pouvez hériter de l'interface existante pour en créer une nouvelle qui spécifie les nouveaux accessoires requis par votre code. Lorsque vous appelez la classe parent ou la méthode de base, vous pouvez généralement transmettre l'intégralité de l'argument props que vous avez reçu, car tous les attributs fournis dans l'objet mais non spécifiés dans l'interface seront ignorés.

Une future version du AWS CDK pourrait par hasard ajouter une nouvelle propriété avec un nom que vous avez utilisé pour votre propre propriété. Le transfert de la valeur que vous recevez vers le haut de la chaîne d'héritage peut alors provoquer un comportement inattendu. Il est plus sûr de

transmettre une copie sommaire des accessoires que vous avez reçus lorsque vos biens ont été retirés ou transférésundefined. Par exemple :

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Vous pouvez également nommer vos propriétés de manière à ce qu'il soit clair qu'elles appartiennent à votre construction. De cette façon, il est peu probable qu'ils entrent en collision avec des propriétés dans les futures AWS CDK versions. S'ils sont nombreux, utilisez un seul objet portant le nom approprié pour les conserver.

Valeurs manquantes

Les valeurs manquantes dans un objet (comme les accessoires) ont pour valeur undefined in TypeScript. La version 3.7 du langage a introduit des opérateurs qui simplifient le travail avec ces valeurs, en facilitant la spécification des valeurs par défaut et en « court-circuitant » le chaînage lorsqu'une valeur indéfinie est atteinte. Pour plus d'informations sur ces fonctionnalités, consultez les notes de mise à jour de la version TypeScript 3.7, en particulier les deux premières fonctionnalités, le chaînage facultatif et la coalescence nulle.

Création, synthèse et déploiement

En règle générale, vous devez vous trouver dans le répertoire racine du projet lors de la création et de l'exécution de votre application.

Node.js ne peut pas s'exécuter TypeScript directement ; au lieu de cela, votre application est convertie à l' JavaScript aide du TypeScript compilateurtsc. Le JavaScript code obtenu est ensuite exécuté.

Il le fait AWS CDK automatiquement chaque fois qu'il doit exécuter votre application. Cependant, il peut être utile de compiler manuellement pour vérifier les erreurs et pour exécuter des tests. Pour compiler votre TypeScript application manuellement, lanceznpm run build. Vous pouvez également npm run watch entrer en mode veille, dans lequel le TypeScript compilateur reconstruit automatiquement votre application chaque fois que vous enregistrez des modifications dans un fichier source.

Les piles définies dans votre AWS CDK application peuvent être synthétisées et déployées individuellement ou ensemble à l'aide des commandes ci-dessous. En général, vous devez vous trouver dans le répertoire principal de votre projet lorsque vous les publiez.

- `cdk synth`: synthétise un AWS CloudFormation modèle à partir d'une ou de plusieurs piles de votre AWS CDK application.
- `cdk deploy`: déploie les ressources définies par une ou plusieurs piles de votre AWS CDK application vers. AWS

Vous pouvez spécifier les noms de plusieurs piles à synthétiser ou à déployer en une seule commande. Si votre application ne définit qu'une seule pile, il n'est pas nécessaire de la spécifier.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Vous pouvez également utiliser les caractères génériques * (n'importe quel nombre de caractères) et ? (n'importe quel caractère) pour identifier les piles par modèle. Lorsque vous utilisez des caractères génériques, placez le motif entre guillemets. Sinon, le shell peut essayer de l'étendre aux noms des fichiers du répertoire actuel avant qu'ils ne soient transmis au AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Il n'est pas nécessaire de synthétiser explicitement les piles avant de les déployer. `cdk deploy` Effectuez cette étape pour vous assurer que votre dernier code est déployé.

Pour une documentation complète de la `cdk` commande, consultezthe section called “AWS CDK Boîte à outils”.

Travailler avec le AWS CDK in JavaScript

JavaScriptest une langue client entièrement prise en charge pour le AWS CDK et est considérée comme stable. L'utilisation du AWS Cloud Development Kit (AWS CDK) in JavaScript utilise des outils familiers, notamment Node.js et le Node Package Manager (`npm`). Vous pouvez également utiliser Yarn si vous préférez, bien que les exemples de ce guide utilisent NPM. Les modules composant la bibliothèque AWS Construct sont distribués via le référentiel NPM, npmjs.org.

Vous pouvez utiliser n'importe quel éditeur ou IDE. De nombreux AWS CDK développeurs utilisent Visual Studio Code (ou son équivalent open source VSCodium), qui bénéficie d'un bon support pour. JavaScript

Rubriques

- Premiers pas avec JavaScript
- Création d’un projet
- Utilisation du local cdk
- Gestion des modules AWS de la bibliothèque Construct
- Gestion des dépendances dans JavaScript
- AWS CDK expressions idiomatiques dans JavaScript
- Synthèse et déploiement
- À l'aide TypeScript d'exemples avec JavaScript
- Migration vers TypeScript

Premiers pas avec JavaScript

Pour utiliser le AWS CDK, vous devez disposer d'un AWS compte et d'informations d'identification et avoir installé Node.js et le AWS CDK Toolkit. veuillez consulter Commencer à utiliser le AWS CDK.

JavaScript AWS CDK les applications ne nécessitent aucun prérequis supplémentaire.

> **Note**
>
> Obsolète linguistique tierce : la version linguistique n'est prise en charge que jusqu'à sa fin de vie (EOL) partagée par le fournisseur ou la communauté et est sujette à modification avec préavis.

Création d’un projet

Vous créez un nouveau AWS CDK projet en l'invoquant `cdk init` dans un répertoire vide. Utilisez l'`--language`option et spécifiez `javascript` :

```
mkdir my-project
```

```
cd my-project
cdk init app --language javascript
```

La création d'un projet installe également le `aws-cdk-lib`module et ses dépendances.

`cdk init`utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Toutefois, le nom doit sinon prendre la forme d'un JavaScript identifiant ; par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Utilisation du local **cdk**

Dans l'ensemble, ce guide suppose que vous installez le kit d'outils CDK globalement (`npm install -g aws-cdk`), et les exemples de commandes fournis (tels que`cdk synth`) suivent cette hypothèse. Cette approche facilite la mise à jour du kit d'outils CDK, et comme le CDK adopte une approche stricte en matière de rétrocompatibilité, il y a généralement peu de risques à toujours utiliser la dernière version.

Certaines équipes préfèrent spécifier toutes les dépendances au sein de chaque projet, y compris des outils tels que le kit d'outils CDK. Cette pratique vous permet d'associer ces composants à des versions spécifiques et de vous assurer que tous les développeurs de votre équipe (et de votre environnement CI/CD) utilisent exactement ces versions. Cela élimine toute source potentielle de changement, ce qui contribue à rendre les builds et les déploiements plus cohérents et reproductibles.

Le CDK inclut une dépendance pour le kit d'outils CDK dans le modèle de JavaScript projet`package.json`, donc si vous souhaitez utiliser cette approche, vous n'avez pas besoin d'apporter de modifications à votre projet. Il vous suffit d'utiliser des commandes légèrement différentes pour créer votre application et pour émettre des `cdk` commandes.

Opération	Utilisez le kit d'outils CDK mondial	Utiliser le kit d'outils CDK local
Initialiser le projet	`cdk init --language javascript`	`npx aws-cdk init --language javascript`
Exécuter la commande CDK Toolkit	`cdk...`	`npm run cdk...` or `npx aws-cdk...`

`npx aws-cdk`exécute la version du kit d'outils CDK installée localement dans le projet en cours, s'il en existe une, en revenant à l'installation globale, le cas échéant. S'il n'existe aucune installation globale, `npx` télécharge une copie temporaire du kit CDK et l'exécute. Vous pouvez spécifier une version arbitraire du kit CDK en utilisant la `@` syntaxe : `npx aws-cdk@1.120 --version` prints`1.120.0`.

> **Tip**
>
> Configurez un alias afin de pouvoir utiliser la `cdk` commande avec une installation locale du CDK Toolkit.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

Gestion des modules AWS de la bibliothèque Construct

Utilisez le Node Package Manager (`npm`) pour installer et mettre à jour les modules AWS Construct Library destinés à être utilisés par vos applications, ainsi que par les autres packages dont vous avez besoin. (Vous pouvez utiliser à la `yarn` place de `npm` si vous préférez.) `npm`installe également automatiquement les dépendances de ces modules.

La plupart AWS CDK des constructions se trouvent dans le package CDK principal, nommé`aws-cdk-lib`, qui est une dépendance par défaut dans les nouveaux projets créés par. cdk init Les modules de la bibliothèque de AWS construction « expérimentaux », où des constructions de niveau supérieur sont encore en cours de développement, sont nommés ainsi. `aws-cdk-lib/`*`SERVICE-NAME`*`-alpha` Le nom du service comporte un préfixe aws-. Si vous n'êtes pas sûr du nom d'un module, recherchez-le sur NPM.

> **Note**
>
> La référence de l'API CDK indique également les noms des packages.

Par exemple, la commande ci-dessous installe le module expérimental pour AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

La prise en charge de la bibliothèque Construct par certains services se trouve dans plusieurs espaces de noms. Par exemple`aws-route53`, il existe en outre trois espaces de noms Amazon Route 53 supplémentaires, `aws-route53-targets``aws-route53-patterns`, et`aws-route53resolver`.

Les dépendances de votre projet sont conservées dans`package.json`. Vous pouvez modifier ce fichier pour verrouiller certaines ou toutes vos dépendances dans une version spécifique ou pour permettre leur mise à jour vers des versions plus récentes selon certains critères. Pour mettre à jour les dépendances NPM de votre projet vers la dernière version autorisée conformément aux règles que vous avez spécifiées dans `package.json` :

```
npm update
```

Dans JavaScript, vous importez des modules dans votre code sous le même nom que celui que vous utilisez pour les installer à l'aide de NPM. Nous recommandons les pratiques suivantes lors de l'importation de AWS CDK classes et de modules AWS Construct Library dans vos applications. Le respect de ces directives vous aidera à rendre votre code cohérent avec AWS CDK les autres applications et à le rendre plus facile à comprendre.

- Utilisez`require()`, pas des directives de style ES6. `import` Les anciennes versions de Node.js ne prennent pas en charge les importations ES6. L'utilisation de l'ancienne syntaxe est donc plus largement compatible. (Si vous souhaitez vraiment utiliser les importations ES6, utilisez esm pour vous assurer que votre projet est compatible avec toutes les versions prises en charge de Node.js.)
- En général, importez des classes individuelles depuis`aws-cdk-lib`.

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

- Si vous avez besoin de nombreuses classes`aws-cdk-lib`, vous pouvez utiliser un alias d'espace de noms de `cdk` au lieu d'importer les classes individuelles. Évitez de faire les deux.

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

- En général, importez des bibliothèques de AWS construction en utilisant des alias d'espace de noms courts.

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

Gestion des dépendances dans JavaScript

Dans les projets JavaScript CDK, les dépendances sont spécifiées dans le `package.json` fichier du répertoire principal du projet. Les AWS CDK modules principaux se trouvent dans un seul NPM package appelé`aws-cdk-lib`.

Lorsque vous installez un package à l'aide denpm install, NPM enregistre le package `package.json` pour vous.

Si vous préférez, vous pouvez utiliser Yarn à la place de NPM. Cependant, le CDK ne prend pas en charge le plug-and-play mode Yarn, qui est le mode par défaut dans Yarn 2. Ajoutez ce qui suit au `.yarnrc.yml` fichier de votre projet pour désactiver cette fonctionnalité.

```
nodeLinker: node-modules
```

Applications CDK

Voici un exemple de `package.json` fichier généré par la `cdk init --language typescript` commande. Le fichier généré pour JavaScript est similaire, mais sans les TypeScript entrées associées.

```
{
  "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"
  }
}
```

Pour les applications CDK déployables, cela `aws-cdk-lib` doit être spécifié dans la `dependencies` section de. `package.json` Vous pouvez utiliser un spécificateur de numéro de version caret (^) pour indiquer que vous accepterez des versions ultérieures à celle spécifiée, à condition qu'elles soient dans la même version majeure.

Pour les constructions expérimentales, spécifiez les versions exactes des modules de la bibliothèque de constructions alpha, dont les API peuvent changer. N'utilisez pas ^ ou ~ car les versions ultérieures de ces modules peuvent entraîner des modifications de l'API susceptibles de perturber votre application.

Spécifiez les versions des bibliothèques et des outils nécessaires pour tester votre application (par exemple, le framework de `jest` test) dans la `devDependencies` section de`package.json`. Utilisez éventuellement ^ pour indiquer que les versions compatibles ultérieures sont acceptables.

Bibliothèques de construction tierces

Si vous développez une bibliothèque de constructions, spécifiez ses dépendances à l'aide d'une combinaison des `devDependencies` sections `peerDependencies` et, comme indiqué dans l'exemple de `package.json` fichier suivant.

```
{
  "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"
  }
}
```

Dans`peerDependencies`, utilisez un curseur (^) pour spécifier la version la plus basse de `aws-cdk-lib` celle avec laquelle votre bibliothèque fonctionne. Cela maximise la compatibilité de votre bibliothèque avec une gamme de versions de CDK. Spécifiez les versions exactes des modules de la bibliothèque de construction alpha, dont les API sont susceptibles de changer. L'utilisation `peerDependencies` garantit qu'il n'y a qu'une seule copie de toutes les bibliothèques CDK dans l'`node_modules`arborescence.

Dans`devDependencies`, spécifiez les outils et les bibliothèques dont vous avez besoin pour les tests, éventuellement avec ^ pour indiquer que les versions compatibles ultérieures sont acceptables. Spécifiez exactement (sans ^ ou ~) les versions les plus basses `aws-cdk-lib` et les autres packages CDK avec lesquels vous annoncez que votre bibliothèque doit être compatible. Cette pratique garantit que vos tests s'exécutent par rapport à ces versions. Ainsi, si vous utilisez par inadvertance une fonctionnalité présente uniquement dans les versions les plus récentes, vos tests peuvent la détecter.

> **Warning**
>
> `peerDependencies`sont installés automatiquement uniquement par NPM 7 et versions ultérieures. Si vous utilisez NPM 6 ou une version antérieure, ou si vous utilisez Yarn, vous devez inclure les dépendances de vos dépendances dans`devDependencies`. Dans le cas contraire, ils ne seront pas installés et vous recevrez un avertissement concernant les dépendances non résolues entre pairs.

Installation et mise à jour des dépendances

Exécutez la commande suivante pour installer les dépendances de votre projet.

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
```

Pour mettre à jour les modules installés, les yarn upgrade commandes npm install et précédentes peuvent être utilisées. L'une ou l'autre des commandes met `node_modules` à jour les packages vers les dernières versions conformes aux règles de`package.json`. Cependant, ils ne `package.json` se mettent pas à jour d'eux-mêmes, ce que vous pouvez faire pour définir une nouvelle version minimale. Si vous hébergez votre package sur GitHub, vous pouvez configurer les mises à jour de version de Dependabot pour qu'elles soient mises à jour `package.json` automatiquement. Sinon, utilisez npm-check-updates.

> **Important**
>
> Par conception, lorsque vous installez ou mettez à jour des dépendances, NPM et Yarn choisissent la dernière version de chaque package qui répond aux exigences spécifiées dans. `package.json` Il existe toujours un risque que ces versions soient cassées (accidentellement ou intentionnellement). Effectuez des tests approfondis après avoir mis à jour les dépendances de votre projet.

AWS CDK expressions idiomatiques dans JavaScript

accessoires

Toutes les classes de la bibliothèque de AWS construction sont instanciées à l'aide de trois arguments : la portée dans laquelle la construction est définie (son parent dans l'arbre de construction), un identifiant, et props, un ensemble de paires clé/valeur que la construction utilise pour configurer les ressources qu'elle crée. AWS D'autres classes et méthodes utilisent également le modèle « bundle of attributes » pour les arguments.

L'utilisation d'un IDE ou d'un éditeur doté d'une bonne JavaScript saisie semi-automatique permet d'éviter les fautes d'orthographe dans les noms de propriété. Si une construction attend une `encryptionKeys` propriété, et que vous l'épelez`encryptionkeys`, lors de l'instanciation de la construction, vous n'avez pas transmis la valeur que vous vouliez. Cela peut provoquer une erreur au moment de la synthèse si la propriété est obligatoire, ou entraîner son ignorance silencieuse si elle est facultative. Dans ce dernier cas, il se peut que vous obteniez un comportement par défaut que vous vouliez remplacer. Faites particulièrement attention ici.

Lorsque vous sous-classez une classe AWS Construct Library (ou que vous remplacez une méthode qui prend un argument semblable à un accessoire), vous souhaiterez peut-être accepter des propriétés supplémentaires pour votre propre usage. Ces valeurs seront ignorées par la classe parent ou la méthode remplacée, car elles ne sont jamais accessibles dans ce code. Vous pouvez donc généralement transmettre tous les accessoires que vous avez reçus.

Une future version du AWS CDK pourrait par hasard ajouter une nouvelle propriété avec un nom que vous avez utilisé pour votre propre propriété. Le transfert de la valeur que vous recevez vers le haut de la chaîne d'héritage peut alors provoquer un comportement inattendu. Il est plus sûr de transmettre une copie sommaire des accessoires que vous avez reçus lorsque vos biens ont été retirés ou transférés`undefined`. Par exemple :

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Vous pouvez également nommer vos propriétés de manière à ce qu'il soit clair qu'elles appartiennent à votre construction. De cette façon, il est peu probable qu'ils entrent en collision avec des propriétés dans les futures AWS CDK versions. S'ils sont nombreux, utilisez un seul objet portant le nom approprié pour les conserver.

Valeurs manquantes

Les valeurs manquantes dans un objet (par exemple`props`) ont pour valeur `undefined` in JavaScript. Les techniques habituelles s'appliquent pour y faire face. Par exemple, un idiome courant pour accéder à une propriété dont la valeur n'est peut-être pas définie est le suivant :

```
// 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;
```

Cependant, s'il a peut avoir une autre valeur « fausse »`undefined`, il est préférable de rendre le test plus explicite. Ici, nous allons profiter du fait que `null` et `undefined` sommes égaux pour les tester tous les deux en même temps :

```
let c = a == null ? a : a.b;
```

> **Tip**
>
> Node.js 14.0 et versions ultérieures prennent en charge de nouveaux opérateurs qui peuvent simplifier la gestion des valeurs non définies. Pour plus d'informations, consultez les propositions facultatives de chaînage et de fusion nulle.

Synthèse et déploiement

Les piles définies dans votre AWS CDK application peuvent être synthétisées et déployées individuellement ou ensemble à l'aide des commandes ci-dessous. En général, vous devez vous trouver dans le répertoire principal de votre projet lorsque vous les publiez.

- `cdk synth`: synthétise un AWS CloudFormation modèle à partir d'une ou de plusieurs piles de votre AWS CDK application.
- `cdk deploy`: Déploie les ressources définies par une ou plusieurs piles de votre AWS CDK application vers. AWS

Vous pouvez spécifier les noms de plusieurs piles à synthétiser ou à déployer en une seule commande. Si votre application ne définit qu'une seule pile, il n'est pas nécessaire de la spécifier.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Vous pouvez également utiliser les caractères génériques * (n'importe quel nombre de caractères) et ? (n'importe quel caractère) pour identifier les piles par modèle. Lorsque vous utilisez des caractères génériques, placez le motif entre guillemets. Sinon, le shell peut essayer de l'étendre aux noms des fichiers du répertoire actuel avant qu'ils ne soient transmis au AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Il n'est pas nécessaire de synthétiser explicitement les piles avant de les déployer. `cdk deploy` Effectuez cette étape pour vous assurer que votre dernier code est déployé.

Pour une documentation complète de la `cdk` commande, consultezthe section called “AWS CDK Boîte à outils”.

À l'aide TypeScript d'exemples avec JavaScript

TypeScriptest le langage que nous utilisons pour le développer AWS CDK, et c'est le premier langage pris en charge pour le développement d'applications. De nombreux exemples de AWS CDK code disponibles y sont donc écrits TypeScript. Ces exemples de code peuvent être une bonne ressource pour JavaScript les développeurs ; il suffit de supprimer les parties TypeScript spécifiques du code.

TypeScript les extraits utilisent souvent le dernier ECMAScript `import` et des `export` mots clés pour importer des objets depuis d'autres modules et pour déclarer les objets à rendre disponibles en dehors du module actuel. Node.js vient de commencer à prendre en charge ces mots clés dans ses dernières versions. Selon la version de Node.js que vous utilisez (ou que vous souhaitez prendre en charge), vous pouvez réécrire les importations et les exportations pour utiliser l'ancienne syntaxe.

Les importations peuvent être remplacées par des appels à la `require()` fonction.

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');
```

Les exportations peuvent être attribuées à l'`module.exports`objet.

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**
>
> Une alternative à l'utilisation des anciennes méthodes d'importation et d'exportation consiste à utiliser le `esm`module.

Une fois que vous avez trié les importations et les exportations, vous pouvez accéder au code lui-même. Il se peut que vous rencontriez ces fonctionnalités couramment utilisées TypeScript :

- Tapez des annotations
- Définitions d'interface
- Type de conversions/moulages
- Modificateurs d'accès

Des annotations de type peuvent être fournies pour les variables, les membres de classe, les paramètres de fonction et les types de retour de fonction. Pour les variables, les paramètres et les

membres, les types sont spécifiés en suivant l'identifiant par deux points et le type. Les valeurs renvoyées par la fonction suivent la signature de la fonction et se composent de deux points et du type.

Pour convertir le code annoté en caractères typographiques JavaScript, supprimez les deux points et le type. Les membres de la classe doivent avoir une certaine valeur dans JavaScript ; définissez-leur sur `undefined` s'ils n'ont qu'une annotation de type dedans 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) {
    // ...
}
```

Dans TypeScript, les interfaces sont utilisées pour donner un nom à des ensembles de propriétés obligatoires et facultatives, ainsi qu'à leurs types. Vous pouvez ensuite utiliser le nom de l'interface comme annotation de type. TypeScript garantira que l'objet que vous utilisez, par exemple, comme argument d'une fonction possède les propriétés requises du bon type.

```
interface myFuncProps {
    code: lambda.Code,
```

```
    handler?: string
}
```

JavaScript ne possède pas de fonctionnalité d'interface, donc une fois que vous avez supprimé les annotations de type, supprimez complètement les déclarations d'interface.

Lorsqu'une fonction ou une méthode renvoie un type à usage général (tel que`object`), mais que vous souhaitez traiter cette valeur comme un type enfant plus spécifique pour accéder à des propriétés ou à des méthodes qui ne font pas partie de l'interface du type plus général, vous TypeScript permet de convertir la valeur en utilisant le nom `as` du type ou de l'interface. JavaScript ne le prend pas en charge (ou n'en a pas besoin), il suffit donc `as` de supprimer l'identifiant suivant. Une syntaxe de conversion moins courante consiste à utiliser un nom de type entre crochets `<LikeThis>` ; ces conversions doivent également être supprimées.

Enfin, TypeScript prend en charge les modificateurs d'accès `publicprotected`, et `private` pour les membres des classes. Tous les membres de la classe JavaScript sont publics. Supprimez simplement ces modificateurs partout où vous les voyez.

Savoir comment identifier et supprimer ces TypeScript fonctionnalités contribue grandement à l'adaptation des TypeScript extraits courts à. JavaScript Mais il peut s'avérer peu pratique de convertir TypeScript des exemples plus longs de cette manière, car ils sont plus susceptibles d'utiliser d'autres TypeScript fonctionnalités. Dans ces situations, nous recommandons la sucrase. Sucrase ne se plaindra pas si le code utilise une variable non définie, par exemple, comme le `tsc` ferait. S'il est syntaxiquement valide, à quelques exceptions près, Sucrase peut le traduire en. JavaScript Cela le rend particulièrement utile pour convertir des extraits de code qui ne peuvent pas être exécutés seuls.

Migration vers TypeScript

De nombreux JavaScript développeurs optent pour cette TypeScriptsolution au fur et à mesure que leurs projets deviennent plus importants et plus complexes. TypeScript est un surensemble de JavaScript (tout le JavaScript code est un code valide, aucune modification de votre TypeScript code n'est donc requise) et il s'agit également d'une langue prise en charge. AWS CDK Les annotations de type et les autres TypeScript fonctionnalités sont facultatives et peuvent être ajoutées à votre AWS CDK application au fur et à mesure que vous y trouvez de la valeur. TypeScript vous donne également un accès anticipé aux nouvelles JavaScript fonctionnalités, telles que le chaînage facultatif et la fusion nulle, avant leur finalisation, sans avoir à mettre à jour Node.js.

TypeScriptles interfaces « basées sur les formes », qui définissent des ensembles de propriétés obligatoires et facultatives (et leurs types) au sein d'un objet, permettent de détecter les erreurs

courantes lors de l'écriture du code et permettent à votre IDE de fournir plus facilement des conseils de saisie semi-automatique robustes et d'autres conseils de codage en temps réel.

Le codage TypeScript implique une étape supplémentaire : compiler votre application avec le TypeScript compilateur,`tsc`. Pour les AWS CDK applications classiques, la compilation ne prend que quelques secondes au maximum.

Le moyen le plus simple de migrer une JavaScript AWS CDK application existante TypeScript est de créer un nouveau TypeScript projet en utilisant`cdk init app --language typescript`, puis de copier vos fichiers source (et tous les autres fichiers nécessaires, tels que les actifs tels que le code source des AWS Lambda fonctions) dans le nouveau projet. Renommez vos JavaScript fichiers pour qu'ils se terminent par `.ts` et que vous commenciez à les développer en TypeScript.

Travailler avec le AWS CDK en Python

Python est un langage client entièrement pris en charge AWS Cloud Development Kit (AWS CDK) et est considéré comme stable. Travailler avec le AWS CDK en Python utilise des outils familiers, notamment l'implémentation standard de Python (CPython), des environnements virtuels avec `virtualenv` et le programme d'installation `pip` de packages Python. Les modules composant la bibliothèque AWS Construct sont distribués via pypi.org. La version Python de l'événement utilise AWS CDK des identifiants de type Python (par exemple, les noms de `snake_case` méthodes).

Vous pouvez utiliser n'importe quel éditeur ou IDE. De nombreux AWS CDK développeurs utilisent Visual Studio Code (ou son équivalent open source VSCodium), qui supporte bien Python via une extension officielle. L'éditeur IDLE inclus dans Python suffira pour démarrer. Les modules Python correspondants AWS CDK contiennent des indications de type, utiles pour un outil de linting ou un IDE qui prend en charge la validation de type.

Rubriques

- Premiers pas avec Python
- Création d’un projet
- Gestion des modules AWS de la bibliothèque Construct
- Gestion des dépendances dans Python
- AWS CDK expressions idiomatiques en Python
- Synthèse et déploiement

Premiers pas avec Python

Pour utiliser le AWS CDK, vous devez disposer d'un AWS compte et d'informations d'identification et avoir installé Node.js et le AWS CDK Toolkit. veuillez consulter Commencer à utiliser le AWS CDK.

AWS CDK Les applications Python nécessitent Python 3.6 ou version ultérieure. Si ce n'est pas déjà fait, téléchargez une version compatible pour votre système d'exploitation sur python.org. Si vous utilisez Linux, votre système est peut-être fourni avec une version compatible, ou vous pouvez l'installer à l'aide du gestionnaire de paquets de votre distribution (`yum`,`apt`, etc.). Les utilisateurs de Mac peuvent être intéressés par Homebrew, un gestionnaire de paquets de style Linux pour macOS.

> **Note**
>
> Obsolète linguistique tierce : la version linguistique n'est prise en charge que jusqu'à sa fin de vie (EOL) partagée par le fournisseur ou la communauté et est sujette à modification avec préavis.

Le programme d'installation du package Python et le gestionnaire d'`virtualenv`environnement virtuel sont également requis. `pip` Les installations Windows des versions compatibles de Python incluent ces outils. Sous Linux, ils `pip` `virtualenv` peuvent être fournis sous forme de packages séparés dans votre gestionnaire de packages. Vous pouvez également les installer à l'aide des commandes suivantes :

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

Si vous rencontrez une erreur d'autorisation, exécutez les commandes ci-dessus avec l'`--user`indicateur afin que les modules soient installés dans votre répertoire utilisateur, ou utilisez-les `sudo` pour obtenir les autorisations nécessaires pour installer les modules à l'échelle du système.

> **Note**
>
> Il est courant que les distributions Linux utilisent le nom de l'exécutable `python3` pour Python 3.x et fassent `python` référence à une installation de Python 2.x. Certaines distributions ont un package optionnel que vous pouvez installer et qui fait référence à Python 3 à la

`python` commande. À défaut, vous pouvez ajuster la commande utilisée pour exécuter votre application en la modifiant `cdk.json` dans le répertoire principal du projet.

> **Note**
>
> Sous Windows, vous souhaiterez peut-être invoquer Python (etpip) à l'aide de l'pyexécutable, le lanceur >Python pour Windows. Entre autres choses, le lanceur vous permet de spécifier facilement la version installée de Python que vous souhaitez utiliser.
> Si vous tapez python sur la ligne de commande pour afficher un message concernant l'installation de Python depuis le Windows Store, même après avoir installé une version Windows de Python, ouvrez le panneau des paramètres de gestion des alias d'exécution des applications de Windows et désactivez les deux entrées App Installer pour Python.

Création d'un projet

Vous créez un nouveau AWS CDK projet en l'appelant `cdk init` dans un répertoire vide. Utilisez l'`--language`option et spécifiez `python` :

```
mkdir my-project
cd my-project
cdk init app --language python
```

`cdk init`utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Toutefois, le nom doit sinon prendre la forme d'un identifiant Python ; par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Pour travailler avec le nouveau projet, activez son environnement virtuel. Cela permet d'installer les dépendances du projet localement dans le dossier du projet, plutôt que globalement.

```
source .venv/bin/activate
```

> **Note**
>
> Vous pouvez reconnaître qu'il s'agit de la commande Mac/Linux pour activer un environnement virtuel. Les modèles Python incluent un fichier batch, `source.bat`, qui

permet d'utiliser la même commande sous Windows. La commande Windows traditionnelle fonctionne également. `.venv\Scripts\activate.bat`
Si vous avez initialisé votre AWS CDK projet à l'aide de CDK Toolkit v1.70.0 ou version antérieure, votre environnement virtuel se trouve dans le `.env` répertoire au lieu de. `.venv`

> **Important**
>
> Activez l'environnement virtuel du projet dès que vous commencez à travailler dessus. Sinon, vous n'aurez pas accès aux modules qui y sont installés, et les modules que vous installez seront placés dans le répertoire global des modules Python (ou provoqueront une erreur d'autorisation).

Après avoir activé votre environnement virtuel pour la première fois, installez les dépendances standard de l'application :

```
python -m pip install -r requirements.txt
```

Gestion des modules AWS de la bibliothèque Construct

Utilisez le programme d'installation de packages Python pour installer et mettre à jour les modules AWS Construct Library destinés à être utilisés par vos applications, ainsi que par les autres packages dont vous avez besoin. pip pipinstalle également automatiquement les dépendances de ces modules. Si votre système n'est pas reconnu pip comme une commande autonome, invoquez-la pip en tant que module Python, comme ceci :

```
python -m pip PIP-COMMAND
```

La plupart AWS CDK des constructions sont incluses. `aws-cdk-lib` Les modules expérimentaux se trouvent dans des modules distincts nommés comme`aws-cdk.SERVICE-NAME.alpha`. Le nom du service inclut un préfixe aws. Si vous n'êtes pas sûr du nom d'un module, recherchez-le sur PyPI. Par exemple, la commande ci-dessous installe la AWS CodeStar bibliothèque.

```
python -m pip install aws-cdk.aws-codestar-alpha
```

Les constructions de certains services se trouvent dans plusieurs espaces de noms. Par exemple`aws-cdk.aws-route53`, il existe en outre trois espaces de noms Amazon Route

53 supplémentaires, nommés `aws-route53-targets`aws-route53-patterns, etaws-route53resolver.

> **Note**
>
> L'édition Python du CDK API Reference indique également les noms des packages.

Les noms utilisés pour importer les modules AWS Construct Library dans votre code Python sont les suivants.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

Nous recommandons les pratiques suivantes lors de l'importation de AWS CDK classes et de modules AWS Construct Library dans vos applications. Le respect de ces directives vous aidera à rendre votre code cohérent avec AWS CDK les autres applications et à le rendre plus facile à comprendre.

- En général, importez des classes individuelles depuis le niveau supérieur. `aws_cdk`

  ```
  from aws_cdk import App, Construct
  ```

- Si vous avez besoin de nombreuses classes provenant de`aws_cdk`, vous pouvez utiliser un alias d'espace de noms de `cdk` au lieu d'importer des classes individuelles. Évitez de faire les deux.

  ```
  import aws_cdk as cdk
  ```

- En général, importez des bibliothèques de AWS construction en utilisant des alias d'espace de noms courts.

  ```
  import aws_cdk.aws_s3 as s3
  ```

Après avoir installé un module, mettez à jour `requirements.txt` le fichier de votre projet, qui répertorie les dépendances de votre projet. Il est préférable de le faire manuellement plutôt que d'utiliser`pip freeze`. `pip freeze`capture les versions actuelles de tous les modules installés dans votre environnement virtuel Python, ce qui peut être utile lorsque vous regroupez un projet à exécuter ailleurs.

Cependant, vous ne `requirements.txt` devez généralement répertorier que les dépendances de haut niveau (modules dont dépend directement votre application) et non les dépendances de ces bibliothèques. Cette stratégie simplifie la mise à jour de vos dépendances.

Vous pouvez modifier `requirements.txt` pour autoriser les mises à niveau ; il suffit de remplacer le numéro de version == précédent par ~= pour autoriser les mises à niveau vers une version compatible supérieure, ou de supprimer complètement l'exigence de version pour spécifier la dernière version disponible du module.

Après avoir `requirements.txt` été modifiée de manière appropriée pour autoriser les mises à niveau, émettez cette commande pour mettre à niveau les modules installés de votre projet à tout moment :

```
pip install --upgrade -r requirements.txt
```

Gestion des dépendances dans Python

En Python, vous spécifiez les dépendances en les insérant `requirements.txt` pour les applications ou `setup.py` pour les bibliothèques de construction. Les dépendances sont ensuite gérées avec l'outil PIP. Le PIP est invoqué de l'une des manières suivantes :

```
pip command options
python -m pip command options
```

L'python -m pipinvocation fonctionne sur la plupart des systèmes ; pip nécessite que l'exécutable du PIP se trouve sur le chemin du système. Si pip cela ne fonctionne pas, essayez de le remplacer parpython -m pip.

La cdk init --language python commande crée un environnement virtuel pour votre nouveau projet. Cela permet à chaque projet de disposer de ses propres versions des dépendances, ainsi que d'un `requirements.txt` fichier de base. Vous devez activer cet environnement virtuel en l'exécutant source .venv/bin/activate chaque fois que vous commencez à travailler sur le projet.

Applications CDK

Voici un exemple de fichier `requirements.txt`. Comme PIP ne possède pas de fonction de verrouillage des dépendances, nous vous recommandons d'utiliser l'opérateur == pour spécifier les versions exactes de toutes les dépendances, comme indiqué ici.

```
aws-cdk-lib==2.14.0
aws-cdk.aws-appsync-alpha==2.10.0a0
```

L'installation d'un module avec pip install ne l'ajoute pas automatiquement à`requirements.txt`. Tu dois le faire toi-même. Si vous souhaitez effectuer une mise à niveau vers une version ultérieure d'une dépendance, modifiez son numéro de version dans`requirements.txt`.

Pour installer ou mettre à jour les dépendances de votre projet après la création ou la modification`requirements.txt`, exécutez ce qui suit :

```
python -m pip install -r requirements.txt
```

> **Tip**
>
> La pip freeze commande affiche les versions de toutes les dépendances installées dans un format qui peut être écrit dans un fichier texte. Cela peut être utilisé comme fichier d'exigences avec`pip install -r`. Ce fichier est pratique pour attribuer toutes les dépendances (y compris les dépendances transitives) aux versions exactes avec lesquelles vous avez effectué les tests. Pour éviter tout problème lors de la mise à niveau ultérieure des packages, utilisez un fichier distinct, tel que `freeze.txt` (not`requirements.txt`). Puis, régénérez-le lorsque vous mettez à niveau les dépendances de votre projet.

Bibliothèques de construction tierces

Dans les bibliothèques, les dépendances sont spécifiées dans`setup.py`, de sorte que les dépendances transitives sont automatiquement téléchargées lorsque le package est consommé par une application. Dans le cas contraire, chaque application qui souhaite utiliser votre package doit copier vos dépendances dans son propre package`requirements.txt`. Un exemple `setup.py` est présenté ici.

```
from setuptools import setup

setup(
  name='my-package',
  version='0.0.1',
  install_requires=[
    'aws-cdk-lib==2.14.0',
  ],
```

```
  ...
)
```

Pour travailler sur le package à des fins de développement, créez ou activez un environnement virtuel, puis exécutez la commande suivante.

```
python -m pip install -e .
```

Bien que PIP installe automatiquement les dépendances transitives, il ne peut y avoir qu'une seule copie installée d'un package. La version spécifiée le plus haut dans l'arborescence des dépendances est sélectionnée ; les applications ont toujours le dernier mot quant à la version des packages à installer.

AWS CDK expressions idiomatiques en Python

Conflits linguistiques

En Python, `lambda` est un mot-clé de langage, vous ne pouvez donc pas l'utiliser comme nom pour le module de bibliothèque de AWS Lambda construction ou les fonctions Lambda. La convention de Python pour de tels conflits consiste à utiliser un trait de soulignement final, comme dans`lambda_`, dans le nom de la variable.

Par convention, le deuxième argument des AWS CDK constructions est nommé`id`. Lorsque vous écrivez vos propres piles et constructions, appelez un paramètre `id` « ombres » la fonction intégrée de Python`id()`, qui renvoie l'identifiant unique d'un objet. Cette fonction n'est pas souvent utilisée, mais si vous en avez besoin dans votre construction, renommez l'argument, par exemple`construct_id`.

Arguments et propriétés

Toutes les classes de la bibliothèque de AWS construction sont instanciées à l'aide de trois arguments : la portée dans laquelle la construction est définie (son parent dans l'arbre de construction), un identifiant, et props, un ensemble de paires clé/valeur que la construction utilise pour configurer les ressources qu'elle crée. D'autres classes et méthodes utilisent également le modèle « bundle of attributes » pour les arguments.

scope et id doivent toujours être transmis en tant qu'arguments positionnels, et non en tant qu'arguments de mot clé, car leurs noms changent si la construction accepte une propriété nommée scope ou id.

En Python, les accessoires sont exprimés sous forme d'arguments de mots clés. Si un argument contient des structures de données imbriquées, celles-ci sont exprimées à l'aide d'une classe qui prend ses propres arguments de mots clés lors de l'instanciation. Le même modèle est appliqué aux autres appels de méthode qui prennent un argument structuré.

Par exemple, dans la `add_lifecycle_rule` méthode d'un compartiment Amazon S3, la `transitions` propriété est une liste d'`Transition`instances.

```
bucket.add_lifecycle_rule(
  transitions=[
    Transition(
      storage_class=StorageClass.GLACIER,
      transition_after=Duration.days(10)
    )
  ]
)
```

Lorsque vous étendez une classe ou que vous remplacez une méthode, vous souhaiterez peut-être accepter des arguments supplémentaires pour vos propres besoins qui ne sont pas compris par la classe parent. Dans ce cas, vous devez accepter les arguments qui ne vous intéressent pas et utiliser des `**kwargs` arguments contenant uniquement des mots clés pour accepter les arguments qui vous intéressent. Lorsque vous appelez le constructeur du parent ou la méthode remplacée, transmettez uniquement les arguments attendus (souvent juste). `**kwargs` La transmission d'arguments auxquels la classe ou la méthode parent ne s'attend pas entraîne une erreur.

```
class MyConstruct(Construct):
    def __init__(self, id, *, MyProperty=42, **kwargs):
        super().__init__(self, id, **kwargs)
        # ...
```

Une future version du AWS CDK pourrait par hasard ajouter une nouvelle propriété avec un nom que vous avez utilisé pour votre propre propriété. Cela ne posera aucun problème technique aux utilisateurs de votre construction ou de votre méthode (étant donné que votre propriété n'est pas transmise « en haut de la chaîne », la classe parent ou la méthode remplacée utilisera simplement une valeur par défaut) mais cela peut être source de confusion. Vous pouvez éviter ce problème potentiel en nommant vos propriétés de manière à ce qu'elles appartiennent clairement à votre construction. S'il existe de nombreuses nouvelles propriétés, regroupez-les dans une classe au nom approprié et transmettez-la en tant qu'argument de mot clé unique.

Valeurs manquantes

Les AWS CDK utilisations `None` pour représenter des valeurs manquantes ou non définies. Lorsque vous travaillez avec`**kwargs`, utilisez la `get()` méthode du dictionnaire pour fournir une valeur par défaut si aucune propriété n'est fournie. Évitez d'en utiliser`kwargs[...]`, car cela augmente le `KeyError` nombre de valeurs manquantes.

```
encrypted = kwargs.get("encrypted")         # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)  # specify default of False if property is
 missing
```

Certaines AWS CDK méthodes (par exemple `tryGetContext()` pour obtenir une valeur de contexte d'exécution) peuvent être renvoyées`None`, ce que vous devrez vérifier explicitement.

Utilisation d'interfaces

Python ne possède pas de fonctionnalité d'interface comme certains autres langages, bien qu'il possède des classes de base abstraites, qui sont similaires. (Si vous n'êtes pas familier avec les interfaces, Wikipedia propose une bonne introduction.) TypeScript, le langage dans lequel le AWS CDK est implémenté, fournit des interfaces, et les constructions et autres AWS CDK objets nécessitent souvent un objet qui adhère à une interface particulière, plutôt que d'hériter d'une classe particulière. Il AWS CDK fournit donc sa propre fonctionnalité d'interface dans le cadre de la couche JSII.

Pour indiquer qu'une classe implémente une interface particulière, vous pouvez utiliser le `@jsii.implements` décorateur :

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Type de pièges

Python utilise le typage dynamique, où toutes les variables peuvent faire référence à une valeur de n'importe quel type. Les paramètres et les valeurs de retour peuvent être annotés avec des types, mais ce sont des « indices » qui ne sont pas appliqués. Cela signifie qu'en Python, il est facile de

transmettre le type de valeur incorrect à une AWS CDK construction. Au lieu de recevoir une erreur de type lors de la construction, comme c'est le cas dans un langage à typage statique, vous pouvez obtenir une erreur d'exécution lorsque la couche JSII (qui traduit entre Python et le TypeScript noyau) n'est pas en mesure AWS CDK de gérer le type inattendu.

D'après notre expérience, les erreurs de type commises par les programmeurs Python ont tendance à entrer dans ces catégories.

- Transmettre une valeur unique lorsqu'une construction attend un conteneur (liste Python ou dictionnaire) ou vice versa.
- Transmission d'une valeur d'un type associé à une construction de couche 1 (`CfnXxxxxx`) à une construction L2 ou L3, ou vice versa.

Les modules AWS CDK Python incluent des annotations de type. Vous pouvez donc utiliser les outils qui les prennent en charge pour vous aider à utiliser les types. Si vous n'utilisez pas un IDE qui les prend en charge, par exemple PyCharm, vous pouvez appeler le validateur de MyPytype comme étape de votre processus de construction. Il existe également des vérificateurs de type d'exécution qui peuvent améliorer les messages d'erreur relatifs aux erreurs liées au type.

Synthèse et déploiement

Les piles définies dans votre AWS CDK application peuvent être synthétisées et déployées individuellement ou ensemble à l'aide des commandes ci-dessous. En général, vous devez vous trouver dans le répertoire principal de votre projet lorsque vous les publiez.

- `cdk synth`: synthétise un AWS CloudFormation modèle à partir d'une ou de plusieurs piles de votre AWS CDK application.
- `cdk deploy`: Déploie les ressources définies par une ou plusieurs piles de votre AWS CDK application vers. AWS

Vous pouvez spécifier les noms de plusieurs piles à synthétiser ou à déployer en une seule commande. Si votre application ne définit qu'une seule pile, il n'est pas nécessaire de la spécifier.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Vous pouvez également utiliser les caractères génériques * (n'importe quel nombre de caractères) et ? (n'importe quel caractère) pour identifier les piles par modèle. Lorsque vous utilisez des

caractères génériques, placez le motif entre guillemets. Sinon, le shell peut essayer de l'étendre aux noms des fichiers du répertoire actuel avant qu'ils ne soient transmis au AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Il n'est pas nécessaire de synthétiser explicitement les piles avant de les déployer. `cdk deploy` Effectuez cette étape pour vous assurer que votre dernier code est déployé.

Pour une documentation complète de la `cdk` commande, consultezthe section called “AWS CDK Boîte à outils”.

Travailler avec le AWS CDK en Java

Java est un langage client entièrement pris en charge AWS CDK et est considéré comme stable. Vous pouvez développer des AWS CDK applications en Java à l'aide d'outils courants, notamment le JDK (celui d'Oracle ou une distribution OpenJDK telle qu'Amazon Corretto) et Apache Maven.

AWS CDK Supporte Java 8 et versions ultérieures. Nous vous recommandons toutefois d'utiliser la dernière version possible, car les versions ultérieures du langage incluent des améliorations particulièrement pratiques pour le développement d' AWS CDK applications. Par exemple, Java 9 introduit la `Map.of()` méthode (un moyen pratique de déclarer des hashmaps qui seraient écrits sous forme de littéraux d'objets). TypeScript Java 10 introduit l'inférence de type local à l'aide du `var` mot clé.

> **Note**
>
> La plupart des exemples de code présentés dans ce guide du développeur fonctionnent avec Java 8. Quelques exemples d'utilisation `Map.of()` ; ces exemples incluent des commentaires indiquant qu'ils nécessitent Java 9.

Vous pouvez utiliser n'importe quel éditeur de texte ou un IDE Java capable de lire les projets Maven pour travailler sur vos AWS CDK applications. Nous fournissons des conseils sur Eclipse dans ce

guide, mais IntelliJ IDEA et d'autres IDE peuvent importer des projets Maven et peuvent être utilisés pour développer des AWS CDK applications en Java. NetBeans

Il est possible d'écrire AWS CDK des applications dans des langages hébergés par JVM autres que Java (par exemple, Kotlin, Groovy, Clojure ou Scala), mais l'expérience n'est peut-être pas particulièrement idiomatique et nous ne sommes pas en mesure de fournir un support pour ces langages.

Rubriques

Premiers pas avec Java

Pour utiliser le AWS CDK, vous devez disposer d'un AWS compte et d'informations d'identification et avoir installé Node.js et le AWS CDK Toolkit. veuillez consulter Commencer à utiliser le AWS CDK.

AWS CDK Les applications Java nécessitent Java 8 (v1.8) ou version ultérieure. Nous recommandons Amazon Corretto, mais vous pouvez utiliser n'importe quelle distribution OpenJDK ou le JDK d'Oracle. Vous aurez également besoin d'Apache Maven 3.5 ou version ultérieure. Vous pouvez également utiliser des outils tels que Gradle, mais les squelettes d'applications générés par le AWS CDK Toolkit sont des projets Maven.

> **Note**
>
> Obsolète linguistique tierce : la version linguistique n'est prise en charge que jusqu'à sa fin de vie (EOL) partagée par le fournisseur ou la communauté et est sujette à modification avec préavis.

Création d'un projet

Vous créez un nouveau AWS CDK projet en l'appelant `cdk init` dans un répertoire vide. Utilisez l'`--language`option et spécifiez `java` :

```
mkdir my-project
cd my-project
cdk init app --language java
```

`cdk init`utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Toutefois, le nom doit sinon prendre la forme d'un identifiant Java ; par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Le projet qui en résulte inclut une référence au package `software.amazon.awscdk` Maven. Il et ses dépendances sont automatiquement installés par Maven.

Si vous utilisez un IDE, vous pouvez désormais ouvrir ou importer le projet. Dans Eclipse, par exemple, choisissez Fichier > Importer > Maven > Projets Maven existants. Assurez-vous que les paramètres du projet sont définis pour utiliser Java 8 (1.8).

Gestion des modules AWS de la bibliothèque Construct

Utilisez Maven pour installer les packages AWS Construct Library, qui font partie du groupe`software.amazon.awscdk`. La plupart des constructions se trouvent dans l'artefact`aws-cdk-lib`, qui est ajouté aux nouveaux projets Java par défaut. Les modules destinés aux services dont le support CDK de niveau supérieur est encore en cours de développement se trouvent dans des packages « expérimentaux » distincts, nommés avec une version courte (aucun ou préfixe AWS Amazon) du nom de leur service. Effectuez une recherche dans le référentiel central Maven pour trouver les noms de toutes les bibliothèques AWS CDK et AWS Construct Module.

> **Note**
>
> L'édition Java de la référence d'API CDK indique également les noms des packages.

La prise en charge de la bibliothèque AWS Construct par certains services se trouve dans plusieurs espaces de noms. Par exemple, les fonctionnalités d'Amazon Route 53 sont divisées en

`software.amazon.awscdk.route53route53-patterns`,`route53resolver`, et`route53-targets`.

Le AWS CDK package principal est importé en code Java au format`software.amazon.awscdk`. Les modules des différents services de la bibliothèque AWS Construct se trouvent sous le même nom que leur package Maven `software.amazon.awscdk.services` et sont nommés de la même manière que le nom de leur package Maven. Par exemple, l'espace de noms du module Amazon S3 est`software.amazon.awscdk.services.s3`.

Nous vous recommandons d'écrire une `import` instruction Java distincte pour chaque classe de bibliothèque de AWS construction que vous utilisez dans chacun de vos fichiers source Java, et d'éviter les importations de caractères génériques. Vous pouvez toujours utiliser le nom complet d'un type (y compris son espace de noms) sans instruction. `import`

Si votre application dépend d'un package expérimental, modifiez celle de votre projet `pom.xml` et ajoutez un nouvel `<dependency>` élément dans le `<dependencies>` conteneur. Par exemple, l'`<dependency>`élément suivant spécifie le module de bibliothèque de constructions CodeStar expérimentales :

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>codestar-alpha</artifactId>
    <version>2.0.0-alpha.10</version>
</dependency>
```

> **Tip**
>
> Si vous utilisez un IDE Java, il possède probablement des fonctionnalités permettant de gérer les dépendances de Maven. Nous vous recommandons toutefois de modifier `pom.xml` directement, sauf si vous êtes absolument certain que les fonctionnalités de l'IDE correspondent à ce que vous feriez à la main.

Gestion des dépendances dans Java

En Java, les dépendances sont spécifiées `pom.xml` et installées à l'aide de Maven. Le `<dependencies>` contenant comprend un `<dependency>` élément pour chaque emballage. Voici une section consacrée `pom.xml` à une application CDK Java typique.

```
<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**
>
> De nombreux IDE Java intègrent le support Maven et des `pom.xml` éditeurs visuels, ce qui peut vous être utile pour gérer les dépendances.

Maven ne prend pas en charge le verrouillage des dépendances. Bien qu'il soit possible de spécifier des plages de versions`pom.xml`, nous vous recommandons de toujours utiliser des versions exactes pour garantir la répétabilité de vos versions.

Maven installe automatiquement les dépendances transitives, mais il ne peut y avoir qu'une seule copie installée de chaque package. La version spécifiée le plus haut dans l'arborescence POM est sélectionnée ; les applications ont toujours le dernier mot quant à la version des packages à installer.

Maven installe ou met à jour automatiquement vos dépendances chaque fois que vous compilez (mvn compile) ou empaquetez (mvn package) votre projet. Le CDK Toolkit le fait automatiquement à chaque fois que vous l'exécutez, il n'est donc généralement pas nécessaire d'invoquer Maven manuellement.

AWS CDK expressions idiomatiques en Java

accessoires

Toutes les classes de la bibliothèque de AWS construction sont instanciées à l'aide de trois arguments : la portée dans laquelle la construction est définie (son parent dans l'arbre de construction), un identifiant, et props, un ensemble de paires clé/valeur que la construction utilise

pour configurer les ressources qu'elle crée. D'autres classes et méthodes utilisent également le modèle « bundle of attributes » pour les arguments.

En Java, les accessoires sont exprimés à l'aide du modèle Builder. Chaque type de construction possède un type d'accessoire correspondant ; par exemple, la `Bucket` construction (qui représente un compartiment Amazon S3) prend comme accessoires une instance de. `BucketProps`

La `BucketProps` classe (comme toutes les classes d'accessoires AWS Construct Library) possède une classe interne appeléeBuilder. Le `BucketProps.Builder` type propose des méthodes pour définir les différentes propriétés d'une `BucketProps` instance. Chaque méthode renvoie l'`Builder`instance, de sorte que les appels de méthode peuvent être enchaînés pour définir plusieurs propriétés. À la fin de la chaîne, vous appelez `build()` pour réellement produire l'`BucketProps`objet.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build());
```

Les constructions, ainsi que les autres classes qui prennent un objet semblable à un accessoire comme argument final, proposent un raccourci. La classe possède son propre objet qui `Builder` l'instancie ainsi que son objet props en une seule étape. De cette façon, vous n'avez pas besoin d'instancier explicitement (par exemple) à la fois `BucketProps` et a, et `Bucket` vous n'avez pas besoin d'importer le type d'accessoire.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build();
```

Lorsque vous dérivez votre propre construction à partir d'une construction existante, vous souhaiterez peut-être accepter des propriétés supplémentaires. Nous vous recommandons de suivre ces modèles de création. Cependant, cela n'est pas aussi simple que de sous-classer une classe de construction. Vous devez fournir vous-même les pièces mobiles des deux nouvelles `Builder` classes. Vous préférerez peut-être simplement que votre construction accepte un ou plusieurs arguments supplémentaires. Vous devez fournir des constructeurs supplémentaires lorsqu'un argument est facultatif.

Structures génériques

Dans certaines API, ils AWS CDK utilisent des JavaScript tableaux ou des objets non typés comme entrée dans une méthode. (Voir, par exemple, AWS CodeBuild la `BuildSpec.fromObject()`méthode.) En Java, ces objets sont représentés par`java.util.Map<String, Object>`. Dans les cas où les valeurs sont toutes des chaînes, vous pouvez utiliser`Map<String, String>`.

Java ne permet pas d'écrire des littéraux pour de tels conteneurs comme le font d'autres langages. Dans Java 9 et versions ultérieures, vous pouvez facilement `java.util.Map.of()`définir des cartes comportant jusqu'à dix entrées en ligne avec l'un de ces appels.

```
java.util.Map.of(
    "base-directory", "dist",
    "files", "LambdaStack.template.json"
 )
```

Pour créer des cartes comportant plus de dix entrées, utilisez `java.util.Map.ofEntries()`.

Si vous utilisez Java 8, vous pouvez fournir vos propres méthodes similaires à celles-ci.

JavaScript les tableaux sont représentés sous la forme `List<Object>` ou `List<String>` en Java. La méthode `java.util.Arrays.asList` est pratique pour définir `List` un s court.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

Valeurs manquantes

En Java, les valeurs manquantes dans AWS CDK des objets tels que les accessoires sont représentées par`null`. Vous devez tester explicitement toute valeur qui pourrait l'être `null` pour vous assurer qu'elle contient une valeur avant de faire quoi que ce soit avec elle. Java n'a pas de « sucre syntaxique » pour aider à gérer les valeurs nulles comme le font certains autres langages. Apache ObjectUtil peut vous être firstNonNullutile dans certaines situations. defaultIfNull Vous pouvez également écrire vos propres méthodes d'assistance statiques pour faciliter la gestion des valeurs potentiellement nulles et rendre votre code plus lisible.

Création, synthèse et déploiement

Compile AWS CDK automatiquement votre application avant de l'exécuter. Cependant, il peut être utile de créer votre application manuellement pour vérifier les erreurs et exécuter des tests. Vous

pouvez le faire dans votre IDE (par exemple, appuyez sur Ctrl-B dans Eclipse) ou en lançant `mvn compile` une invite de commande dans le répertoire racine de votre projet.

Exécutez tous les tests que vous avez écrits `mvn test` à l'aide d'une invite de commande.

Les piles définies dans votre AWS CDK application peuvent être synthétisées et déployées individuellement ou ensemble à l'aide des commandes ci-dessous. En général, vous devez vous trouver dans le répertoire principal de votre projet lorsque vous les publiez.

- `cdk synth`: synthétise un AWS CloudFormation modèle à partir d'une ou de plusieurs piles de votre AWS CDK application.
- `cdk deploy`: Déploie les ressources définies par une ou plusieurs piles de votre AWS CDK application vers. AWS

Vous pouvez spécifier les noms de plusieurs piles à synthétiser ou à déployer en une seule commande. Si votre application ne définit qu'une seule pile, il n'est pas nécessaire de la spécifier.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Vous pouvez également utiliser les caractères génériques * (n'importe quel nombre de caractères) et ? (n'importe quel caractère) pour identifier les piles par modèle. Lorsque vous utilisez des caractères génériques, placez le motif entre guillemets. Sinon, le shell peut essayer de l'étendre aux noms des fichiers du répertoire actuel avant qu'ils ne soient transmis au AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Il n'est pas nécessaire de synthétiser explicitement les piles avant de les déployer. `cdk deploy` Effectuez cette étape pour vous assurer que votre dernier code est déployé.

Pour une documentation complète de la `cdk` commande, consultezthe section called “AWS CDK Boîte à outils”.

Travailler avec le AWS CDK en C#

.NET est un langage client entièrement pris en charge AWS CDK et est considéré comme stable. C# est le principal langage .NET pour lequel nous fournissons des exemples et du support. Vous pouvez choisir d'écrire AWS CDK des applications dans d'autres langages .NET, tels que Visual Basic ou F#, mais la prise en charge AWS de l'utilisation de ces langages avec le CDK est limitée.

Vous pouvez développer des AWS CDK applications en C# à l'aide d'outils familiers tels que Visual Studio, Visual Studio Code, la `dotnet` commande et le gestionnaire de NuGet packages. Les modules composant la bibliothèque AWS Construct sont distribués via nuget.org.

Nous vous conseillons d'utiliser Visual Studio 2019 (n'importe quelle édition) sous Windows pour développer des AWS CDK applications en C#.

Rubriques

Premiers pas avec C#

Pour utiliser le AWS CDK, vous devez disposer d'un AWS compte et d'informations d'identification et avoir installé Node.js et le AWS CDK Toolkit. veuillez consulter Commencer à utiliser le AWS CDK.

AWS CDK Les applications C# nécessitent .NET Core v3.1 ou version ultérieure, disponible ici.

La chaîne d'outils .NET inclut un outil `dotnet` en ligne de commande permettant de créer et d'exécuter des applications .NET et de gérer des packages. NuGet Même si vous travaillez principalement dans Visual Studio, cette commande peut être utile pour les opérations par lots et pour installer les packages AWS Construct Library.

Création d’un projet

Vous créez un nouveau AWS CDK projet en l'appelant `cdk init` dans un répertoire vide. Utilisez l'`--language`option et spécifiez `csharp` :

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init`utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Toutefois, le nom doit sinon prendre la forme d'un identifiant C# ; par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Le projet qui en résulte inclut une référence au `Amazon.CDK.Lib` NuGet package. Il et ses dépendances sont installés automatiquement par NuGet.

Gestion des modules AWS de la bibliothèque Construct

L'écosystème .NET utilise le gestionnaire de NuGet packages. Le package CDK principal, qui contient les classes de base et toutes les constructions de service stables, est. `Amazon.CDK.Lib` Les modules expérimentaux, dans lesquels de nouvelles fonctionnalités sont en cours de développement actif`Amazon.CDK.AWS.`*`SERVICE-NAME`*`.Alpha`, sont nommés comme suit : le nom du service est un AWS nom court sans préfixe Amazon. Par exemple, le nom NuGet du package pour le AWS IoT module est`Amazon.CDK.AWS.IoT.Alpha`. Si vous ne trouvez pas le package que vous recherchez, recherchez NuGet.org.

> **Note**
>
> L'édition .NET du CDK API Reference indique également les noms des packages.

La prise en charge de la bibliothèque AWS Construct par certains services se trouve dans plusieurs modules. Par exemple, AWS IoT possède un deuxième module nommé`Amazon.CDK.AWS.IoT.Actions.Alpha`.

Le AWS CDK module principal, dont vous aurez besoin dans la plupart des AWS CDK applications, est importé en code C# sous `Amazon.CDK` le nom de. Les modules des différents services de la bibliothèque AWS Construct se trouvent sous cette rubrique`Amazon.CDK.AWS`. Par exemple, l'espace de noms du module Amazon S3 est`Amazon.CDK.AWS.S3`.

Nous vous recommandons d'écrire des `using` directives C# pour les constructions de base du CDK et pour chaque AWS service que vous utilisez dans chacun de vos fichiers source C#. Il peut être pratique d'utiliser un alias pour un espace de noms ou un type afin de résoudre les conflits de

noms. Vous pouvez toujours utiliser le nom complet d'un type (y compris son espace de noms) sans instruction. `using`

Gestion des dépendances dans C#

Dans les AWS CDK applications C#, vous gérez les dépendances à l'aide NuGet de. NuGet possède quatre interfaces standard, pour la plupart équivalentes. Utilisez celui qui correspond à vos besoins et à votre style de travail. Vous pouvez également utiliser des outils compatibles, tels que Paket MyGetou même modifier directement le `.csproj` fichier.

NuGet ne vous permet pas de spécifier des plages de versions pour les dépendances. Chaque dépendance est associée à une version spécifique.

Après avoir mis à jour vos dépendances, Visual Studio récupérera les versions spécifiées de chaque package lors de la prochaine génération. NuGet Si vous n'utilisez pas Visual Studio, utilisez la dotnet restore commande pour mettre à jour vos dépendances.

Modification directe du fichier de projet

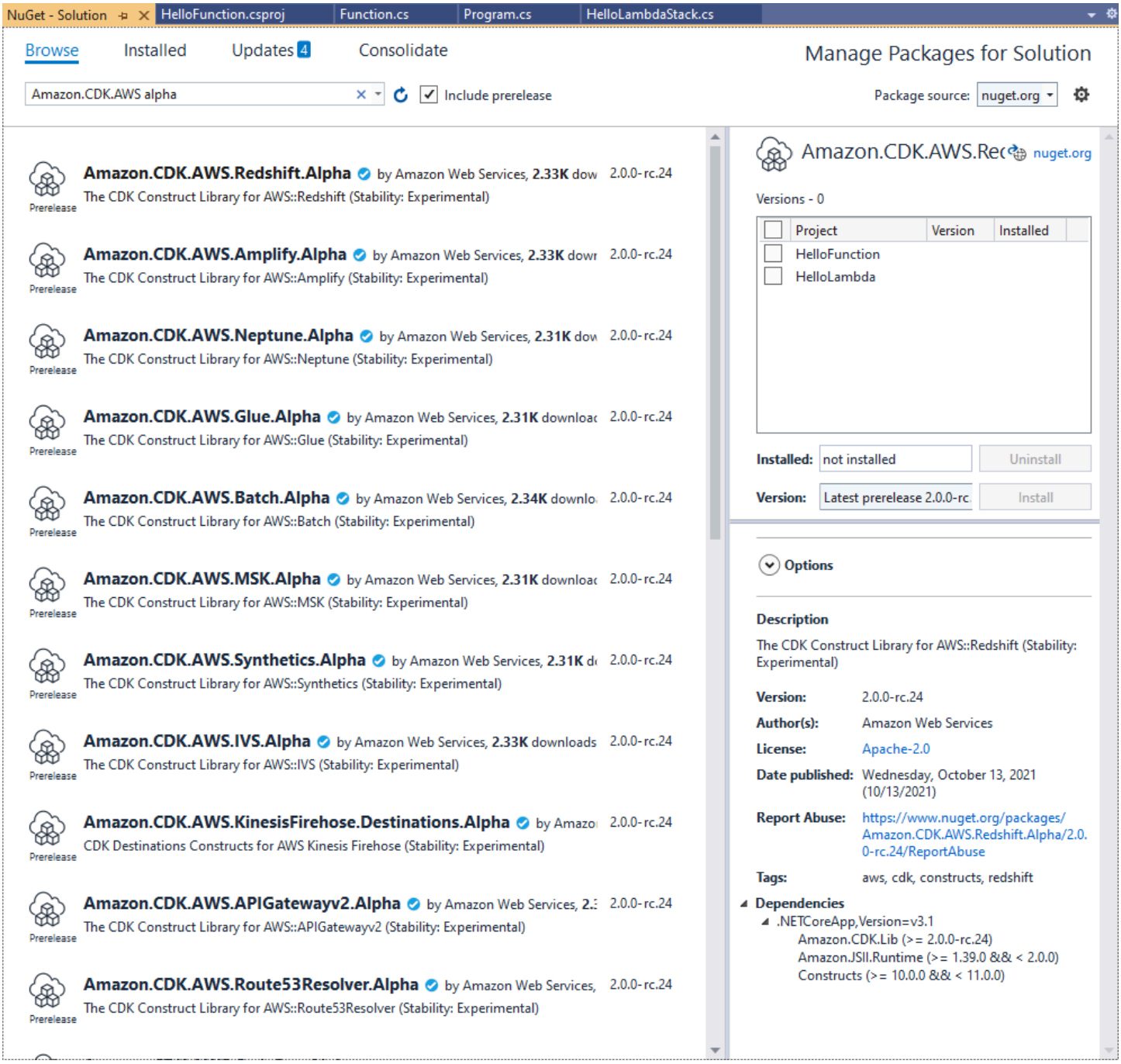
Le `.csproj` fichier de votre projet contient un `<ItemGroup>` conteneur qui répertorie vos dépendances sous forme d'`<PackageReference>`éléments.

```
<ItemGroup>
    <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
    <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

L' NuGet interface graphique de Visual Studio

Les NuGet outils de Visual Studio sont accessibles depuis Outils > Gestionnaire de NuGet packages > Gérer les NuGet packages pour la solution. Utilisez l'onglet Parcourir pour rechercher les packages AWS Construct Library que vous souhaitez installer. Vous pouvez choisir la version souhaitée, y compris les versions préliminaires de vos modules, et les ajouter à tous les projets ouverts.

> **Note**
>
> Tous les modules de AWS Construct Library considérés comme « expérimentaux » (voirthe section called “Gestion des versions”) sont marqués comme préversions NuGet et ont un `alpha` suffixe de nom.



Consultez la page Mises à jour pour installer les nouvelles versions de vos packages.

La NuGet console

La NuGet console est une interface PowerShell basée NuGet qui fonctionne dans le contexte d'un projet Visual Studio. Vous pouvez l'ouvrir dans Visual Studio en choisissant Outils > Gestionnaire de NuGet packages > Console du gestionnaire de packages. Pour plus d'informations sur l'utilisation

de cet outil, voir Installer et gérer des packages à l'aide de la console Package Manager dans Visual Studio.

La **dotnet** commande

La `dotnet` commande est le principal outil de ligne de commande permettant de travailler avec des projets Visual Studio C#. Vous pouvez l'invoquer à partir de n'importe quelle invite de commande Windows. Parmi ses nombreuses fonctionnalités, `dotnet` vous pouvez ajouter NuGet des dépendances à un projet Visual Studio.

En supposant que vous vous trouvez dans le même répertoire que le fichier de projet Visual Studio (`.csproj`), exécutez une commande comme celle-ci pour installer un package. Comme la bibliothèque CDK principale est incluse lorsque vous créez un projet, il vous suffit d'installer explicitement des modules expérimentaux. Les modules expérimentaux nécessitent que vous spécifiiez un numéro de version explicite.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Vous pouvez exécuter la commande depuis un autre répertoire. Pour ce faire, incluez le chemin du fichier de projet ou du répertoire qui le contient après le `add` mot clé. L'exemple suivant suppose que vous vous trouvez dans le répertoire principal de votre AWS CDK projet.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Pour installer une version spécifique d'un package, incluez le `-v` drapeau et la version souhaitée.

Pour mettre à jour un package, exécutez la même `dotnet add` commande que celle que vous avez utilisée pour l'installer. Pour les modules expérimentaux, vous devez à nouveau spécifier un numéro de version explicite.

Pour plus d'informations sur la gestion des packages à l'aide de la `dotnet` commande, voir Installer et gérer les packages à l'aide de la CLI dotnet.

La **nuget** commande

L'outil de ligne de `nuget` commande peut installer et mettre à jour NuGet des packages. Cependant, cela nécessite que votre projet Visual Studio soit configuré différemment de la manière dont `cdk init` les projets sont configurés. (Détails techniques : `nuget` fonctionne avec des `Packages.config` projets, tout en `cdk init` créant un projet de style plus récent`PackageReference`.)

Nous ne recommandons pas l'utilisation de l'`nuget`outil avec AWS CDK des projets créés par`cdk init`. Si vous utilisez un autre type de projet et que vous souhaitez l'utiliser`nuget`, consultez la référence de la NuGet CLI.

AWS CDK expressions idiomatiques en C#

accessoires

Toutes les classes de la bibliothèque de AWS construction sont instanciées à l'aide de trois arguments : la portée dans laquelle la construction est définie (son parent dans l'arbre de construction), un identifiant, et props, un ensemble de paires clé/valeur que la construction utilise pour configurer les ressources qu'elle crée. D'autres classes et méthodes utilisent également le modèle « bundle of attributes » pour les arguments.

En C#, les accessoires sont exprimés à l'aide d'un type d'accessoire. En langage C# idiomatique, nous pouvons utiliser un initialiseur d'objet pour définir les différentes propriétés. Nous créons ici un compartiment Amazon S3 à l'aide de cette `Bucket` construction ; le type d'accessoire correspondant est`BucketProps`.

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
    Versioned = true
});
```

> **Tip**
>
> Ajoutez le package `Amazon.JSII.Analyzers` à votre projet pour vérifier les valeurs requises dans vos définitions d'accessoires dans Visual Studio.

Lorsque vous étendez une classe ou que vous remplacez une méthode, vous souhaiterez peut-être accepter des accessoires supplémentaires pour vos propres besoins qui ne sont pas compris par la classe parent. Pour ce faire, sous-classez le type d'accessoire approprié et ajoutez les nouveaux attributs.

```
// 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"
});
```

Lorsque vous appelez l'initialiseur ou la méthode remplacée de la classe parent, vous pouvez généralement transmettre les accessoires que vous avez reçus. Le nouveau type est compatible avec son parent et les accessoires supplémentaires que vous avez ajoutés sont ignorés.

Une future version du AWS CDK pourrait par hasard ajouter une nouvelle propriété avec un nom que vous avez utilisé pour votre propre propriété. Cela ne posera aucun problème technique lors de l'utilisation de votre construction ou de votre méthode (étant donné que votre propriété n'est pas transmise « en haut de la chaîne », la classe parent ou la méthode remplacée utilisera simplement une valeur par défaut) mais cela peut être source de confusion pour les utilisateurs de votre construction. Vous pouvez éviter ce problème potentiel en nommant vos propriétés de manière à ce qu'elles appartiennent clairement à votre construction. S'il existe de nombreuses nouvelles propriétés, regroupez-les dans une classe portant le nom approprié et transmettez-les en tant que propriété unique.

Structures génériques

Dans certaines API, ils AWS CDK utilisent des JavaScript tableaux ou des objets non typés comme entrée dans une méthode. (Voir, par exemple, AWS CodeBuild la `BuildSpec.fromObject()`méthode.) En C#, ces objets sont représentés par. `System.Collections.Generic.Dictionary<String, Object>` Dans les cas où les valeurs sont toutes des chaînes, vous pouvez utiliser`Dictionary<String, String>`. JavaScript les tableaux sont représentés sous forme `object[]` de types de `string[]` tableaux en C#.

> **Tip**
>
> Vous pouvez définir des alias courts pour faciliter l'utilisation de ces types de dictionnaires spécifiques.

```
using StringDict = System.Collections.Generic.Dictionary<string, string>;
using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
```

Valeurs manquantes

En C#, les valeurs manquantes dans les AWS CDK objets tels que les accessoires sont représentées par. `null` L'opérateur d'accès conditionnel nul aux membres `?.` et l'opérateur de coalescence nul `??` sont pratiques pour travailler avec ces valeurs.

```
// 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";
```

Création, synthèse et déploiement

Compile AWS CDK automatiquement votre application avant de l'exécuter. Cependant, il peut être utile de créer votre application manuellement pour vérifier les erreurs et exécuter des tests. Vous pouvez le faire en appuyant sur F6 dans Visual Studio ou en `dotnet build src` indiquant à partir de la ligne de `src` commande le répertoire de votre projet qui contient le fichier Visual Studio Solution (`.sln`).

Les piles définies dans votre AWS CDK application peuvent être synthétisées et déployées individuellement ou ensemble à l'aide des commandes ci-dessous. En général, vous devez vous trouver dans le répertoire principal de votre projet lorsque vous les publiez.

- `cdk synth`: synthétise un AWS CloudFormation modèle à partir d'une ou de plusieurs piles de votre AWS CDK application.
- `cdk deploy`: Déploie les ressources définies par une ou plusieurs piles de votre AWS CDK application vers. AWS

Vous pouvez spécifier les noms de plusieurs piles à synthétiser ou à déployer en une seule commande. Si votre application ne définit qu'une seule pile, il n'est pas nécessaire de la spécifier.

```
cdk synth                # app defines single stack
```

```
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Vous pouvez également utiliser les caractères génériques * (n'importe quel nombre de caractères) et ? (n'importe quel caractère) pour identifier les piles par modèle. Lorsque vous utilisez des caractères génériques, placez le motif entre guillemets. Sinon, le shell peut essayer de l'étendre aux noms des fichiers du répertoire actuel avant qu'ils ne soient transmis au AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Il n'est pas nécessaire de synthétiser explicitement les piles avant de les déployer. `cdk deploy` Effectuez cette étape pour vous assurer que votre dernier code est déployé.

Pour une documentation complète de la `cdk` commande, consultezthe section called “AWS CDK Boîte à outils”.

Travailler avec le AWS CDK in Go

Go est un langage client entièrement pris en charge pour le AWS Cloud Development Kit (AWS CDK) et est considéré comme stable. L'utilisation d' AWS CDK in Go utilise des outils familiers. La version Go de l'événement utilise AWS CDK des identifiants de type Go.

Contrairement aux autres langages pris en charge par le CDK, Go n'est pas un langage de programmation orienté objet traditionnel. Go utilise la composition alors que d'autres langages exploitent souvent l'héritage. Nous avons essayé d'utiliser autant que possible des approches de Go idiomatiques, mais il y a des endroits où le CDK peut différer.

Cette rubrique fournit des conseils lorsque vous travaillez avec le AWS CDK in Go. Consultez le billet de blog consacré à l'annonce pour une présentation d'un projet Go simple pour le AWS CDK.

Rubriques

- Premiers pas avec Go
- Création d’un projet
- Gestion des modules AWS de la bibliothèque Construct

- Gestion des dépendances dans Go
- AWS CDK Expressions idiomatiques en Go
- Création, synthèse et déploiement

Premiers pas avec Go

Pour utiliser le AWS CDK, vous devez disposer d'un AWS compte et d'informations d'identification et avoir installé Node.js et le AWS CDK Toolkit. veuillez consulter Commencer à utiliser le AWS CDK.

Les liaisons Go pour AWS CDK utiliser la chaîne d'outils Go standard, v1.18 ou ultérieure. Vous pouvez utiliser l'éditeur de votre choix.

> **Note**
>
> Obsolète linguistique tierce : la version linguistique n'est prise en charge que jusqu'à sa fin de vie (EOL) partagée par le fournisseur ou la communauté et est sujette à modification avec préavis.

Création d'un projet

Vous créez un nouveau AWS CDK projet en l'invoquant `cdk init` dans un répertoire vide. Utilisez l'`--language`option et spécifiez go :

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init`utilise le nom du dossier du projet pour nommer les différents éléments du projet, notamment les classes, les sous-dossiers et les fichiers. Les traits d'union figurant dans le nom du dossier sont convertis en traits de soulignement. Toutefois, le nom doit sinon prendre la forme d'un identifiant Go ; par exemple, il ne doit pas commencer par un chiffre ni contenir d'espaces.

Le projet qui en résulte inclut une référence au module AWS CDK Go de base,`github.com/aws/aws-cdk-go/awscdk/v2`, dans`go.mod`. Problème lors go get de l'installation de ce module et des autres modules requis.

Gestion des modules AWS de la bibliothèque Construct

Dans la plupart des AWS CDK documentations et des exemples, le mot « module » est souvent utilisé pour désigner les modules de la bibliothèque AWS Construct, un ou plusieurs par AWS service, ce qui diffère de l'utilisation idiomatique du terme par Go. La bibliothèque CDK Construct est fournie dans un module Go avec les modules individuels de la bibliothèque Construct, qui prennent en charge les différents AWS services, fournis sous forme de packages Go dans ce module.

La prise en charge de la bibliothèque de AWS construction par certains services se trouve dans plusieurs modules de bibliothèque de construction (package Go). Par exemple, Amazon Route 53 possède trois modules Construct Library en plus `awsroute53` du package principal`awsroute53patterns`, nommé`awsroute53resolver`, et`awsroute53targets`.

Le package AWS CDK de base, dont vous aurez besoin dans la plupart des AWS CDK applications, est importé dans le code Go sous le nom de`github.com/aws/aws-cdk-go/awscdk/v2`. Des packages pour les différents services de la bibliothèque AWS Construct se trouvent sous cette rubrique`github.com/aws/aws-cdk-go/awscdk/v2`. Par exemple, l'espace de noms du module Amazon S3 est`github.com/aws/aws-cdk-go/awscdk/v2/awss3`.

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

Une fois que vous avez importé les modules Construct Library (packages Go) pour les services que vous souhaitez utiliser dans votre application, vous accédez aux constructions de ce module en utilisant, par exemple,`awss3.Bucket`.

Gestion des dépendances dans Go

Dans Go, les versions des dépendances sont définies dans`go.mod`. La valeur par défaut `go.mod` est similaire à celle présentée ici.

```
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
)
```

Les noms des packages (modules, dans le langage Go) sont spécifiés par URL avec le numéro de version requis ajouté. Le système de modules de Go ne prend pas en charge les plages de versions.

Émettez la go get commande pour installer tous les modules requis et les mettre à jourgo.mod. Pour voir la liste des mises à jour disponibles pour vos dépendances, publiezgo list -m -u all.

AWS CDK Expressions idiomatiques en Go

Noms des champs et des méthodes

Les noms de champs et de méthodes utilisent camel casing (likeThis) dans TypeScript, la langue d'origine du CDK. Dans Go, ils suivent les conventions de Go, tout comme Pascal-cased (). LikeThis

Nettoyage

Dans votre main méthode, utilisez-le defer jsii.Close() pour vous assurer que votre application CDK se nettoie d'elle-même.

Valeurs manquantes et conversion du pointeur

Dans Go, les valeurs manquantes dans AWS CDK des objets tels que les ensembles de propriétés sont représentées parnil. Go n'a pas de types nullables ; le seul type qui peut contenir nil est un pointeur. Pour permettre aux valeurs d'être facultatives, toutes les propriétés, tous les arguments et toutes les valeurs de retour du CDK sont des pointeurs, même pour les types primitifs. Cela s'applique aux valeurs obligatoires ainsi qu'aux valeurs facultatives. Ainsi, si une valeur requise devient facultative ultérieurement, aucun changement radical de type n'est nécessaire.

Lorsque vous transmettez des valeurs ou des expressions littérales, utilisez les fonctions d'assistance suivantes pour créer des pointeurs vers les valeurs.

- jsii.String
- jsii.Number
- jsii.Bool

- `jsii.Time`

Pour des raisons de cohérence, nous vous recommandons d'utiliser des pointeurs de la même manière lorsque vous définissez vos propres constructions, même s'il peut sembler plus pratique, par exemple, de recevoir vos constructions `id` sous forme de chaîne plutôt que de pointeur vers une chaîne.

Lorsque vous manipulez des AWS CDK valeurs facultatives, y compris des valeurs primitives ou des types complexes, vous devez tester explicitement les pointeurs pour vous assurer qu'ils ne le sont pas `nil` avant de faire quoi que ce soit avec eux. Go n'a pas de « sucre syntaxique » pour aider à gérer les valeurs vides ou manquantes comme le font d'autres langages. Cependant, l'existence des valeurs requises dans les ensembles de propriétés et les structures similaires est garantie (sinon la construction échoue), il n'est donc pas nécessaire de vérifier ces valeurs. `nil`

Constructions et accessoires

Les constructions, qui représentent une ou plusieurs AWS ressources et leurs attributs associés, sont représentées dans Go sous forme d'interfaces. Par exemple, `awss3.Bucket` c'est une interface. Chaque construction possède une fonction d'usine, telle que `awss3.NewBucket` celle de renvoyer une structure qui implémente l'interface correspondante.

Toutes les fonctions d'usine prennent trois arguments : celui `scope` dans lequel la construction est définie (son parent dans l'arbre de construction), un `idprops`, et un ensemble de paires clé/valeur que la construction utilise pour configurer les ressources qu'elle crée. Le modèle « ensemble d'attributs » est également utilisé ailleurs dans le AWS CDK.

Dans Go, les accessoires sont représentés par un type de structure spécifique pour chaque construction. Par exemple, an `awss3.Bucket` prend un argument props de type`awss3.BucketProps`. Utilisez une structure littérale pour écrire les arguments des accessoires.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

Structures génériques

À certains endroits, il AWS CDK utilise des JavaScript tableaux ou des objets non typés comme entrée dans une méthode. (Voir, par exemple, AWS CodeBuild la

`BuildSpec.fromObject()`méthode.) Dans Go, ces objets sont respectivement représentés sous forme de tranches et d'interface vide.

Le CDK fournit des fonctions d'assistance variadiques, par exemple `jsii.Strings` pour créer des tranches contenant des types primitifs.

```
jsii.Strings("One", "Two", "Three")
```

Développement de constructions personnalisées

Dans Go, il est généralement plus simple d'écrire une nouvelle construction que d'étendre une construction existante. Définissez d'abord un nouveau type de structure, en incorporant de manière anonyme un ou plusieurs types existants si vous souhaitez utiliser une sémantique semblable à une extension. Écrivez des méthodes pour chaque nouvelle fonctionnalité que vous ajoutez et les champs nécessaires pour contenir les données dont ils ont besoin. Définissez une interface d'accessoires si votre construction en a besoin. Enfin, écrivez une fonction d'usine `NewMyConstruct()` pour renvoyer une instance de votre construction.

Si vous modifiez simplement certaines valeurs par défaut d'une construction existante ou si vous ajoutez un comportement simple lors de l'instanciation, vous n'avez pas besoin de toute cette plomberie. Écrivez plutôt une fonction d'usine qui appelle la fonction d'usine de la construction que vous « étendez ». Dans d'autres langages CDK, par exemple, vous pouvez créer une `TypedBucket` construction qui impose le type d'objets dans un compartiment Amazon S3 en remplaçant le `s3.Bucket` type et, dans l'initialiseur de votre nouveau type, en ajoutant une politique de compartiment autorisant uniquement l'ajout d'extensions de nom de fichier spécifiées au compartiment. Dans Go, il est plus facile de simplement écrire un `NewTypedBucket` qui renvoie un `s3.Bucket` (instancié à l'aides3.NewBucket) auquel vous avez ajouté une politique de compartiment appropriée. Aucun nouveau type de construction n'est nécessaire car la fonctionnalité est déjà disponible dans la construction standard du bucket ; la nouvelle « construction » fournit simplement un moyen plus simple de la configurer.

Création, synthèse et déploiement

Compile AWS CDK automatiquement votre application avant de l'exécuter. Cependant, il peut être utile de créer votre application manuellement pour vérifier les erreurs et exécuter des tests. Vous pouvez le faire `go build` en lançant une commande dans le répertoire racine de votre projet.

Exécutez tous les tests que vous avez écrits `go test` à l'aide d'une invite de commande.

Les piles définies dans votre AWS CDK application peuvent être synthétisées et déployées individuellement ou ensemble à l'aide des commandes ci-dessous. En général, vous devez vous trouver dans le répertoire principal de votre projet lorsque vous les publiez.

- `cdk synth`: synthétise un AWS CloudFormation modèle à partir d'une ou de plusieurs piles de votre AWS CDK application.
- `cdk deploy`: Déploie les ressources définies par une ou plusieurs piles de votre AWS CDK application vers. AWS

Vous pouvez spécifier les noms de plusieurs piles à synthétiser ou à déployer en une seule commande. Si votre application ne définit qu'une seule pile, il n'est pas nécessaire de la spécifier.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

Vous pouvez également utiliser les caractères génériques * (n'importe quel nombre de caractères) et ? (n'importe quel caractère) pour identifier les piles par modèle. Lorsque vous utilisez des caractères génériques, placez le motif entre guillemets. Sinon, le shell peut essayer de l'étendre aux noms des fichiers du répertoire actuel avant qu'ils ne soient transmis au AWS CDK Toolkit.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> Il n'est pas nécessaire de synthétiser explicitement les piles avant de les déployer. `cdk deploy` Effectuez cette étape pour vous assurer que votre dernier code est déployé.

Pour une documentation complète de la `cdk` commande, consultezthe section called “AWS CDK Boîte à outils”.

Développement d' AWS CDK applications

Développez AWS Cloud Development Kit (AWS CDK) des applications.

Rubriques

Personnalisation des constructions à partir de la AWS bibliothèque de constructions

Personnalisez les constructions de la bibliothèque de constructions AWS à l'aide de trappes d'évacuation, de remplacements bruts et de ressources personnalisées.

Rubriques

Utilisation de trappes d'évacuation

La AWS bibliothèque de constructions fournit des constructions de différents niveaux d'abstraction.

Au plus haut niveau, votre AWS CDK application et les éléments qu'elle contient sont eux-mêmes des abstractions de l'ensemble de votre infrastructure cloud, ou de parties importantes de celle-ci.

Ils peuvent être paramétrés pour les déployer dans différents environnements ou pour différents besoins.

Les abstractions sont de puissants outils de conception et de mise en œuvre d'applications cloud. AWS CDK Cela vous donne le pouvoir non seulement de construire avec ses abstractions, mais également de créer de nouvelles abstractions. En utilisant les structures L2 et L3 open source existantes comme guide, vous pouvez créer vos propres structures L2 et L3 afin de refléter les meilleures pratiques et opinions de votre propre organisation.

Aucune abstraction n'est parfaite, et même de bonnes abstractions ne peuvent pas couvrir tous les cas d'utilisation possibles. Au cours du développement, il se peut que vous trouviez une construction qui correspond presque à vos besoins, nécessitant une petite ou une grande personnalisation.

Pour cette raison, il AWS CDK fournit des moyens de sortir du modèle de construction. Cela inclut le passage à une abstraction de niveau inférieur ou à un modèle complètement différent. Les trappes d'évacuation vous permettent d'échapper au AWS CDK paradigme et de le personnaliser en fonction de vos besoins. Vous pouvez ensuite intégrer vos modifications dans une nouvelle construction pour éliminer la complexité sous-jacente et fournir une API propre aux autres développeurs.

Voici des exemples de situations dans lesquelles vous pouvez utiliser des trappes d'évacuation :

- Une fonctionnalité AWS de service est disponible via AWS CloudFormation, mais il n'existe aucune construction L2 pour elle.
- Une fonctionnalité de AWS service est disponible via AWS CloudFormation, et il existe des constructions L2 pour le service, mais celles-ci n'exposent pas encore la fonctionnalité. Comme les constructions L2 sont conçues par l'équipe du CDK, il est possible qu'elles ne soient pas immédiatement disponibles pour les nouvelles fonctionnalités.
- La fonctionnalité n'est pas encore disponible AWS CloudFormation du tout.

  Pour déterminer si une fonctionnalité est disponible via AWS CloudFormation, consultez la section Référence des types de AWS ressources et de propriétés.

Développez des trappes d'évacuation pour les constructions L1

Si les constructions L2 ne sont pas disponibles pour le service, vous pouvez utiliser les constructions L1 générées automatiquement. Ces ressources peuvent être reconnues par leur nom commençant par`Cfn`, tel que `CfnBucket` ou`CfnRole`. Vous les instanciez exactement comme vous utiliseriez la ressource équivalente AWS CloudFormation .

Par exemple, pour instancier un bucket Amazon S3 L1 de bas niveau avec l'analyse activée, vous devez écrire quelque chose comme ceci.

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",
        // ...
    }
});
```

Dans de rares cas, il peut arriver que vous souhaitiez définir une ressource qui n'a pas de `CfnXxx` classe correspondante. Il peut s'agir d'un nouveau type de ressource qui n'a pas encore été publié dans la spécification de la AWS CloudFormation ressource. Dans de tels cas, vous pouvez les instancier `cdk.CfnResource` directement et spécifier le type et les propriétés de la ressource. Voici un exemple :

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"
            }
```

```
        }
    }
});
```

Développez des trappes d'évacuation pour les constructions L2

S'il manque une fonctionnalité à une construction L2 ou si vous essayez de contourner un problème, vous pouvez modifier la construction L1 encapsulée par la construction L2.

Toutes les constructions L2 contiennent la construction L1 correspondante. Par exemple, la construction de haut niveau enveloppe la `Bucket` construction de bas niveau. `CfnBucket` Comme il `CfnBucket` correspond directement à la AWS CloudFormation ressource, il expose toutes les fonctionnalités disponibles via AWS CloudFormation.

L'approche de base pour accéder à la construction L1 consiste à utiliser `construct.node.defaultChild` (Python :`default_child`), à la convertir dans le bon type (si nécessaire) et à modifier ses propriétés. Encore une fois, prenons l'exemple d'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",
        // ...
    }
};
```

Vous pouvez également utiliser cet objet pour modifier AWS CloudFormation des options telles que `Metadata` et`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"
};
```

Trappes UN-Escape

AWS CDK Il permet également d'augmenter un niveau d'abstraction, ce que nous pourrions appeler une trappe « unescape ». Si vous avez une construction L1, par exemple`CfnBucket`, vous pouvez créer une nouvelle construction L2 (`Bucket`dans ce cas) pour encapsuler la construction L1.

C'est pratique lorsque vous créez une ressource L1 mais que vous souhaitez l'utiliser avec une construction qui nécessite une ressource L2. C'est également utile lorsque vous souhaitez utiliser des

méthodes pratiques telles `.grantXxxx()` que celles qui ne sont pas disponibles sur la construction L1.

Vous passez au niveau d'abstraction supérieur à l'aide d'une méthode statique sur la classe L2 appelée, par exemple, `.fromCfnXxxx()` `Bucket.fromCfnBucket()` pour les buckets Amazon S3. La ressource L1 est le seul paramètre.

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);
```

Les constructions L2 créées à partir des constructions L1 sont des objets proxy qui font référence à la ressource L1, similaires à ceux créés à partir de noms de ressources, d'ARN ou de recherches. Les modifications apportées à ces constructions n'affectent pas le AWS CloudFormation modèle synthétisé final (puisque vous disposez de la ressource L1, vous pouvez toutefois la modifier à la

place). Pour plus d'informations sur les objets proxy, consultezthe section called “Référencement des ressources de votre compte AWS”.

Pour éviter toute confusion, ne créez pas plusieurs constructions L2 faisant référence à la même construction L1. Par exemple, si vous extrayez le fichier `CfnBucket` d'un `Bucket` en utilisant la technique décrite dans la section précédente, vous ne devez pas créer une deuxième `Bucket` instance en appelant `Bucket.fromCfnBucket()` avec cette techniqueCfnBucket. Cela fonctionne réellement comme prévu (un seul `AWS::S3::Bucket` est synthétisé) mais cela rend votre code plus difficile à maintenir.

Dérogations brutes

Si certaines propriétés sont absentes de la construction L1, vous pouvez éviter toute saisie en utilisant des remplacements bruts. Cela permet également de supprimer des propriétés synthétisées.

Utilisez l'une des `addOverride` méthodes (Python :`add_override`), comme indiqué dans l'exemple suivant.

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");
```

Ressources personnalisées

Si la fonctionnalité n'est pas disponible via un appel d'API direct AWS CloudFormation, mais uniquement via un appel d'API, vous devez écrire une ressource AWS CloudFormation personnalisée pour effectuer l'appel d'API dont vous avez besoin. Vous pouvez utiliser le AWS CDK pour écrire des ressources personnalisées et les intégrer dans une interface de construction normale. Du point de vue d'un consommateur de votre produit, l'expérience semblera native.

La création d'une ressource personnalisée implique l'écriture d'une fonction Lambda qui répond aux événements liés à une ressource et à son `DELETE` cycle de `CREATE` vie. `UPDATE` Si votre ressource personnalisée ne doit effectuer qu'un seul appel d'API, pensez à utiliser le AwsCustomResource. Cela permet d'effectuer des appels au SDK arbitraires lors d'un AWS CloudFormation déploiement. Sinon, vous devez écrire votre propre fonction Lambda pour effectuer le travail dont vous avez besoin.

Le sujet est trop vaste pour être traité dans son intégralité ici, mais les liens suivants devraient vous aider à démarrer :

- Personnalisation des ressources
- Exemple de ressource personnalisée
- Pour un exemple plus complet, consultez la DnsValidatedCertificateclasse dans la bibliothèque standard du CDK. Ceci est implémenté en tant que ressource personnalisée.

Obtenir une valeur à partir d'une variable d'environnement

Pour obtenir la valeur d'une variable d'environnement, utilisez le code suivant. Ce code obtient la valeur de la variable d'environnement`MYBUCKET`.

TypeScript

```
// Sets bucket_name to undefined if environment variable not set
var bucket_name = process.env.MYBUCKET;

// Sets bucket_name to a default if env var doesn't exist
var bucket_name = process.env.MYBUCKET || "DefaultName";
```

JavaScript

```
// Sets bucket_name to undefined if environment variable not set
var bucket_name = process.env.MYBUCKET;

// Sets bucket_name to a default if env var doesn't exist
var bucket_name = process.env.MYBUCKET || "DefaultName";
```

Python

```
import os

# Raises KeyError if environment variable doesn't exist
bucket_name = os.environ["MYBUCKET"]

# Sets bucket_name to None if environment variable doesn't exist
bucket_name = os.getenv("MYBUCKET")

# Sets bucket_name to a default if env var doesn't exist
bucket_name = os.getenv("MYBUCKET", "DefaultName")
```

Java

```
// Sets bucketName to null if environment variable doesn't exist
String bucketName = System.getenv("MYBUCKET");

// Sets bucketName to a default if env var doesn't exist
String bucketName = System.getenv("MYBUCKET");
if (bucketName == null) bucketName = "DefaultName";
```

C#

```
using System;

// Sets bucket name to null if environment variable doesn't exist
string bucketName = Environment.GetEnvironmentVariable("MYBUCKET");

// Sets bucket_name to a default if env var doesn't exist
string bucketName = Environment.GetEnvironmentVariable("MYBUCKET") ?? "DefaultName";
```

Utiliser une AWS CloudFormation valeur

the section called “Paramètres”Pour plus d'informations sur l'utilisation de AWS CloudFormation paramètres avec le AWS CDK.

Pour obtenir une référence à une ressource dans un AWS CloudFormation modèle existant, consultezthe section called “Importer un AWS CloudFormation modèle”.

Importer un AWS CloudFormation modèle existant

Importez des ressources d'un AWS CloudFormation modèle dans vos AWS Cloud Development Kit (AWS CDK) applications en utilisant la `cloudformation-include.CfnInclude`construction pour convertir les ressources en constructions L1.

Après l'importation, vous pouvez utiliser ces ressources dans votre application de la même manière que si elles étaient initialement définies dans le AWS CDK code. Vous pouvez également utiliser ces constructions L1 dans des constructions de niveau supérieur AWS CDK . Par exemple, cela peut vous permettre d'utiliser les méthodes d'octroi d'autorisations L2 avec les ressources qu'elles définissent.

La `cloudformation-include.CfnInclude` construction ajoute essentiellement un wrapper d' AWS CDK API à n'importe quelle ressource de votre AWS CloudFormation modèle. Utilisez cette fonctionnalité pour importer vos AWS CloudFormation modèles existants AWS CDK un par un. Ce faisant, vous pouvez gérer vos ressources existantes à l'aide de AWS CDK constructions permettant de tirer parti des avantages des abstractions de haut niveau. Vous pouvez également utiliser cette fonctionnalité pour vendre vos AWS CloudFormation modèles aux AWS CDK développeurs en fournissant une API de AWS CDK construction.

> **Note**
>
> AWS CDK La v1 était également incluse `aws-cdk-lib.CfnInclude`, qui était auparavant utilisée dans le même but général. Cependant, il lui manque une grande partie des fonctionnalités de`cloudformation-include.CfnInclude`.

Rubriques

Importation d'un AWS CloudFormation modèle

Voici un exemple de AWS CloudFormation modèle que nous utiliserons pour fournir des exemples dans cette rubrique. Copiez et enregistrez le modèle comme `my-template.json` suit. Après avoir étudié ces exemples, vous pouvez approfondir votre exploration en utilisant l'un de vos AWS CloudFormation modèles déployés existants. Vous pouvez les obtenir depuis la AWS CloudFormation console.

```
{
  "Resources": {
    "MyBucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

Vous pouvez utiliser des modèles JSON ou YAML. Nous recommandons le format JSON s'il est disponible, car les analyseurs YAML peuvent légèrement varier quant à ce qu'ils acceptent.

Voici un exemple d'importation de l'exemple de modèle dans votre AWS CDK application à l'aide de`cloudformation-include`. Les modèles sont importés dans le contexte d'une pile 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"
            });
        }
    }
}
```

Par défaut, l'importation d'une ressource préserve l'ID logique d'origine de la ressource tel qu'il figure dans le modèle. Ce comportement convient à l'importation d'un AWS CloudFormation modèle dans le AWS CDK, où les identifiants logiques doivent être conservés. AWS CloudFormation a besoin de ces informations pour reconnaître ces ressources importées comme étant les mêmes ressources que celles du AWS CloudFormation modèle.

Si vous développez un wrapper de AWS CDK construction pour le modèle afin qu'il puisse être utilisé par d'autres AWS CDK développeurs, demandez plutôt de AWS CDK générer de nouveaux identifiants de ressources. Ce faisant, la construction peut être utilisée plusieurs fois dans une pile

sans conflits de noms. Pour ce faire, définissez la `preserveLogicalIds` propriété sur `false` lors de l'importation du modèle. Voici un exemple :

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
});
```

Pour placer les ressources importées sous le contrôle de votre AWS CDK application, ajoutez la pile à App :

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { MyStack } from '../lib/my-stack';

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

JavaScript

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

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

Python

```
import aws_cdk as cdk
from mystack.my_stack import MyStack

app = cdk.App()
MyStack(app, "MyStack")
```

Java

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

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

        new MyStack(app, "MyStack");
    }
}
```

C#

```
using Amazon.CDK;

namespace CdkApp
```

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

Pour vérifier qu'aucune modification involontaire n'est apportée aux AWS ressources de la pile, vous pouvez effectuer une différence. Utilisez la AWS CDK CLI `cdk diff` commande et omettez les métadonnées AWS CDK spécifiques. Voici un exemple :

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

Une fois que vous avez importé un AWS CloudFormation modèle, l' AWS CDK application doit devenir la source fiable de vos ressources importées. Pour apporter des modifications à vos ressources, modifiez-les dans votre AWS CDK application et déployez-les à l'aide de la AWS CDK CLI cdk deploy commande.

Accès aux ressources importées

Le nom `template` figurant dans l'exemple de code représente le AWS CloudFormation modèle importé. Pour accéder à une ressource à partir de celui-ci, utilisez la `getResource()`méthode de l'objet. Pour accéder à la ressource renvoyée en tant que type de ressource spécifique, convertissez le résultat dans le type souhaité. Cela n'est pas nécessaire en Python ou JavaScript. Voici un exemple :

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");
```

À partir de cet exemple, `cfnBucket` il s'agit désormais d'une instance de la `aws-s3.CfnBucket`classe. Il s'agit d'une construction L1 qui représente la AWS CloudFormation ressource correspondante. Vous pouvez la traiter comme n'importe quelle autre ressource de ce type. Par exemple, vous pouvez obtenir sa valeur ARN à l'aide de la `bucket.attrArn` propriété.

Pour encapsuler la `CfnBucket` ressource L1 dans une `aws-s3.Bucket`instance L2 à la place, utilisez les méthodes statiques `fromBucketArn()fromBucketAttributes()`, ou. `fromBucketName()` Habituellement, la `fromBucketName()` méthode est la plus pratique. Voici un exemple :

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);
```

D'autres constructions L2 ont des méthodes similaires pour créer la construction à partir d'une ressource existante.

Lorsque vous encapsulez une construction L1 dans une construction L2, cela ne crée pas de nouvelle ressource. D'après notre exemple, nous ne créons pas un deuxième compartiment S3 ;. Au lieu de cela, la nouvelle `Bucket` instance encapsule l'instance existante`CfnBucket`.

D'après l'exemple, `bucket` il s'agit désormais d'une `Bucket` construction L2 qui se comporte comme n'importe quelle autre construction L2. Par exemple, vous pouvez accorder à une AWS Lambda fonction l'accès en écriture au bucket en utilisant la `grantWrite()`méthode pratique du bucket. Il n'est pas nécessaire de définir manuellement la politique AWS Identity and Access Management (IAM) nécessaire. Voici un exemple :

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

Remplacement des paramètres

Si votre AWS CloudFormation modèle contient des paramètres, vous pouvez les remplacer par des valeurs de temps de construction lors de l'importation en utilisant la `parameters` propriété. Dans l'exemple suivant, nous remplaçons le `UploadBucket` paramètre par l'ARN d'un bucket défini ailleurs dans notre AWS CDK code.

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 }
    }
});
```

Autres éléments du modèle

Vous pouvez importer n'importe quel élément de AWS CloudFormation modèle, pas uniquement des ressources. Les éléments importés font partie de la AWS CDK pile. Pour importer ces éléments, utilisez les méthodes suivantes de l'`CfnInclude`objet :

- `getCondition()`— AWS CloudFormation conditions.
- `getHook()`— AWS CloudFormation crochets pour les déploiements bleu/vert.
- `getMapping()`— AWS CloudFormation mappages.
- `getOutput()`— AWS CloudFormation sorties.
- `getParameter()`— AWS CloudFormation paramètres.
- `getRule()`— AWS CloudFormation règles pour les AWS Service Catalog modèles.

Chacune de ces méthodes renvoie une instance d'une classe qui représente le type spécifique d' AWS CloudFormation élément. Ces objets sont mutables. Les modifications que vous leur apportez apparaîtront dans le modèle généré à partir de la AWS CDK pile. Voici un exemple d'importation d'un paramètre depuis le modèle et de modification de sa valeur par défaut :

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";
```

Piles imbriquées

Vous pouvez importer des piles imbriquées en les spécifiant soit lors de l'importation de leur modèle principal, soit ultérieurement. Le modèle imbriqué doit être stocké dans un fichier local, mais référencé en tant que `NestedStack` ressource dans le modèle principal. En outre, le nom de ressource utilisé dans le AWS CDK code doit correspondre au nom utilisé pour la pile imbriquée dans le modèle principal.

Compte tenu de cette définition de ressource dans le modèle principal, le code suivant montre comment importer la pile imbriquée référencée dans les deux sens.

```
"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'
});
```

Vous pouvez importer plusieurs piles imbriquées avec l'une ou l'autre méthode. Lorsque vous importez le modèle principal, vous fournissez un mappage entre le nom de ressource de chaque pile imbriquée et son fichier modèle. Ce mappage peut contenir autant d'entrées que vous le souhaitez. Pour le faire après l'importation initiale, appelez `loadNestedStack()` une fois pour chaque pile imbriquée.

Après avoir importé une pile imbriquée, vous pouvez y accéder en utilisant la `getNestedStack()`méthode du modèle principal.

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;
```

La `getNestedStack()` méthode renvoie une `IncludedNestedStack`instance. À partir de cette instance, vous pouvez accéder à l' AWS CDK `NestedStack`instance via la `stack` propriété, comme indiqué dans l'exemple. Vous pouvez également accéder à l'objet AWS CloudFormation modèle d'origine via `includedTemplate` lequel vous pouvez charger des ressources et d'autres AWS CloudFormation éléments.

Obtenir une valeur depuis le magasin de paramètres de Systems Manager

Ils AWS Cloud Development Kit (AWS CDK) peuvent récupérer la valeur des attributs du AWS Systems Manager Parameter Store. Lors de la synthèse, le AWS CDK produit un jeton qui est résolu AWS CloudFormation lors du déploiement.

Les AWS CDK supports permettent de récupérer à la fois des valeurs simples et sécurisées. Vous pouvez demander une version spécifique de l'un ou l'autre type de valeur. Pour les valeurs simples,

vous pouvez omettre la version dans votre demande de récupération de la dernière version. Pour les valeurs sécurisées, vous devez spécifier la version lorsque vous demandez la valeur de l'attribut sécurisé.

> **Note**
>
> Cette rubrique explique comment lire les attributs depuis le magasin de AWS Systems Manager paramètres. Vous pouvez également lire les secrets du AWS Secrets Manager (voirObtenez une valeur de AWS Secrets Manager).

Rubriques

Lire les valeurs de Systems Manager au moment du déploiement

Pour lire les valeurs du magasin de paramètres de Systems Manager, utilisez le valueForStringparamètre et valueForSecureStringParameterles méthodes. Choisissez une méthode selon que l'attribut souhaité est une chaîne simple ou une valeur de chaîne sécurisée. Ces méthodes renvoient des jetons, et non la valeur réelle. La valeur est résolue AWS CloudFormation lors du déploiement. Voici un exemple :

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
```

Un nombre limité de AWS services prennent actuellement en charge cette fonctionnalité.

Lire les valeurs de Systems Manager au moment de la synthèse

Il est parfois utile de fournir un paramètre au moment de la synthèse. Ce faisant, le AWS CloudFormation modèle utilisera toujours la même valeur au lieu de résoudre la valeur lors du déploiement.

Pour lire une valeur depuis le magasin de paramètres de Systems Manager au moment de la synthèse, utilisez la valueFromLookupméthode (Python :`value_from_lookup`). Cette méthode renvoie la valeur réelle du paramètre sous forme de the section called “Contexte” valeur. Si la valeur n'est pas déjà mise en cache `cdk.json` ou transmise sur la ligne de commande, elle est extraite du AWS compte courant. Pour cette raison, la pile doit être synthétisée avec des informations d' AWS environnement explicites.

Voici un exemple :

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");
```

Seules les chaînes de Systems Manager simples peuvent être récupérées. Les chaînes sécurisées ne peuvent pas être récupérées. La dernière version sera toujours renvoyée. Il n'est pas possible de demander des versions spécifiques.

> **Important**
>
> La valeur récupérée se retrouvera dans votre AWS CloudFormation modèle synthétisé. Cela peut constituer un risque de sécurité, selon les personnes ayant accès à vos AWS CloudFormation modèles et le type de valeur qu'ils contiennent. En règle générale, n'utilisez pas cette fonctionnalité pour les mots de passe, les clés ou les autres valeurs que vous souhaitez garder confidentielles.

Écrire des valeurs dans Systems Manager

Vous pouvez utiliser la AWS CLI AWS Management Console, le ou un AWS SDK pour définir les valeurs des paramètres de Systems Manager. Les exemples suivants utilisent la commande 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"
```

Lorsque vous mettez à jour une valeur SSM qui existe déjà, incluez également l'`--overwrite`option.

```
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"
```

Obtenez une valeur de AWS Secrets Manager

Pour utiliser les valeurs AWS Secrets Manager de votre AWS CDK application, utilisez la méthode fromSecretAttributes(). Il représente une valeur extraite de Secrets Manager et utilisée au moment du AWS CloudFormation déploiement. Voici un exemple :

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

Utilisez la commande AWS CLI create-secret CLI pour créer un secret à partir de la ligne de commande, par exemple lors de tests :

```
aws secretsmanager create-secret --name ImportedSecret --secret-string
 mygroovybucket
```

La commande renvoie un ARN que vous pouvez utiliser avec l'exemple précédent.

Une fois que vous avez créé une `Secret` instance, vous pouvez obtenir la valeur du secret à partir de l'`secretValue`attribut de l'instance. La valeur est représentée par une `SecretValue` instance, un type spécial dethe section called “Jetons”. Parce que c'est un jeton, cela n'a de sens qu'après résolution. Votre application CDK n'a pas besoin d'accéder à sa valeur réelle. Au lieu de cela, l'application peut transmettre l'`SecretValue`instance (ou sa chaîne ou sa représentation numérique) à la méthode CDK qui a besoin de la valeur.

Régler une CloudWatch alarme

Utilisez le package aws-cloudwatch pour configurer les CloudWatch alarmes Amazon sur les métriques. CloudWatch Vous pouvez utiliser des indicateurs prédéfinis ou créer les vôtres.

Rubriques

- Utilisation d'une métrique existante
- Création de votre propre métrique
- Création de l'alarme

Utilisation d'une métrique existante

De nombreux modules de AWS Construct Library vous permettent de définir une alarme sur une métrique existante en transmettant le nom de la métrique à une méthode pratique sur une instance d'un objet contenant des métriques. Par exemple, étant donné une file d'attente Amazon SQS, vous pouvez obtenir la métrique à l'aide ApproximateNumberOfMessagesVisiblede la méthode metric () de la file d'attente :

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");
```

Création de votre propre métrique

Créez votre propre métrique comme suit, où la valeur de l'espace de noms doit être quelque chose comme AWS/SQS pour une file d'attente Amazon SQS. Vous devez également spécifier le nom et la dimension de votre métrique :

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" }
    }
});
```

Création de l'alarme

Une fois que vous avez une métrique, qu'elle soit existante ou que vous avez définie, vous pouvez créer une alarme. Dans cet exemple, l'alarme est déclenchée lorsque votre métrique est supérieure à 100 au cours de deux des trois dernières périodes d'évaluation. Vous pouvez utiliser des comparaisons telles que « less-than » dans vos alarmes via la `comparisonOperator` propriété. G reater-than-or-equal -to est la AWS CDK valeur par défaut, nous n'avons donc pas besoin de le spécifier.

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
});
```

Une autre méthode pour créer une alarme consiste à utiliser la méthode createAlarm () de la métrique, qui utilise essentiellement les mêmes propriétés que le `Alarm` constructeur. Vous n'avez pas besoin de transmettre la métrique, car elle est déjà connue.

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
});
```

Enregistrer et récupérer les valeurs des variables de contexte

Vous pouvez spécifier des variables de contexte à l'aide du AWS Cloud Development Kit (AWS CDK) CLI ou dans le `cdk.json` fichier. Utilisez ensuite la `TryGetContext` méthode pour récupérer les valeurs.

Rubriques

- Spécifier les variables de contexte
- Récupérer les valeurs des variables de contexte

Spécifier les variables de contexte

Vous pouvez spécifier une variable de contexte soit dans le cadre d'une AWS CDK CLI commande, soit dans`cdk.json`.

Pour créer une variable de contexte en ligne de commande, utilisez l'option --context (-c), comme indiqué dans l'exemple suivant.

```
cdk synth -c bucket_name=mygroovybucket
```

Pour spécifier la même variable de contexte et la même valeur dans le `cdk.json` fichier, utilisez le code suivant.

```
{
  "context": {
    "bucket_name": "myotherbucket"
```

```
  }
}
```

Si vous spécifiez une variable de contexte en utilisant à la fois le `cdk.json` fichier AWS CDK CLI et, la AWS CDK CLI valeur est prioritaire.

Récupérer les valeurs des variables de contexte

Pour obtenir la valeur d'une variable de contexte dans votre application, utilisez la `TryGetContext` méthode dans le contexte d'une construction. (C'est-à-dire quand`this`, ou `self` en Python, est une instance d'une construction.)

Dans cet exemple, nous récupérons la valeur de la variable de `bucket_name` contexte. Si la valeur demandée n'est pas définie, `TryGetContext` renvoie `undefined` (`None`en Python ; `null` en Java et C# ; `nil` en Go) plutôt que de déclencher une exception.

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");
```

En dehors du contexte d'une construction, vous pouvez accéder à la variable de contexte depuis l'objet de l'application, comme ceci.

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");
```

Pour plus de détails sur l'utilisation des variables de contexte, consultezthe section called “Contexte”.

Utilisation des ressources du registre AWS CloudFormation public

Le registre AWS CloudFormation public vous permet de gérer les extensions, publiques et privées, telles que les ressources, les modules et les hooks qui peuvent être utilisés dans votre Compte AWS. Vous pouvez utiliser des extensions de ressources publiques dans vos AWS Cloud Development Kit (AWS CDK) applications avec la `CfnResource`construction.

Pour en savoir plus sur le registre AWS CloudFormation public, voir Utilisation du AWS CloudFormation registre dans le guide de AWS CloudFormation l'utilisateur.

Toutes les extensions publiques publiées par AWS sont disponibles pour tous les comptes dans toutes les régions sans aucune action de votre part. Cependant, vous devez activer chaque extension

tierce que vous souhaitez utiliser, dans chaque compte et dans chaque région où vous souhaitez l'utiliser.

> **Note**
>
> Lorsque vous utilisez AWS CloudFormation des types de ressources tiers, des frais vous seront facturés. Les frais sont basés sur le nombre d'opérations de traitement que vous effectuez par mois et sur la durée des opérations de traitement. Consultez les CloudFormation tarifs pour obtenir tous les détails.

Pour en savoir plus sur les extensions publiques, voir Utilisation des extensions publiques CloudFormation dans le Guide de AWS CloudFormation l'utilisateur

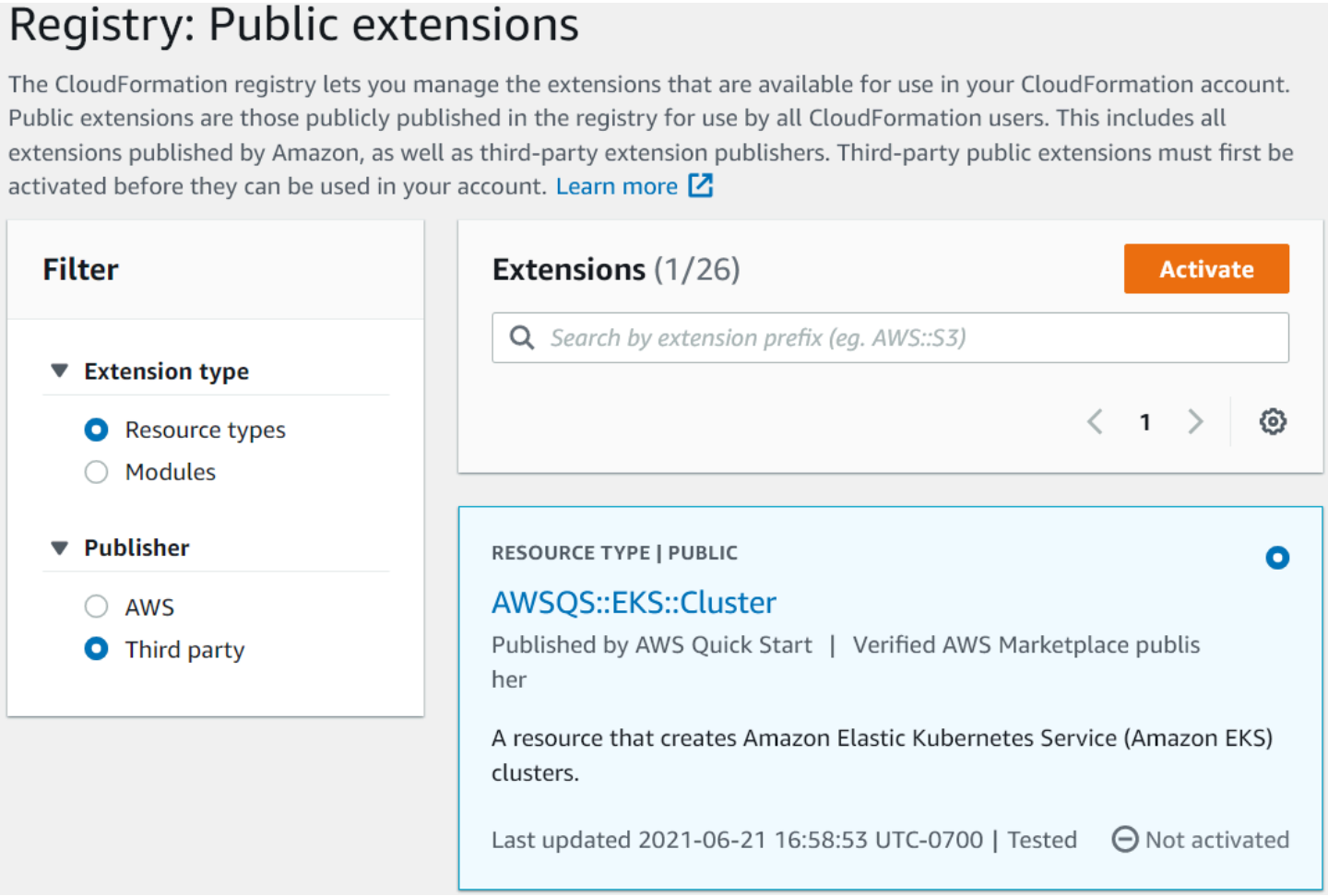
Rubriques

- Activation d'une ressource tierce dans votre compte et dans votre région
- Ajouter une ressource du registre AWS CloudFormation public à votre application CDK

Activation d'une ressource tierce dans votre compte et dans votre région

Les extensions publiées par AWS ne nécessitent pas d'activation. Ils sont toujours disponibles dans tous les comptes et toutes les régions. Vous pouvez activer une extension tierce par le AWS Management Console biais de AWS Command Line Interface, ou en déployant une AWS CloudFormation ressource spéciale.

Pour activer une extension tierce via le AWS Management Console ou pour voir quelles ressources sont disponibles



1. Connectez-vous au AWS compte sur lequel vous souhaitez utiliser l'extension, puis passez à la région dans laquelle vous souhaitez l'utiliser.
2. Accédez à la CloudFormation console via le menu Services.
3. Choisissez Extensions publiques dans la barre de navigation, puis activez le bouton radio Third party sous Publisher. La liste des extensions publiques tierces disponibles s'affiche. (Vous pouvez également AWSchoisir de consulter la liste des extensions publiques publiées par AWS, mais vous n'avez pas besoin de les activer.)
4. Parcourez la liste et trouvez l'extension que vous souhaitez activer. Vous pouvez également le rechercher, puis activer le bouton radio dans le coin supérieur droit de la carte de l'extension.
5. Cliquez sur le bouton Activer en haut de la liste pour activer l'extension sélectionnée. La page Activer de l'extension apparaît.

6. Sur la page Activer, vous pouvez remplacer le nom par défaut de l'extension et spécifier un rôle d'exécution et une configuration de journalisation. Vous pouvez également choisir de mettre à jour automatiquement l'extension lorsqu'une nouvelle version est publiée. Lorsque vous avez défini ces options comme vous le souhaitez, choisissez Activer l'extension en bas de la page.

Pour activer une extension tierce à l'aide du AWS CLI

- Utilisez la commande `activate-type`. Remplacez l'ARN du type personnalisé que vous souhaitez utiliser là où cela est indiqué.

  Voici un exemple :

  ```
  aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
  ```

Pour activer une extension tierce par le biais CloudFormation d'un CDK

- Déployez une ressource de type `AWS::CloudFormation::TypeActivation` et spécifiez les propriétés suivantes :

  a. `TypeName`- Le nom du type, par exemple`AWSQS::EKS::Cluster`.

  b. `MajorVersion`- Le numéro de version majeure de l'extension que vous souhaitez. Omettez cette option si vous voulez la dernière version.

  c. `AutoUpdate`- S'il faut mettre à jour automatiquement cette extension lorsqu'une nouvelle version mineure est publiée par l'éditeur. (Les mises à jour majeures des versions nécessitent une modification explicite de la `MajorVersion` propriété.)

  d. `ExecutionRoleArn`- L'ARN du rôle IAM sous lequel cette extension sera exécutée.

  e. `LoggingConfig`- La configuration de journalisation de l'extension.

  La `TypeActivation` ressource peut être déployée par le CDK à l'aide de la `CfnResource`construction. Ceci est indiqué pour les extensions réelles dans la section suivante.

Ajouter une ressource du registre AWS CloudFormation public à votre application CDK

Utilisez la `CfnResource`construction pour inclure une ressource du registre AWS CloudFormation public dans votre application. Cette construction se trouve dans le `aws-cdk-lib` module du CDK.

Supposons, par exemple, qu'il existe une ressource publique nommée `MY::S5::UltimateBucket` que vous souhaitez utiliser dans votre AWS CDK application. Cette ressource n'a qu'une seule propriété : le nom du bucket. L'`CfnResource`instanciation correspondante ressemble à ceci.

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"
    }
});
```

Déploiement AWS CDK d'applications

Déployez AWS Cloud Development Kit (AWS CDK) des applications.

Rubriques

AWS CDK validation des politiques au moment de la synthèse

Rubriques

Validation des politiques au moment de la synthèse

Si vous ou votre organisation utilisez un outil de validation des politiques, tel que AWS CloudFormation Guardl'OPA, pour définir des contraintes sur votre AWS CloudFormation modèle, vous pouvez les intégrer AWS CDK au moment de la synthèse. En utilisant le plugin de validation des politiques approprié, vous pouvez faire en sorte que l' AWS CDK application vérifie le AWS CloudFormation modèle généré par rapport à vos politiques immédiatement après la synthèse. En cas de violation, la synthèse échouera et un rapport sera imprimé sur la console.

La validation effectuée AWS CDK au moment de la synthèse contrôle les contrôles à un moment donné du cycle de vie du déploiement, mais elle ne peut pas affecter les actions qui se produisent en dehors de la synthèse. Les exemples incluent les actions effectuées directement dans la console ou via des API de service. Ils ne résistent pas à l'altération des AWS CloudFormation modèles après la synthèse. Un autre mécanisme permettant de valider le même ensemble de règles avec plus d'autorité doit être configuré indépendamment, comme les AWS CloudFormation hooks ou AWS Config. Néanmoins, la capacité d' AWS CDK évaluer l'ensemble de règles pendant le développement est toujours utile car elle améliorera la vitesse de détection et la productivité des développeurs.

L'objectif de la validation des AWS CDK politiques est de minimiser la quantité de configuration nécessaire au cours du développement et de le rendre aussi simple que possible.

> **Note**
>
> Cette fonctionnalité est considérée comme expérimentale, et l'API du plugin et le format du rapport de validation sont susceptibles d'être modifiés à l'avenir.

Rubriques

Pour les développeurs d'applications

Pour utiliser un ou plusieurs plugins de validation dans votre application, utilisez la `policyValidationBeta1` propriété suivante `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: [...],
});
```

Immédiatement après la synthèse, tous les plugins enregistrés de cette manière seront invoqués pour valider tous les modèles générés dans le champ que vous avez défini. En particulier, si vous enregistrez les modèles dans l'Appobjet, tous les modèles seront soumis à validation.

> **Warning**
>
> Outre la modification de l'assemblage du cloud, les plugins peuvent faire tout ce que votre AWS CDK application peut faire. Ils peuvent lire les données du système de fichiers, accéder au réseau, etc. En tant que consommateur d'un plugin, il est de votre responsabilité de vérifier que son utilisation est sûre.

AWS CloudFormation Guard plugin

L'utilisation du CfnGuardValidatorplugin vous permet d'AWS CloudFormation Guardeffectuer des validations de politiques. Le `CfnGuardValidator` plugin est livré avec un ensemble sélectionné de contrôles AWS Control Tower proactifs intégrés. L'ensemble de règles actuel se trouve dans la documentation du projet. Comme indiqué dansValidation des politiques au moment de la synthèse, nous recommandons aux organisations de mettre en place une méthode de validation plus fiable à l'aide de AWS CloudFormation crochets.

Pour les AWS Control Towerclients, ces mêmes contrôles proactifs peuvent être déployés dans l'ensemble de votre organisation. Lorsque vous activez des contrôles AWS Control Tower proactifs dans votre AWS Control Tower environnement, ils peuvent arrêter le déploiement de ressources non conformes déployées via AWS CloudFormation. Pour plus d'informations sur les contrôles proactifs gérés et leur fonctionnement, consultez la AWS Control Tower documentation.

Il est préférable d'utiliser conjointement ces contrôles AWS CDK groupés et ces contrôles AWS Control Tower proactifs gérés. Dans ce scénario, vous pouvez configurer ce plugin de validation avec les mêmes contrôles proactifs que ceux actifs dans votre environnement AWS Control Tower cloud. Vous pouvez alors rapidement être sûr que votre AWS CDK application passera les AWS Control Tower contrôles en s'exécutant `cdk synth` localement.

Rapport de validation

Lorsque vous synthétiserez l' AWS CDK application, les plugins de validation seront appelés et les résultats seront imprimés. Un exemple de rapport est présenté ci-dessous.

```
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
```

Par défaut, le rapport sera imprimé dans un format lisible par l'homme. Si vous souhaitez un rapport au format JSON, activez-le `@aws-cdk/core:validationReportJson` via la CLI ou transmettez-le directement à l'application :

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

Vous pouvez également définir cette paire clé-valeur de contexte à l'aide des `cdk.context.json` fichiers `cdk.json` or du répertoire de votre projet (voirContexte d'exécution).

Si vous choisissez le format JSON, le rapport de validation des politiques AWS CDK sera imprimé dans un fichier appelé `policy-validation-report.json` dans le répertoire d'assemblage du cloud. Pour le format par défaut lisible par l'homme, le rapport sera imprimé sur la sortie standard.

Pour les auteurs de plugins

Plugins

Le framework de AWS CDK base est chargé d'enregistrer et d'appeler les plugins, puis d'afficher le rapport de validation formaté. La responsabilité du plugin est d'agir en tant que couche de traduction entre le AWS CDK framework et l'outil de validation des politiques. Un plugin peut être créé dans n'importe quelle langue prise en charge par AWS CDK. Si vous créez un plugin susceptible d'être utilisé par plusieurs langues, il est recommandé de le créer `TypeScript` afin de pouvoir utiliser JSII pour le publier dans chaque AWS CDK langue.

Création de plugins

Le protocole de communication entre le module AWS CDK principal et votre outil de politique est défini par l'`IPolicyValidationPluginBeta1`interface. Pour créer un nouveau plugin, vous devez écrire une classe qui implémente cette interface. Il y a deux choses que vous devez implémenter : le nom du plugin (en remplaçant la `name` propriété) et la `validate()` méthode.

Le framework appellera `validate()` en passant un `IValidationContextBeta1` objet. L'emplacement des modèles à valider est indiqué par`templatePaths`. Le plugin doit renvoyer une instance de`ValidationPluginReportBeta1`. Cet objet représente le rapport que l'utilisateur recevra à la fin de la synthèse.

```
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',
      }],
    }],
  };
```

```
}
```

Notez que les plugins ne sont pas autorisés à modifier quoi que ce soit dans l'assemblage du cloud. Toute tentative en ce sens entraînera l'échec de la synthèse.

Si votre plugin dépend d'un outil externe, n'oubliez pas que certains développeurs n'ont peut-être pas encore installé cet outil sur leur poste de travail. Pour minimiser les frictions, nous vous recommandons vivement de fournir un script d'installation avec votre package de plugin, afin d'automatiser l'ensemble du processus. Mieux encore, exécutez ce script dans le cadre de l'installation de votre package. Avec`npm`, par exemple, vous pouvez l'ajouter au `postinstall` script du `package.json` fichier.

Gestion des exemptions

Si votre organisation dispose d'un mécanisme de gestion des exemptions, celui-ci peut être mis en œuvre dans le cadre du plug-in de validation.

Exemple de scénario illustrant un éventuel mécanisme d'exemption :

- Une organisation dispose d'une règle selon laquelle les compartiments publics Amazon S3 ne sont pas autorisés, sauf dans certains cas.
- Un développeur crée un compartiment Amazon S3 correspondant à l'un de ces scénarios et demande une exemption (création d'un ticket par exemple).
- Les outils de sécurité savent lire à partir du système interne qui enregistre les exemptions

Dans ce scénario, le développeur demanderait une exception dans le système interne, puis aurait besoin d'un moyen « d'enregistrer » cette exception. Pour compléter l'exemple du plugin Guard, vous pouvez créer un plugin qui gère les exemptions en filtrant les violations associées à une exemption correspondante dans un système de billetterie interne.

Consultez les plugins existants pour des exemples d'implémentations.

- @cdklabs/cdk-validator-cfnguard

Intégration et livraison continues (CI/CD) à l'aide de CDK Pipelines

Utilisez le module CDK Pipelines de AWS la bibliothèque Construct pour configurer la livraison continue AWS CDK des applications. Lorsque vous validez le code source de votre application CDK

dans AWS CodeCommitGitHub AWS CodeStar, CDK Pipelines peut automatiquement créer, tester et déployer votre nouvelle version.

Les CDK Pipelines se mettent à jour automatiquement. Si vous ajoutez des étapes ou des piles d'application, le pipeline se reconfigure automatiquement pour déployer ces nouvelles étapes ou piles.

> **Note**
>
> CDK Pipelines prend en charge deux API. L'une est l'API originale qui a été mise à disposition dans la version préliminaire de CDK Pipelines pour les développeurs. L'autre est une API moderne qui intègre les commentaires des clients du CDK reçus lors de la phase de prévisualisation. Les exemples présentés dans cette rubrique utilisent l'API moderne. Pour plus de détails sur les différences entre les deux API prises en charge, consultez l'API originale de CDK Pipelines dans le référentiel GitHub aws-cdk.

Rubriques

Bootstrap vos environnements AWS

Avant de pouvoir utiliser CDK Pipelines, vous devez démarrer AWS l'environnement dans lequel vous allez déployer vos stacks.

Un pipeline CDK implique au moins deux environnements. Le premier environnement est celui où le pipeline est approvisionné. Le second environnement est celui dans lequel vous souhaitez déployer les piles ou les étapes de l'application (les étapes sont des groupes de piles connexes). Ces environnements peuvent être identiques, mais l'une des meilleures pratiques consiste à isoler les étapes les unes des autres dans des environnements différents.

> **Note**
>
> Consultez the section called “Action d'amorçage” pour plus d'informations sur les types de ressources créées par le bootstrap et sur la façon de personnaliser la pile bootstrap.

Le déploiement continu avec CDK Pipelines nécessite l'inclusion des éléments suivants dans la pile CDK Toolkit :

- Un bucket Amazon Simple Storage Service (Amazon S3).
- Un référentiel Amazon ECR.
- Des rôles IAM pour donner aux différentes parties d'un pipeline les autorisations dont elles ont besoin.

Le kit d'outils CDK mettra à niveau votre stack bootstrap existant ou en créera un nouveau si nécessaire.

Pour démarrer un environnement capable de provisionner un AWS CDK pipeline, invoquez-le `cdk bootstrap` comme indiqué dans l'exemple suivant. L'invocation du AWS CDK Toolkit via la `npx` commande l'installe temporairement si nécessaire. Il utilisera également la version du Toolkit installée dans le projet en cours, s'il en existe une.

--cloudformation-execution-policiesspécifie l'ARN d'une politique selon laquelle les futurs déploiements de CDK Pipelines seront exécutés. La `AdministratorAccess` politique par défaut garantit que votre pipeline peut déployer tous les types de AWS ressources. Si vous utilisez cette politique, assurez-vous de faire confiance à l'ensemble du code et aux dépendances qui constituent votre AWS CDK application.

La plupart des organisations imposent des contrôles plus stricts sur les types de ressources pouvant être déployés par automatisation. Renseignez-vous auprès du service approprié au sein de votre organisation pour déterminer la politique que votre pipeline doit utiliser.

Vous pouvez omettre --profile cette option si votre AWS profil par défaut contient la configuration d'authentification nécessaire et Région 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
```

Pour démarrer des environnements supplémentaires dans lesquels AWS CDK les applications seront déployées par le pipeline, utilisez plutôt les commandes suivantes. L'--trustoption indique quel autre compte doit être autorisé à déployer AWS CDK des applications dans cet environnement. Pour cette option, spécifiez l'ID de AWS compte du pipeline.

Encore une fois, vous pouvez omettre --profile cette option si votre AWS profil par défaut contient la configuration d'authentification nécessaire et Région 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**
>
> Utilisez les informations d'identification administratives uniquement pour démarrer et approvisionner le pipeline initial. Ensuite, utilisez le pipeline lui-même, et non votre machine locale, pour déployer les modifications.

Si vous mettez à niveau un environnement bootstrap existant, le compartiment Amazon S3 précédent devient orphelin lors de la création du nouveau compartiment. Supprimez-le manuellement à l'aide de la console Amazon S3.

Initialisation d'un projet

Créez un nouveau GitHub projet vide et clonez-le sur votre poste de travail dans le `my-pipeline` répertoire. (Les exemples de code présentés dans cette rubrique utilisent GitHub. Vous pouvez également utiliser AWS CodeStar ou AWS CodeCommit.)

```
git clone GITHUB-CLONE-URL my-pipeline
cd my-pipeline
```

> **Note**
>
> Vous pouvez utiliser un nom autre que celui `my-pipeline` du répertoire principal de votre application. Toutefois, si vous le faites, vous devrez modifier les noms de fichiers et de classes plus loin dans cette rubrique. Cela est dû au fait que le AWS CDK Toolkit base certains noms de fichiers et de classes sur le nom du répertoire principal.

Après le clonage, initialisez le projet comme d'habitude.

TypeScript

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

JavaScript

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

Python

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

Une fois l'application créée, entrez également les deux commandes suivantes. Ils activent l'environnement virtuel Python de l'application et installent les dépendances AWS CDK principales.

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

Java

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

Si vous utilisez un IDE, vous pouvez désormais ouvrir ou importer le projet. Dans Eclipse, par exemple, choisissez Fichier > Importer > Maven > Projets Maven existants. Assurez-vous que les paramètres du projet sont définis pour utiliser Java 8 (1.8).

C#

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

Si vous utilisez Visual Studio, ouvrez le fichier de solution dans le `src` répertoire.

Go

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

Une fois l'application créée, entrez également la commande suivante pour installer les modules AWS Construct Library dont l'application a besoin.

```
go get
```

Important

Assurez-vous de valider vos `cdk.context.json` fichiers `cdk.json` et dans le contrôle de source. Les informations contextuelles (telles que les indicateurs de fonctionnalités et les valeurs mises en cache extraites de votre AWS compte) font partie de l'état de votre projet. Les valeurs peuvent être différentes dans un autre environnement, ce qui peut entraîner des modifications inattendues de vos résultats. Pour plus d'informations, consultez the section called “Contexte”.

Définition d'un pipeline

Votre application CDK Pipelines comprendra au moins deux piles : une qui représente le pipeline lui-même et une ou plusieurs piles qui représentent l'application déployée par son intermédiaire. Les piles peuvent également être regroupées en étapes, que vous pouvez utiliser pour déployer des copies des piles d'infrastructure dans différents environnements. Pour l'instant, nous examinerons le pipeline, puis nous pencherons sur l'application qu'il déploiera.

La construction `CodePipeline`est la construction qui représente un pipeline CDK utilisé AWS CodePipeline comme moteur de déploiement. Lorsque vous instanciez `CodePipeline` dans une pile, vous définissez l'emplacement de la source pour le pipeline (tel qu'un GitHub référentiel). Vous définissez également les commandes pour créer l'application.

Par exemple, ce qui suit définit un pipeline dont la source est stockée dans un GitHub référentiel. Il inclut également une étape de génération pour une application TypeScript CDK. Renseignez les informations relatives à votre GitHub dépôt à l'endroit indiqué.

> **Note**
>
> Par défaut, le pipeline s'authentifie à GitHub l'aide d'un jeton d'accès personnel stocké dans Secrets Manager sous le nom`github-token`.

Vous devrez également mettre à jour l'instanciation de la pile de pipelines pour spécifier le AWS compte et la région.

TypeScript

Dans `lib/my-pipeline-stack.ts` (peut varier si le dossier de votre projet n'est pas nommé`my-pipeline`) :

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { CodePipeline, CodePipelineSource, ShellStep } from 'aws-cdk-lib/pipelines';

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

    const pipeline = new CodePipeline(this, 'Pipeline', {
```

```
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });
  }
}
```

Dans `bin/my-pipeline.ts` (peut varier si le dossier de votre projet n'est pas nommé`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

Dans `lib/my-pipeline-stack.js` (peut varier si le dossier de votre projet n'est pas nommé`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 }
```

Dans `bin/my-pipeline.js` (peut varier si le dossier de votre projet n'est pas nommé`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

Dans `my-pipeline/my-pipeline-stack.py` (peut varier si le dossier de votre projet n'est pas nommé`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"]
                        )
                    )
```

Dans `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

Dans `src/main/java/com/myorg/MyPipelineStack.java` (peut varier si le dossier de votre projet n'est pas nommé`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();
    }
}
```

Dans `src/main/java/com/myorg/MyPipelineApp.java` (peut varier si le dossier de votre projet n'est pas nommé`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#

Dans `src/MyPipeline/MyPipelineStack.cs` (peut varier si le dossier de votre projet n'est pas nommé`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" }
                })
            });
        }
    }
}
```

Dans `src/MyPipeline/Program.cs` (peut varier si le dossier de votre projet n'est pas nommé`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();
        }
    }
}
```

Vous devez déployer un pipeline manuellement une fois. Ensuite, le pipeline se tient à jour à partir du référentiel de code source. Assurez-vous donc que le code du dépôt est celui que vous souhaitez déployer. Enregistrez vos modifications, puis envoyez-les GitHub, puis déployez-les :

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

> **Tip**
>
> Maintenant que vous avez effectué le déploiement initial, votre AWS compte local n'a plus besoin d'accès administratif. En effet, toutes les modifications apportées à votre application seront déployées via le pipeline. Tout ce que vous devez être capable de faire, c'est d'appuyer dessus GitHub.

Étapes de candidature

Pour définir une AWS application multi-stack pouvant être ajoutée au pipeline en une seule fois, définissez une sous-classe de. `Stage` (Ceci est différent de celui du `CdkStage` module CDK Pipelines.)

La scène contient les piles qui constituent votre application. S'il existe des dépendances entre les piles, celles-ci sont automatiquement ajoutées au pipeline dans le bon ordre. Les piles qui ne dépendent pas les unes des autres sont déployées en parallèle. Vous pouvez ajouter une relation de dépendance entre les piles en appelant`stack1.addDependency(stack2)`.

Les stages acceptent un `env` argument par défaut, qui devient l'environnement par défaut pour les piles qu'ils contiennent. (Les piles peuvent toujours avoir leur propre environnement spécifié.)

Une application est ajoutée au pipeline en appelant `addStage()` avec des instances de `Stage`. Une étape peut être instanciée et ajoutée au pipeline plusieurs fois pour définir les différentes étapes de votre pipeline d'applications DTAP ou multirégional.

Nous allons créer une pile contenant une fonction Lambda simple et placer cette pile dans une phase. Ensuite, nous ajouterons la scène au pipeline afin qu'elle puisse être déployée.

TypeScript

Créez le nouveau fichier `lib/my-pipeline-lambda-stack.ts` contenant notre pile d'applications contenant une fonction 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";')
      });
    }
}
```

Créez le nouveau fichier `lib/my-pipeline-app-stage.ts` pour accueillir notre scène.

```
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');
    }
}
```

Modifiez `lib/my-pipeline-stack.ts` pour ajouter l'étape à notre pipeline.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { CodePipeline, CodePipelineSource, ShellStep } from 'aws-cdk-lib/pipelines';
import { MyPipelineAppStage } from './my-pipeline-app-stage';
```

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

    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });

    pipeline.addStage(new MyPipelineAppStage(this, "test", {
      env: { account: "111111111111", region: "eu-west-1" }
    }));
  }
}
```

JavaScript

Créez le nouveau fichier `lib/my-pipeline-lambda-stack.js` contenant notre pile d'applications contenant une fonction 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 }
```

Créez le nouveau fichier `lib/my-pipeline-app-stage.js` pour accueillir notre scène.

```
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 };
```

Modifiez `lib/my-pipeline-stack.ts` pour ajouter l'étape à notre pipeline.

```
const cdk = require('aws-cdk-lib');
const { CodePipeline, CodePipelineSource, ShellStep } = require('aws-cdk-lib/
pipelines');
const { MyPipelineAppStage } = require('./my-pipeline-app-stage');

 class MyPipelineStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });

    pipeline.addStage(new MyPipelineAppStage(this, "test", {
      env: { account: "111111111111", region: "eu-west-1" }
    }));

  }
}

module.exports = { MyPipelineStack }
```

Python

Créez le nouveau fichier `my_pipeline/my_pipeline_lambda_stack.py` contenant notre pile d'applications contenant une fonction 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';")
        )
```

Créez le nouveau fichier `my_pipeline/my_pipeline_app_stage.py` pour accueillir notre scène.

```
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")
```

Modifiez `my_pipeline/my-pipeline-stack.py` pour ajouter l'étape à notre pipeline.

```
import aws_cdk as cdk
from constructs import Construct
from aws_cdk.pipelines import CodePipeline, CodePipelineSource, ShellStep
from my_pipeline.my_pipeline_app_stage import MyPipelineAppStage

class MyPipelineStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
```

```
        super().__init__(scope, construct_id, **kwargs)

        pipeline =  CodePipeline(self, "Pipeline",
                        pipeline_name="MyPipeline",
                        synth=ShellStep("Synth",
                            input=CodePipelineSource.git_hub("OWNER/REPO", "main"),
                            commands=["npm install -g aws-cdk",
                                "python -m pip install -r requirements.txt",
                                "cdk synth"]))

        pipeline.add_stage(MyPipelineAppStage(self, "test",
            env=cdk.Environment(account="111111111111", region="eu-west-1")))
```

Java

Créez le nouveau fichier `src/main/java/com.myorg/MyPipelineLambdaStack.java` contenant notre pile d'applications contenant une fonction 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();
```

```
    }

}
```

Créez le nouveau fichier `src/main/java/com.myorg/MyPipelineAppStage.java` pour accueillir notre scène.

```
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");
    }

}
```

Modifiez `src/main/java/com.myorg/MyPipelineStack.java` pour ajouter l'étape à notre pipeline.

```
package com.myorg;

import java.util.Arrays;
import software.constructs.Construct;
import software.amazon.awscdk.Environment;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.StageProps;
import software.amazon.awscdk.pipelines.CodePipeline;
import software.amazon.awscdk.pipelines.CodePipelineSource;
import software.amazon.awscdk.pipelines.ShellStep;
```

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

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

        final CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
            .pipelineName("MyPipeline")
            .synth(ShellStep.Builder.create("Synth")
                .input(CodePipelineSource.gitHub("OWNER/REPO", "main"))
                .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))
                .build())
            .build();

        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
            .env(Environment.builder()
                .account("111111111111")
                .region("eu-west-1")
                .build())
            .build()));
    }
}
```

C#

Créez le nouveau fichier `src/MyPipeline/MyPipelineLambdaStack.cs` contenant notre pile d'applications contenant une fonction 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';")
            });
        }
    }
}
```

Créez le nouveau fichier `src/MyPipeline/MyPipelineAppStage.cs` pour accueillir notre scène.

```
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");
        }
    }
}
```

Modifiez `src/MyPipeline/MyPipelineStack.cs` pour ajouter l'étape à notre pipeline.

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.Pipelines;

namespace MyPipeline
{
    public class MyPipelineStack : Stack
    {
        internal MyPipelineStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
            {
```

```
                PipelineName = "MyPipeline",
                Synth = new ShellStep("Synth", new ShellStepProps
                {
                    Input = CodePipelineSource.GitHub("OWNER/REPO", "main"),
                    Commands = new string[] { "npm install -g aws-cdk", "cdk
 synth" }
                })
            });

            pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps
            {
                Env = new Environment
                {
                    Account = "111111111111", Region = "eu-west-1"
                }
            }));
        }
    }
}
```

Chaque étape d'application ajoutée par `addStage()` entraîne l'ajout d'une étape de pipeline correspondante, représentée par une StageDeploymentinstance renvoyée par l'`addStage()`appel. Vous pouvez ajouter des actions avant ou après le déploiement à la phase en appelant sa méthode `addPre()` or`addPost()`.

TypeScript

```
// import { ManualApprovalStep } from 'aws-cdk-lib/pipelines';

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, 'testing', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

    testingStage.addPost(new ManualApprovalStep('approval'));
```

JavaScript

```
// const { ManualApprovalStep } = require('aws-cdk-lib/pipelines');

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, 'testing', {
  env: { account: '111111111111', region: 'eu-west-1' }
```

```
}));

testingStage.addPost(new ManualApprovalStep('approval'));
```

Python

```
# from aws_cdk.pipelines import ManualApprovalStep

testing_stage = pipeline.add_stage(MyPipelineAppStage(self, "testing",
    env=cdk.Environment(account="111111111111", region="eu-west-1")))

testing_stage.add_post(ManualApprovalStep('approval'))
```

Java

```
// import software.amazon.awscdk.pipelines.StageDeployment;
// import software.amazon.awscdk.pipelines.ManualApprovalStep;

StageDeployment testingStage =
        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
                .env(Environment.builder()
                        .account("111111111111")
                        .region("eu-west-1")
                        .build())
                .build()));

testingStage.addPost(new ManualApprovalStep("approval"));
```

C#

```
var testingStage = pipeline.AddStage(new MyPipelineAppStage(this, "test", new
 StageProps
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));

testingStage.AddPost(new ManualApprovalStep("approval"));
```

Vous pouvez ajouter des étapes à une Wave pour les déployer en parallèle, par exemple lorsque vous déployez une étape sur plusieurs comptes ou régions.

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"
    }
}));
```

Tester les déploiements

Vous pouvez ajouter des étapes à un pipeline CDK pour valider les déploiements que vous effectuez. Par exemple, vous pouvez utiliser la bibliothèque CDK Pipeline `ShellStep` pour effectuer des tâches telles que les suivantes :

- Essayer d'accéder à un Amazon API Gateway récemment déployé soutenu par une fonction Lambda
- Vérification d'un paramètre d'une ressource déployée en émettant une AWS CLI commande

Dans sa forme la plus simple, l'ajout d'actions de validation ressemble à ceci :

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'" }
}));
```

De nombreux AWS CloudFormation déploiements entraînent la génération de ressources aux noms imprévisibles. De ce fait, les CDK Pipelines fournissent un moyen de AWS CloudFormation lire

les sorties après un déploiement. Cela permet de transmettre (par exemple) l'URL générée d'un équilibreur de charge à une action de test.

Pour utiliser les sorties, exposez l'`CfnOutput`objet qui vous intéresse. Transmettez-le ensuite dans la `envFromCfnOutputs` propriété d'une étape pour le rendre disponible en tant que variable d'environnement au sein de cette étape.

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" }
}));
```

Vous pouvez écrire des tests de validation simples directement dans le`ShellStep`, mais cette approche devient peu pratique lorsque le test ne se limite pas à quelques lignes. Pour les tests plus complexes, vous pouvez intégrer des fichiers supplémentaires (tels que des scripts shell complets ou des programmes dans d'autres langages) dans la `inputs` propriété `ShellStep` via. Les entrées peuvent être n'importe quelle étape ayant une sortie, y compris une source (telle qu'un GitHub dépôt) ou une autre`ShellStep`.

L'importation de fichiers depuis le dépôt source est appropriée si les fichiers sont directement utilisables dans le test (par exemple, s'ils sont eux-mêmes exécutables). Dans cet exemple, nous déclarons notre GitHub dépôt comme `source` (plutôt que de l'instancier en ligne dans le cadre du). `CodePipeline` Ensuite, nous transmettons ce jeu de fichiers au pipeline et au test de validation.

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" }
}));
```

L'obtention des fichiers supplémentaires à partir de l'étape de synthèse est appropriée si vos tests doivent être compilés, ce qui est effectué dans le cadre de la synthèse.

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" }
}));
```

Remarque de sécurité

Toute forme de livraison continue comporte des risques de sécurité inhérents. Dans le cadre du modèle de responsabilité AWS partagée, vous êtes responsable de la sécurité de vos informations dans le AWS cloud. La bibliothèque CDK Pipelines vous donne une longueur d'avance en incorporant des valeurs par défaut sécurisées et les meilleures pratiques de modélisation.

Cependant, de par sa nature même, une bibliothèque qui a besoin d'un niveau d'accès élevé pour remplir son objectif ne peut pas garantir une sécurité complète. Il existe de nombreux vecteurs d'attaque extérieurs AWS à votre organisation.

Gardez en particulier à l'esprit les points suivants :

- Faites attention aux logiciels dont vous dépendez. Vérifiez tous les logiciels tiers que vous utilisez dans votre pipeline, car ils peuvent modifier l'infrastructure déployée.
- Utilisez le verrouillage des dépendances pour éviter les mises à niveau accidentelles. CDK Pipelines `package-lock.json` respecte `yarn.lock` et veille à ce que vos dépendances soient celles que vous attendez.
- CDK Pipelines fonctionne sur des ressources créées dans votre propre compte, et la configuration de ces ressources est contrôlée par les développeurs qui soumettent du code via le pipeline. Par conséquent, CDK Pipelines ne peut à lui seul protéger contre les développeurs malveillants qui tentent de contourner les contrôles de conformité. Si votre modèle de menace inclut des développeurs écrivant du code CDK, vous devez disposer de mécanismes de conformité externes tels que AWS CloudFormation Hooks (préventif) ou AWS Config (réactif) que le rôle AWS CloudFormation d'exécution n'est pas autorisé à désactiver.
- Les informations d'identification pour les environnements de production doivent être de courte durée. Après le démarrage et le provisionnement initial, les développeurs n'ont pas du tout besoin d'avoir des informations d'identification de compte. Les modifications peuvent être déployées via le pipeline. Réduisez le risque de fuite d'informations d'identification en n'en ayant pas besoin au départ.

Résolution des problèmes

Les problèmes suivants sont fréquemment rencontrés lors de la prise en main de CDK Pipelines.

Pipeline : défaillance interne

```
CREATE_FAILED  | AWS::CodePipeline::Pipeline | Pipeline/Pipeline
Internal Failure
```

Vérifiez votre jeton GitHub d'accès. Il est peut-être absent ou ne dispose pas des autorisations nécessaires pour accéder au référentiel.

Clé : la politique contient une déclaration contenant un ou plusieurs principes non valides

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey
Policy contains a statement with one or more invalid principals.
```

L'un des environnements cibles n'a pas été amorcé avec la nouvelle pile d'amorçage. Assurez-vous que tous vos environnements cibles sont démarrés.

La pile est dans l'état ROLLBACK_COMPLETE et ne peut pas être mise à jour.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
ID: ...)
```

La pile a échoué lors de son déploiement précédent et est dans un état non réessayable. Supprimez la pile de la AWS CloudFormation console et réessayez le déploiement.

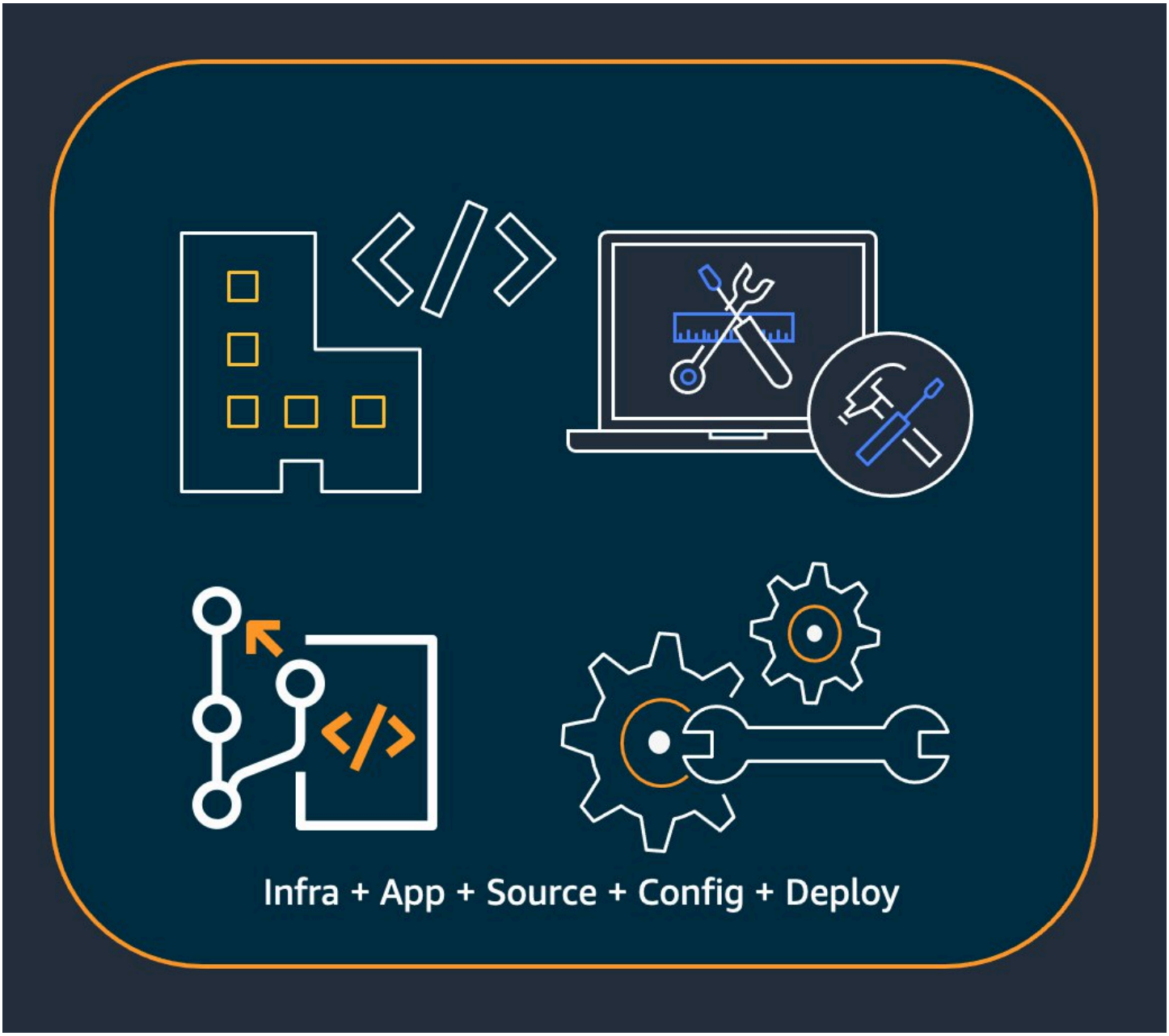
Les meilleures pratiques pour le développement et le déploiement d'une infrastructure cloud avec AWS CDK

Avec le AWS CDK, les développeurs ou les administrateurs peuvent définir leur infrastructure cloud à l'aide d'un langage de programmation pris en charge. Les applications CDK doivent être organisées en unités logiques, telles que l'API, la base de données et les ressources de surveillance, et éventuellement disposer d'un pipeline pour les déploiements automatisés. Les unités logiques doivent être implémentées sous forme de constructions comprenant les éléments suivants :

- Infrastructure (par exemple, des compartiments Amazon S3, des bases de données Amazon RDS ou un réseau Amazon VPC)
- Code d'exécution (AWS Lambda fonctions, par exemple)
- Code de configuration

Les piles définissent le modèle de déploiement de ces unités logiques. Pour une introduction plus détaillée aux concepts sous-jacents au CDK, voirPremiers pas.

Cela AWS CDK reflète une prise en compte attentive des besoins de nos clients et de nos équipes internes ainsi que des modèles de défaillance qui surviennent souvent lors du déploiement et de la maintenance continue d'applications cloud complexes. Nous avons découvert que les échecs sont souvent liés à des « out-of-band » modifications apportées à une application qui n'ont pas été entièrement testées, telles que des modifications de configuration. C'est pourquoi nous l'avons développé AWS CDK autour d'un modèle dans lequel l'ensemble de votre application est défini dans le code, non seulement la logique métier, mais également l'infrastructure et la configuration. Ainsi, les modifications proposées peuvent être soigneusement examinées, testées de manière exhaustive dans des environnements ressemblant à la production à des degrés divers, et annulées complètement en cas de problème.



Au moment du déploiement, le AWS CDK synthétise un assemblage cloud contenant les éléments suivants :

- AWS CloudFormation modèles décrivant votre infrastructure dans tous les environnements cibles
- Ressources de fichiers contenant votre code d'exécution et ses fichiers de support

Avec le CDK, chaque commit dans la branche principale de contrôle de version de votre application peut représenter une version complète, cohérente et déployable de votre application. Votre

application peut ensuite être déployée automatiquement chaque fois qu'une modification est apportée.

La philosophie qui les sous-tend AWS CDK conduit à nos meilleures pratiques recommandées, que nous avons divisées en quatre grandes catégories.

- the section called “Bonnes pratiques organisationnelles”
- the section called “Bonnes pratiques en matière de codage”
- the section called “Élaborez les meilleures pratiques”
- the section called “Bonnes pratiques en matière d'applications”

> **Tip**
>
> Tenez également compte des meilleures pratiques AWS CloudFormation et des AWS services individuels que vous utilisez, le cas échéant, pour l'infrastructure définie par le CDK.

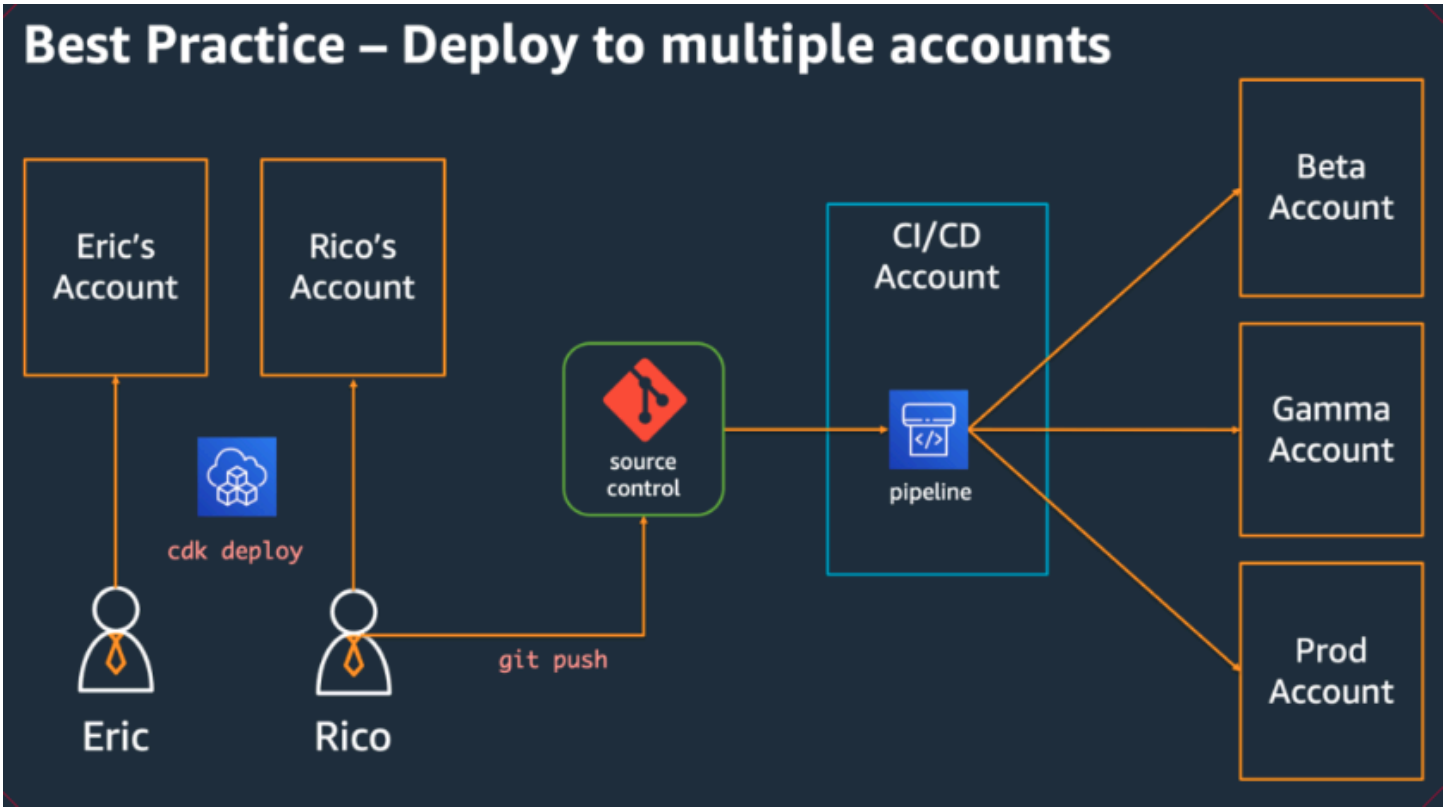
Bonnes pratiques organisationnelles

Au cours des premières étapes de AWS CDK l'adoption, il est important de réfléchir à la manière de mettre votre organisation sur la voie du succès. Il est recommandé de disposer d'une équipe d'experts chargée de former et de guider le reste de l'entreprise lors de l'adoption du CDK. La taille de cette équipe peut varier, allant d'une ou deux personnes dans une petite entreprise à un Cloud Center of Excellence (CCoE) à part entière dans une grande entreprise. Cette équipe est chargée de définir les normes et les politiques relatives à l'infrastructure cloud de votre entreprise, ainsi que de former et d'encadrer les développeurs.

Le CCoE peut fournir des conseils sur les langages de programmation à utiliser pour l'infrastructure cloud. Les détails peuvent varier d'une organisation à l'autre, mais une bonne politique permet de s'assurer que les développeurs peuvent comprendre et gérer l'infrastructure cloud de l'entreprise.

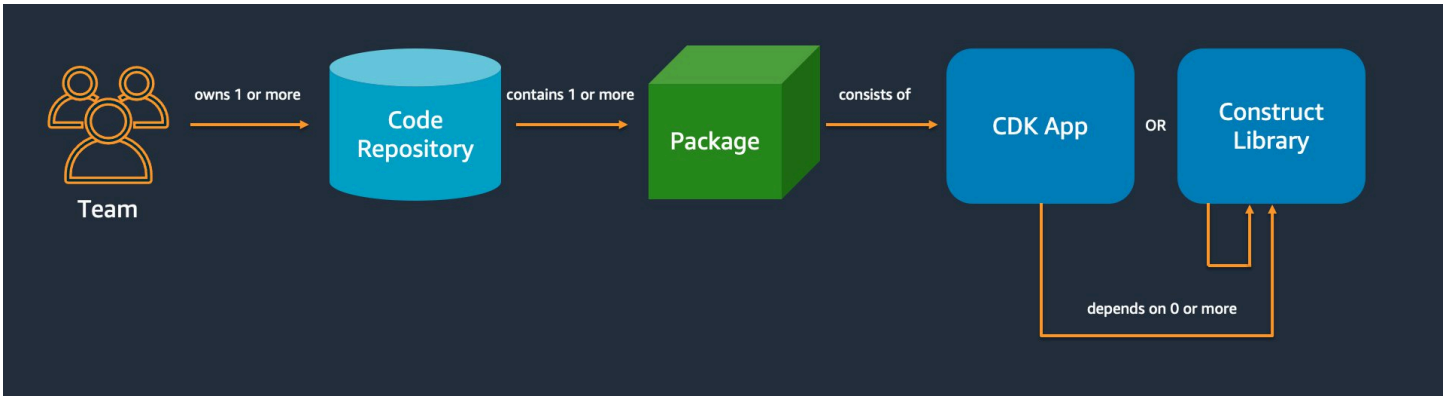
Le CCoE crée également une « zone de landing zone » qui définit les unités organisationnelles au sein AWS desquelles vous vous trouvez. Une zone d'atterrissage est un AWS environnement multi-comptes préconfiguré, sécurisé, évolutif, basé sur les meilleures pratiques. Pour relier les services qui constituent votre zone d'atterrissage, vous pouvez utiliser AWS Control Tower, qui configure et gère l'ensemble de votre système multi-comptes à partir d'une seule interface utilisateur.

Les équipes de développement devraient être en mesure d'utiliser leurs propres comptes pour tester et déployer de nouvelles ressources dans ces comptes selon les besoins. Les développeurs individuels peuvent traiter ces ressources comme des extensions de leur propre poste de développement. À l'aide de CDK Pipelines, AWS CDK les applications peuvent ensuite être déployées via un compte CI/CD dans des environnements de test, d'intégration et de production (chacun étant isolé dans sa AWS propre région ou compte). Cela se fait en fusionnant le code des développeurs dans le référentiel canonique de votre organisation.



Bonnes pratiques en matière de codage

Cette section présente les meilleures pratiques pour organiser votre AWS CDK code. Le schéma suivant montre la relation entre une équipe et les référentiels de code, les packages, les applications et les bibliothèques de construction de cette équipe.



Commencez simplement et ajoutez de la complexité uniquement lorsque vous en avez besoin

Le principe directeur de la plupart de nos meilleures pratiques est de garder les choses aussi simples que possible, mais rien de plus simple. Ajoutez de la complexité uniquement lorsque vos exigences imposent une solution plus complexe. Avec le AWS CDK, vous pouvez refactoriser votre code si nécessaire pour répondre aux nouvelles exigences. Vous n'avez pas à concevoir dès le départ tous les scénarios possibles.

S'aligner sur le framework AWS Well-Architected

Le AWS Well-Architected Framework définit un composant comme le code, la configuration AWS et les ressources qui, ensemble, répondent à une exigence. Un composant est souvent l'unité de propriété technique et est découplé des autres composants. Le terme charge de travail est utilisé pour identifier un ensemble de composants qui, ensemble, apportent une valeur commerciale. Une charge de travail est généralement le niveau de détail sur lequel les responsables commerciaux et technologiques communiquent.

Une AWS CDK application correspond à un composant tel que défini par le AWS Well-Architected Framework. AWS CDK les applications sont un mécanisme permettant de codifier et de diffuser les meilleures pratiques en matière d'applications cloud Well-Architected. Vous pouvez également créer et partager des composants sous forme de bibliothèques de code réutilisables via des référentiels d'artefacts, tels que. AWS CodeArtifact

Chaque application commence par un seul package dans un seul référentiel

Un package unique constitue le point d'entrée de votre AWS CDK application. Vous définissez ici comment et où déployer les différentes unités logiques de votre application. Vous définissez

également le pipeline CI/CD pour déployer l'application. Les structures de l'application définissent les unités logiques de votre solution.

Utilisez des packages supplémentaires pour les constructions que vous utilisez dans plusieurs applications. (Les constructions partagées doivent également avoir leur propre cycle de vie et leur propre stratégie de test.) Les dépendances entre les packages d'un même référentiel sont gérées par les outils de compilation de votre dépôt.

Bien que cela soit possible, nous vous déconseillons de placer plusieurs applications dans le même référentiel, en particulier lorsque vous utilisez des pipelines de déploiement automatisés. Cela augmente le « rayon d'action » des modifications lors du déploiement. Lorsqu'un référentiel contient plusieurs applications, les modifications apportées à une application déclenchent le déploiement des autres (même si les autres n'ont pas changé). De plus, une interruption dans une application empêche le déploiement des autres applications.

Transférez le code vers des référentiels en fonction du cycle de vie du code ou de la propriété de l'équipe

Lorsque des packages commencent à être utilisés dans plusieurs applications, déplacez-les vers leur propre référentiel. Ainsi, les packages peuvent être référencés par les systèmes de création d'applications qui les utilisent, et ils peuvent également être mis à jour à des cadences indépendantes du cycle de vie des applications. Cependant, dans un premier temps, il peut être judicieux de placer toutes les constructions partagées dans un seul référentiel.

Déplacez également les packages vers leur propre référentiel lorsque différentes équipes y travaillent. Cela permet de renforcer le contrôle d'accès.

Pour consommer des packages au-delà des limites des référentiels, vous avez besoin d'un référentiel de packages privé, similaire à NPM ou Maven Central PyPi, mais interne à votre organisation. Vous avez également besoin d'un processus de publication qui génère, teste et publie le package dans le référentiel de packages privé. CodeArtifactpeut héberger des packages pour les langages de programmation les plus courants.

Les dépendances vis-à-vis des packages du référentiel de packages sont gérées par le gestionnaire de packages de votre langue, tel que NPM for TypeScript ou JavaScript applications. Votre gestionnaire de packages permet de s'assurer que les builds sont répétables. Pour ce faire, il enregistre les versions spécifiques de chaque package dont dépend votre application. Il vous permet également de mettre à niveau ces dépendances de manière contrôlée.

Les packages partagés nécessitent une stratégie de test différente. Pour une seule application, il peut être suffisant de déployer l'application dans un environnement de test et de confirmer qu'elle fonctionne toujours. Mais les packages partagés doivent être testés indépendamment de l'application utilisatrice, comme s'ils étaient mis à la disposition du public. (Votre organisation peut choisir de mettre certains packages partagés à la disposition du public.)

N'oubliez pas qu'une construction peut être arbitrairement simple ou complexe. A `Bucket` est une construction, mais `CameraShopWebsite` cela pourrait aussi être une construction.

L'infrastructure et le code d'exécution se trouvent dans le même package

En plus de générer des AWS CloudFormation modèles pour le déploiement de l'infrastructure, il regroupe AWS CDK également les actifs d'exécution tels que les fonctions Lambda et les images Docker et les déploie parallèlement à votre infrastructure. Cela permet de combiner le code qui définit votre infrastructure et le code qui implémente votre logique d'exécution en une seule construction. C'est une bonne pratique de le faire. Ces deux types de code n'ont pas besoin de se trouver dans des référentiels séparés ni même dans des packages distincts.

Pour faire évoluer les deux types de code ensemble, vous pouvez utiliser une construction autonome qui décrit complètement une fonctionnalité, y compris son infrastructure et sa logique. Avec une construction autonome, vous pouvez tester les deux types de code de manière isolée, partager et réutiliser le code entre les projets, et synchroniser les versions de l'ensemble du code.

Élaborez les meilleures pratiques

Cette section contient les meilleures pratiques pour le développement de constructions. Les constructions sont des modules réutilisables et composables qui encapsulent des ressources. Ce sont les éléments de base des AWS CDK applications.

Modélisez avec des constructions, déployez avec des piles

Les piles sont l'unité de déploiement : tous les éléments d'une pile sont déployés ensemble. Ainsi, lorsque vous créez les unités logiques de niveau supérieur de votre application à partir de plusieurs AWS ressources, représentez chaque unité logique sous la forme d'un `Construct`, et non sous la forme d'un. `Stack` Utilisez les piles uniquement pour décrire comment vos constructions doivent être composées et connectées pour vos différents scénarios de déploiement.

Par exemple, si l'une de vos unités logiques est un site Web, les structures qui le composent (comme un compartiment Amazon S3, une API Gateway, des fonctions Lambda ou des tables Amazon RDS)

doivent être composées en une seule construction de haut niveau. Ensuite, cette construction doit être instanciée dans une ou plusieurs piles pour le déploiement.

En utilisant des structures pour la construction et des piles pour le déploiement, vous améliorez le potentiel de réutilisation de votre infrastructure et vous bénéficiez d'une plus grande flexibilité dans la manière dont elle est déployée.

Configuration avec des propriétés et des méthodes, pas avec des variables d'environnement

Les recherches de variables d'environnement dans les constructions et les piles constituent un anti-modèle courant. Les constructions et les piles doivent accepter un objet de propriétés pour permettre une configurabilité complète dans le code. Sinon, cela crée une dépendance à l'égard de la machine sur laquelle le code sera exécuté, ce qui crée encore plus d'informations de configuration que vous devez suivre et gérer.

En général, les recherches de variables d'environnement doivent être limitées au niveau supérieur d'une AWS CDK application. Ils doivent également être utilisés pour transmettre les informations nécessaires à l'exécution dans un environnement de développement. Pour plus d’informations, consultez the section called “Environnements”.

Testez votre infrastructure à l'unité

Pour exécuter de manière cohérente une suite complète de tests unitaires au moment de la création dans tous les environnements, évitez les recherches de réseau lors de la synthèse et modélisez toutes vos étapes de production dans le code. (Ces meilleures pratiques seront abordées plus loin.) Si un seul commit aboutit toujours au même modèle généré, vous pouvez faire confiance aux tests unitaires que vous rédigez pour confirmer que les modèles générés ressemblent à vos attentes. Pour plus d’informations, consultez Constructions de test.

Ne modifiez pas l'ID logique des ressources dynamiques

La modification de l'ID logique d'une ressource entraîne son remplacement par un nouveau lors du prochain déploiement. Pour les ressources dynamiques telles que les bases de données et les compartiments S3, ou pour les infrastructures persistantes telles qu'un Amazon VPC, c'est rarement ce que vous souhaitez. Faites attention à toute refactorisation de votre AWS CDK code susceptible d'entraîner une modification de l'identifiant. Rédigez des tests unitaires qui affirment que les identifiants logiques de vos ressources dynamiques restent statiques. L'identifiant logique est dérivé de celui `id` que vous spécifiez lorsque vous instanciez la construction et de la position de

la construction dans l'arbre de construction. Pour plus d'informations, consultez the section called "Identifiants logiques".

Les constructions ne suffisent pas à assurer la conformité

De nombreuses entreprises clientes écrivent leurs propres enveloppes pour les constructions L2 (les constructions « sélectionnées » qui représentent des AWS ressources individuelles avec des valeurs par défaut et des meilleures pratiques intégrées). Ces wrappers appliquent les meilleures pratiques de sécurité telles que le chiffrement statique et les politiques IAM spécifiques. Par exemple, vous pouvez créer un `MyCompanyBucket` que vous utiliserez ensuite dans vos applications à la place de la structure Amazon S3 `Bucket` habituelle. Ce modèle est utile pour définir les directives de sécurité dès le début du cycle de développement logiciel, mais ne vous y fiez pas comme seul moyen d'application.

Utilisez plutôt des AWS fonctionnalités telles que les politiques de contrôle des services et les limites d'autorisation pour renforcer vos garde-fous au niveau de l'organisation. Utilisez the section called "Aspects" des outils tels que CloudFormation Guard pour affirmer les propriétés de sécurité des éléments de l'infrastructure avant le déploiement. Utilisez-le AWS CDK pour ce qu'il fait le mieux.

Enfin, gardez à l'esprit que l'écriture de vos propres constructions « L2+ » peut empêcher vos développeurs de tirer parti de AWS CDK packages tels que AWS Solutions Constructs ou de constructions tierces de Construct Hub. Ces packages sont généralement construits sur AWS CDK des structures standard et ne pourront pas utiliser vos structures d'emballage.

Bonnes pratiques en matière d'applications

Dans cette section, nous expliquons comment écrire vos AWS CDK applications, en combinant des structures pour définir la manière dont vos AWS ressources sont connectées.

Prendre des décisions au moment de la synthèse

Bien qu'ils vous AWS CloudFormation permettent de prendre des décisions au moment du déploiement (en utilisant `Conditions{ Fn::If }`, et`Parameters`) et qu'ils vous AWS CDK donnent un certain accès à ces mécanismes, nous vous déconseillons de les utiliser. Les types de valeurs que vous pouvez utiliser et les types d'opérations que vous pouvez effectuer sur celles-ci sont limités par rapport à ce qui est disponible dans un langage de programmation à usage général.

Essayez plutôt de prendre toutes les décisions, telles que la construction à instancier, dans votre AWS CDK application en utilisant les `if` instructions et les autres fonctionnalités de votre langage

de programmation. Par exemple, un idiome CDK courant, qui consiste à itérer sur une liste et à instancier une construction avec les valeurs de chaque élément de la liste, n'est tout simplement pas possible à l'aide d'expressions. AWS CloudFormation

AWS CloudFormation Traitez-le comme un détail d'implémentation AWS CDK utilisé pour des déploiements cloud robustes, et non comme une langue cible. Vous n'écrivez pas de AWS CloudFormation modèles en TypeScript Python, vous écrivez du code CDK qui est utilisé CloudFormation pour le déploiement.

Utiliser les noms de ressources générés, pas les noms physiques

Les noms sont une ressource précieuse. Chaque nom ne peut être utilisé qu'une seule fois. Par conséquent, si vous codez en dur le nom d'une table ou d'un bucket dans votre infrastructure et votre application, vous ne pouvez pas déployer cet élément d'infrastructure deux fois dans le même compte. (Le nom dont nous parlons ici est le nom spécifié, par exemple, par la `bucketName` propriété d'une construction de compartiment Amazon S3.)

Pire encore, vous ne pouvez pas apporter de modifications à la ressource qui nécessitent son remplacement. Si une propriété ne peut être définie que lors de la création d'une ressource, telle que celle `KeySchema` d'une table Amazon DynamoDB, cette propriété est immuable. La modification de cette propriété nécessite une nouvelle ressource. Cependant, la nouvelle ressource doit porter le même nom pour être une véritable ressource de remplacement. Mais elle ne peut pas porter le même nom tant que la ressource existante utilise toujours ce nom.

Une meilleure approche consiste à spécifier le moins de noms possible. Si vous omettez les noms des ressources, ils les AWS CDK généreront pour vous d'une manière qui ne posera aucun problème. Supposons que vous ayez une table en tant que ressource. Vous pouvez ensuite transmettre le nom de la table générée en tant que variable d'environnement dans votre AWS Lambda fonction. Dans votre AWS CDK application, vous pouvez référencer le nom de la table sous la forme`table.tableName`. Vous pouvez également générer un fichier de configuration sur votre instance Amazon EC2 au démarrage ou écrire le nom réel de la table dans le AWS Systems Manager Parameter Store afin que votre application puisse le lire à partir de là.

Si l'endroit où vous en avez besoin est une autre AWS CDK pile, c'est encore plus simple. Supposons qu'une pile définisse la ressource et qu'une autre pile doive l'utiliser, les règles suivantes s'appliquent :

- Si les deux piles se trouvent dans la même AWS CDK application, transmettez une référence entre les deux piles. Par exemple, enregistrez une référence à la construction de la ressource en tant

qu'attribut de la stack (`this.stack.uploadBucket = myBucket`) de définition. Transmettez ensuite cet attribut au constructeur de la pile qui a besoin de la ressource.

- Lorsque les deux piles se trouvent dans des AWS CDK applications différentes, utilisez une `from` méthode statique pour utiliser une ressource définie de l'extérieur en fonction de son ARN, de son nom ou d'autres attributs. (Par exemple, utilisez-le `Table.fromArn()` pour une table DynamoDB). Utilisez la `CfnOutput` construction pour imprimer l'ARN ou toute autre valeur requise dans la sortie `decdk deploy`, ou pour consulter le AWS Management Console. La deuxième application peut également lire le CloudFormation modèle généré par la première application et récupérer cette valeur dans la `Outputs` section.

Définissez les politiques de suppression et de conservation des journaux

Les AWS CDK tentatives de vous empêcher de perdre des données en appliquant par défaut des politiques qui conservent tout ce que vous créez. Par exemple, la politique de suppression par défaut pour les ressources contenant des données (telles que les compartiments Amazon S3 et les tables de base de données) consiste à ne pas supprimer la ressource lorsqu'elle est supprimée de la pile. Au lieu de cela, la ressource devient orpheline de la pile. De même, le CDK conserve par défaut tous les journaux pour toujours. Dans les environnements de production, ces valeurs par défaut peuvent rapidement entraîner le stockage de grandes quantités de données dont vous n'avez pas réellement besoin, et une AWS facture correspondante.

Réfléchissez bien aux règles que vous souhaitez appliquer à chaque ressource de production et spécifiez-les en conséquence. the section called “Aspects”À utiliser pour valider les politiques de suppression et de journalisation de votre stack.

Séparez votre application en plusieurs piles selon les exigences de déploiement

Il n'existe pas de règle absolue quant au nombre de piles dont votre application a besoin. Vous finirez généralement par baser votre décision sur vos modèles de déploiement. Gardez à l'esprit les consignes suivantes :

- Il est généralement plus simple de conserver autant de ressources que possible dans la même pile, alors gardez-les ensemble, sauf si vous voulez qu'elles soient séparées.
- Envisagez de conserver les ressources avec état (comme les bases de données) dans une pile séparée des ressources sans état. Vous pouvez ensuite activer la protection contre la résiliation

sur la pile dynamique. Ainsi, vous pouvez librement détruire ou créer plusieurs copies de la pile apatride sans risque de perte de données.

- Les ressources dynamiques sont plus sensibles au changement de nom des constructions ; le changement de nom entraîne le remplacement des ressources. Par conséquent, n'imbriquez pas de ressources dynamiques dans des constructions susceptibles d'être déplacées ou renommées (sauf si l'état peut être reconstruit en cas de perte, comme un cache). C'est une autre bonne raison de placer les ressources stateful dans leur propre pile.

Engagez-vous **`cdk.context.json`** à éviter les comportements non déterministes

Le déterminisme est essentiel à la réussite AWS CDK des déploiements. Une AWS CDK application doit avoir essentiellement le même résultat chaque fois qu'elle est déployée dans un environnement donné.

Comme votre AWS CDK application est écrite dans un langage de programmation à usage général, elle peut exécuter du code arbitraire, utiliser des bibliothèques arbitraires et effectuer des appels réseau arbitraires. Par exemple, vous pouvez utiliser un AWS SDK pour récupérer certaines informations de votre AWS compte lors de la synthèse de votre application. Sachez que cela se traduira par des exigences supplémentaires en matière de configuration des informations d'identification, une latence accrue et un risque, aussi minime soit-il, d'échec à chaque exécution`cdk synth`.

Ne modifiez jamais votre AWS compte ou vos ressources pendant la synthèse. La synthèse d'une application ne devrait pas avoir d'effets secondaires. Les modifications de votre infrastructure ne doivent intervenir que pendant la phase de déploiement, une fois le AWS CloudFormation modèle généré. De cette façon, en cas de problème, AWS CloudFormation vous pouvez automatiquement annuler la modification. Pour apporter des modifications difficiles à apporter dans le AWS CDK framework, utilisez des ressources personnalisées pour exécuter du code arbitraire au moment du déploiement.

Même les appels strictement en lecture seule ne sont pas nécessairement sûrs. Réfléchissez à ce qui se passe si la valeur renvoyée par un appel réseau change. Quel aspect de votre infrastructure cela aura-t-il un impact ? Qu'adviendra-t-il des ressources déjà déployées ? Voici deux exemples de situations dans lesquelles une modification soudaine des valeurs peut poser problème.

- Si vous fournissez un Amazon VPC à toutes les zones de disponibilité disponibles dans une région donnée et que le nombre de zones de disponibilité est de deux le jour du déploiement, votre espace IP est divisé en deux. Si une nouvelle zone de disponibilité est AWS lancée le jour suivant, le déploiement suivant tente de diviser votre espace IP en trois, ce qui nécessite de recréer tous les sous-réseaux. Cela ne sera probablement pas possible car vos instances Amazon EC2 sont toujours en cours d'exécution et vous devrez les nettoyer manuellement.
- Si vous recherchez la dernière image de machine Amazon Linux et que vous déployez une instance Amazon EC2, et qu'une nouvelle image est publiée le lendemain, un déploiement ultérieur récupère la nouvelle AMI et remplace toutes vos instances. Ce n'est peut-être pas ce à quoi vous vous attendiez.

Ces situations peuvent être pernicieuses, car le changement peut survenir après AWS des mois, voire des années, de déploiements réussis. Soudainement, vos déploiements échouent « sans aucune raison » et vous avez oublié depuis longtemps ce que vous avez fait et pourquoi.

Heureusement, il AWS CDK inclut un mécanisme appelé fournisseurs de contexte pour enregistrer un instantané des valeurs non déterministes. Cela permet aux futures opérations de synthèse de produire exactement le même modèle que lors de leur premier déploiement. Les seules modifications apportées au nouveau modèle sont celles que vous avez apportées à votre code. Lorsque vous utilisez la `.fromLookup()` méthode d'une construction, le résultat de l'appel est mis en `cdk.context.json` cache. Vous devez le valider dans le contrôle de version avec le reste de votre code pour vous assurer que les futures exécutions de votre application CDK utilisent la même valeur. Le kit d'outils CDK inclut des commandes pour gérer le cache de contexte, afin que vous puissiez actualiser des entrées spécifiques lorsque vous en avez besoin. Pour plus d’informations, consultez the section called “Contexte”.

Si vous avez besoin d'une valeur (provenant AWS ou d'une autre source) pour laquelle il n'existe aucun fournisseur de contexte CDK natif, nous vous recommandons d'écrire un script distinct. Le script doit récupérer la valeur et l'écrire dans un fichier, puis lire ce fichier dans votre application CDK. Exécutez le script uniquement lorsque vous souhaitez actualiser la valeur stockée, et non dans le cadre de votre processus de génération habituel.

Laissez-les AWS CDK gérer les rôles et les groupes de sécurité

Grâce aux méthodes `grant()` pratiques de la bibliothèque de construction AWS CDK, vous pouvez créer des AWS Identity and Access Management rôles qui accordent l'accès à une ressource à une

autre en utilisant des autorisations d'une portée minimale. Par exemple, considérez une ligne comme celle-ci :

```
myBucket.grantRead(myLambda)
```

Cette seule ligne ajoute une politique au rôle de la fonction Lambda (qui est également créée pour vous). Ce rôle et ses politiques comportent plus d'une douzaine de lignes CloudFormation que vous n'avez pas à écrire. Le n' AWS CDK accorde que les autorisations minimales requises pour que la fonction puisse lire depuis le compartiment.

Si vous demandez aux développeurs de toujours utiliser des rôles prédéfinis créés par une équipe de sécurité, le AWS CDK codage devient beaucoup plus compliqué. Vos équipes risquent de perdre beaucoup de flexibilité dans la conception de leurs applications. Une meilleure alternative consiste à utiliser des politiques de contrôle des services et des limites d'autorisation pour s'assurer que les développeurs respectent les règles de sécurité.

Modélisez toutes les étapes de production dans le code

Dans les AWS CloudFormation scénarios traditionnels, votre objectif est de produire un seul artefact paramétré afin qu'il puisse être déployé dans différents environnements cibles après avoir appliqué des valeurs de configuration spécifiques à ces environnements. Dans le CDK, vous pouvez et devez intégrer cette configuration dans votre code source. Créez une pile pour votre environnement de production et créez une pile distincte pour chacune de vos autres étapes. Ensuite, insérez les valeurs de configuration pour chaque pile dans le code. Utilisez des services tels que Secrets Manager et Systems Manager Parameter Store pour les valeurs sensibles que vous ne souhaitez pas enregistrer dans le contrôle de source, en utilisant les noms ou les ARN de ces ressources.

Lorsque vous synthétisez votre application, l'assemblage cloud créé dans le `cdk.out` dossier contient un modèle distinct pour chaque environnement. L'ensemble de votre build est déterministe. Aucune out-of-band modification n'est apportée à votre application, et chaque validation donne toujours exactement le même AWS CloudFormation modèle et les mêmes ressources associées. Cela rend les tests unitaires beaucoup plus fiables.

Mesurez tout

Atteindre l'objectif d'un déploiement continu complet, sans intervention humaine, nécessite un haut niveau d'automatisation. Cette automatisation n'est possible qu'avec une surveillance étendue. Pour mesurer tous les aspects de vos ressources déployées, créez des métriques, des alarmes

et des tableaux de bord. Ne vous limitez pas à mesurer des éléments tels que l'utilisation du processeur et l'espace disque. Enregistrez également les indicateurs de votre activité et utilisez-les pour automatiser les décisions de déploiement, telles que les annulations. La plupart des constructions L2 contiennent des méthodes pratiques pour vous aider à créer des métriques, comme la `metricUserErrors()` méthode de la classe DynamoDB.Table. AWS CDK

AWS CDK référence

Cette section contient des informations de référence concernant les AWS Cloud Development Kit (AWS CDK).

Rubriques

Référence d'API

La référence des API contient des informations sur la bibliothèque AWS Construct et les autres API fournies par le AWS Cloud Development Kit (AWS CDK). La majeure partie de la bibliothèque AWS Construct est contenue dans un seul package appelé par son TypeScript nom :`aws-cdk-lib`. Le nom réel du package varie en fonction de la langue. Des versions distinctes de la référence d'API sont fournies pour chaque langage de programmation pris en charge.

La référence de l'API CDK est organisée en sous-modules. Il existe un ou plusieurs sous-modules pour chacun Service AWS d'entre eux.

Chaque sous-module possède une vue d'ensemble qui inclut des informations sur l'utilisation de ses API. Par exemple, la présentation de S3 montre comment définir le chiffrement par défaut sur un bucket Amazon Simple Storage Service (Amazon S3).

AWS CDK gestion des versions

Cette rubrique fournit des informations de référence sur la manière dont le AWS Cloud Development Kit (AWS CDK) versionnage est géré.

Les numéros de version se composent de trois parties numériques : majeure. mineur. patch, et respectez strictement le modèle de versionnement sémantique. Cela signifie que les modifications majeures apportées aux API stables sont limitées aux versions majeures.

Les versions mineures et les correctifs sont rétrocompatibles. Le code écrit dans une version précédente avec la même version majeure peut être mis à niveau vers une version plus récente au sein de la même version majeure. Il continuera également à se développer et à fonctionner, produisant le même résultat.

Rubriques

AWS CDKCLIcompatibilité

Le AWS CDK CLI est toujours compatible avec les bibliothèques de construction dont le numéro de version est sémantiquement inférieur ou égal. Il est donc toujours prudent de les mettre à niveau AWS CDK CLI dans la même version majeure.

Le n' AWS CDK CLIest pas toujours compatible avec les bibliothèques de construction d'une version sémantiquement supérieure. La compatibilité dépend de l'utilisation de la même version du schéma d'assemblage cloud par les deux composants. Le AWS CDK framework génère un assemblage cloud lors de la synthèse, puis le AWS CDK CLI consomme pour le déploiement. Le schéma qui définit le format de l'assemblage cloud est strictement spécifié et versionné.

AWS les bibliothèques de construction utilisant une version de schéma d'assemblage cloud donnée sont compatibles avec AWS CDK CLI les versions utilisant cette version de schéma ou une version ultérieure. Cela peut inclure des versions antérieures AWS CDK CLI à une version de bibliothèque de construction donnée.

Lorsque la version d'assemblage cloud requise par la bibliothèque de constructions n'est pas compatible avec la version prise en charge par le AWS CDK CLI, vous recevez un message d'erreur du type suivant :

```
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.
```

Pour résoudre cette erreur, mettez à jour le AWS CDK CLI vers une version compatible avec la version d'assemblage cloud requise ou vers la dernière version disponible. L'alternative (rétrograder les modules de la bibliothèque de construction utilisés par votre application) n'est généralement pas recommandée.

> **Note**
>
> Pour plus de détails sur le schéma d'assemblage cloud, consultez la section Gestion des versions de cloud Assembly.

AWS Gestion des versions de la bibliothèque Construct

Les modules de la bibliothèque AWS Construct passent par différentes étapes au fur et à mesure qu'ils passent du concept à l'API mature. Les différentes étapes offrent différents degrés de stabilité de l'API dans les versions suivantes du AWS CDK.

Les API de la AWS CDK bibliothèque principale sont stables et la bibliothèque est entièrement versionnée sémantiquement. `aws-cdk-lib` Ce package inclut des constructions AWS CloudFormation (L1) pour tous les AWS services et tous les modules stables de niveau supérieur (L2 et L3). (Il inclut également les classes CDK de base telles que `App` et`Stack`). Les API ne seront pas supprimées de ce package (bien qu'elles puissent être obsolètes) avant la prochaine version majeure du CDK. Aucune API individuelle ne subira de modifications majeures. Lorsqu'une modification radicale est requise, une toute nouvelle API sera ajoutée.

Les nouvelles API en cours de développement pour un service déjà intégré `aws-cdk-lib` sont identifiées à l'aide d'un `Beta`*`N`* suffixe, qui `N` commence à 1 et est incrémenté à chaque modification importante apportée à la nouvelle API. `Beta`*`N`* Les API ne sont jamais supprimées, elles sont uniquement déconseillées, de sorte que votre application existante continue de fonctionner avec les nouvelles versions de. `aws-cdk-lib` Lorsque l'API est jugée stable, une nouvelle API sans `Beta`*`N`* suffixe est ajoutée.

Lorsque des API de niveau supérieur (L2 ou L3) commencent à être développées pour un AWS service qui ne possédait auparavant que des API L1, ces API sont initialement distribuées dans un package distinct. Le nom d'un tel package possède le suffixe « Alpha », et sa version correspond à la première version compatible avec une `alpha` sous-version. `aws-cdk-lib` Lorsque le module prend en charge les cas d'utilisation prévus, ses API sont ajoutées`aws-cdk-lib`.

Stabilité des liaisons linguistiques

Au fil du temps, nous pourrons ajouter la prise en charge AWS CDK de quatre langages de programmation supplémentaires. Bien que l'API décrite dans toutes les langues soit la même, la façon dont l'API est exprimée varie selon la langue et peut changer à mesure que le support

linguistique évolue. Pour cette raison, les liaisons linguistiques sont considérées comme expérimentales pendant un certain temps jusqu'à ce qu'elles soient considérées comme prêtes à être utilisées en production.

Language	Stability
TypeScript	Stable
JavaScript	Stable
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK tutoriels

Cette section contient des didacticiels pour AWS Cloud Development Kit (AWS CDK).

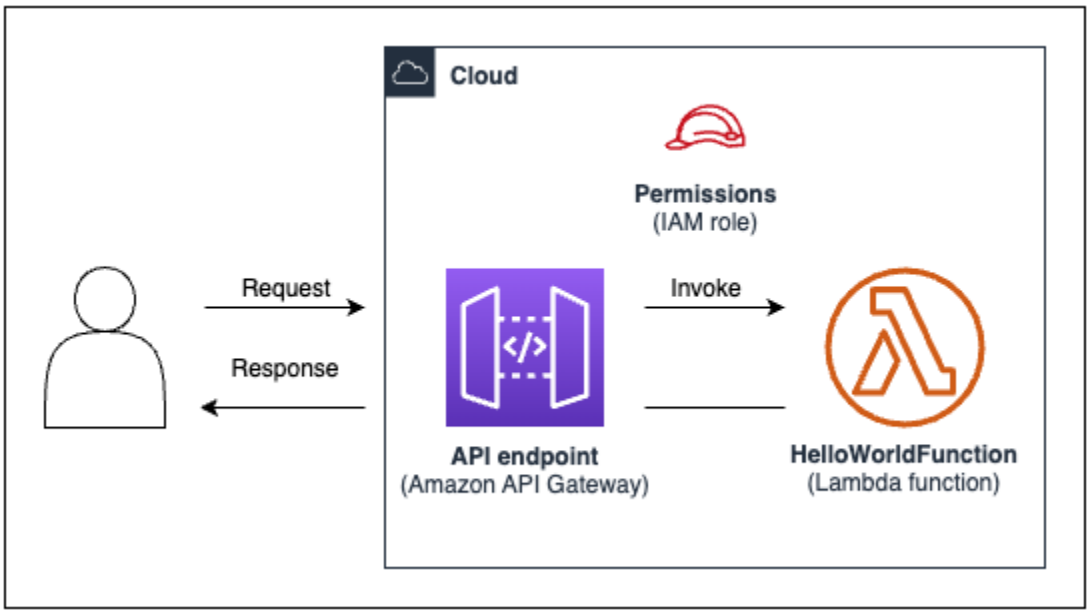
Rubriques

Création d'une application Hello World sans serveur

Dans ce didacticiel, vous allez utiliser le AWS Cloud Development Kit (AWS CDK) pour créer une Hello World application sans serveur simple qui implémente un backend d'API de base en créant ce qui suit :

- Amazon API Gateway REST API — Fournit un point de terminaison HTTP qui est utilisé pour appeler votre fonction par le biais d'une HTTP GET requête.
- AWS Lambda function — Fonction qui renvoie un `Hello World!` message lorsqu'elle est invoquée avec le HTTP point de terminaison.
- Intégrations et autorisations : détails de configuration et autorisations permettant à vos ressources d'interagir entre elles et d'effectuer des actions, telles que la rédaction de journaux sur Amazon CloudWatch.

Le diagramme suivant montre les composants de cette application :



Dans le cadre de ce didacticiel, vous allez effectuer les opérations suivantes :

1. Créez un AWS CDK projet.
2. Définissez une fonction Lambda et une API REST API Gateway à l'aide des constructions L2 de la bibliothèque Construct. AWS
3. Déployez votre application sur le AWS Cloud.
4. Interagissez avec votre application dans le AWS Cloud.
5. Supprimer l'exemple d'application du AWS Cloud.

Prérequis

Avant de commencer ce didacticiel, exécutez les tâches suivantes :

- Créez un Compte AWS et faites installer et configurer le AWS Command Line Interface (AWS CLI).
- Installez Node.js etnpm.
- Installez le kit d'outils CDK globalement, en utilisant`npm install -g aws-cdk`.

Pour plus d’informations, consultez Commencer à utiliser le AWS CDK.

Nous recommandons également une compréhension de base des éléments suivants :

- Qu'est-ce que c'est AWS CDK ?pour une introduction de base au AWS CDK.

- AWS CDK conceptspour un aperçu des concepts fondamentaux du AWS CDK.

Étape 1 : Création d'un projet CDK

Au cours de cette étape, vous créez un nouveau projet CDK à l'aide de la AWS CDK CLI `cdk init` commande.

Pour créer un projet CDK

1. À partir du répertoire de départ de votre choix, créez et accédez à un répertoire de projet nommé `cdk-hello-world` sur votre machine :

   ```
   $ mkdir cdk-hello-world && cd cdk-hello-world
   ```

2. Utilisez la `cdk init` commande pour créer un nouveau projet dans votre langage de programmation préféré :

   TypeScript

   ```
   $ cdk init --language typescript
   ```

   Installez AWS CDK les bibliothèques :

   ```
   $ npm install aws-cdk-lib constructs
   ```

   JavaScript

   ```
   $ cdk init --language javascript
   ```

   Installez AWS CDK les bibliothèques :

   ```
   $ npm install aws-cdk-lib constructs
   ```

   Python

   ```
   $ cdk init --language python
   ```

   Activez l'environnement virtuel :

```
$ source .venv/bin/activate
```

Installez AWS CDK les bibliothèques et les dépendances du projet :

```
(.venv)$ python3 -m pip install -r requirements.txt
```

Java

```
$ cdk init --language java
```

Installez AWS CDK les bibliothèques et les dépendances du projet :

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

Installez AWS CDK les bibliothèques et les dépendances du projet :

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

Installez les dépendances du projet :

```
$ go mod tidy
```

Le CDK CLI crée un projet avec la structure suivante :

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
```

Le CDK crée CLI automatiquement une application CDK contenant une seule pile. L'instance de l'application CDK est créée à partir de la App classe. Voici une partie de votre fichier de candidature CDK :

TypeScript

Situé dans `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

Situé dans `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

Situé dans `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

Situé dans `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#

Situé dans `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

Situé dans `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
}
```

Étape 2 : Création de votre fonction Lambda

Dans votre projet CDK, créez un `lambda` répertoire contenant un nouveau `hello.js` fichier. Voici un exemple :

TypeScript

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Les éléments suivants devraient maintenant être ajoutés à votre projet CDK :

```
cdk-hello-world
### lambda
    ### hello.js
```

JavaScript

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Les éléments suivants devraient maintenant être ajoutés à votre projet CDK :

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mkdir lambda && cd lambda
```

```
$ touch hello.js
```

Les éléments suivants devraient maintenant être ajoutés à votre projet CDK :

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

Les éléments suivants devraient maintenant être ajoutés à votre projet CDK :

```
cdk-hello-world
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Les éléments suivants devraient maintenant être ajoutés à votre projet CDK :

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Les éléments suivants devraient maintenant être ajoutés à votre projet CDK :

```
cdk-hello-world
### lambda
    ### hello.js
```

> **Note**
>
> Pour simplifier ce didacticiel, nous utilisons une fonction JavaScript Lambda pour tous les langages de programmation CDK.

Définissez votre fonction Lambda en ajoutant ce qui suit au fichier nouvellement créé :

```
exports.handler = async (event) => {
    return {
        statusCode: 200,
        headers: { "Content-Type": "text/plain" },
        body: JSON.stringify({ message: "Hello, World!" }),
    };
};
```

Étape 3 : Définissez vos constructions

Au cours de cette étape, vous allez définir vos ressources Lambda et API Gateway à l'aide de constructions AWS CDK L2.

Ouvrez le fichier de projet qui définit votre pile de CDK. Vous allez modifier ce fichier pour définir vos constructions. Voici un exemple de votre fichier de pile de départ :

TypeScript

Situé dans `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

Situé dans `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

Situé dans `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

Situé dans `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#

Situé dans `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

Situé à l'adresse `cdk-hello-world.go` suivante :

```
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
}
```

Dans ce fichier, AWS CDK il effectue les opérations suivantes :

- Votre instance de stack CDK est instanciée à partir de la classe. `Stack`
- La classe `Constructs` de base est importée et fournie en tant que portée ou parent de votre instance de stack.

Définissez votre ressource de fonction Lambda

Pour définir votre ressource de fonction Lambda, vous devez importer et utiliser la construction `aws-lambda` L2 depuis la AWS bibliothèque de constructions.

Modifiez votre fichier de pile comme suit :

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**
>
> Nous importons le `aws_lambda` module `_lambda` car il `lambda` s'agit d'un identifiant intégré dans. 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
}

// ...
```

Ici, vous créez une ressource de fonction Lambda et définissez les propriétés suivantes :

- `runtime`— L'environnement dans lequel la fonction s'exécute. Ici, nous utilisons Node.js la version20.x.
- `code`— Le chemin d'accès au code de fonction sur votre machine locale.
- `handler`— Le nom du fichier spécifique qui contient votre code de fonction.

Définissez votre REST API ressource API Gateway

Pour définir votre REST API ressource API Gateway, vous devez importer et utiliser la construction `aws-apigateway` L2 depuis la bibliothèque AWS Construct.

Modifiez votre fichier de pile comme suit :

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
}

// ...
```

Ici, vous créez une REST API ressource API Gateway, ainsi que les éléments suivants :

- Une intégration entre REST API et votre fonction Lambda, permettant à l'API d'appeler votre fonction. Cela inclut la création d'une ressource d'autorisation Lambda.
- Nouveau nom `hello` de ressource ou de chemin ajouté à la racine du point de terminaison de l'API. Cela crée un nouveau point de terminaison qui `/hello` s'ajoute à votre baseURL.
- Méthode GET pour la `hello` ressource. Lorsqu'une requête GET est envoyée au point de `/hello` terminaison, la fonction Lambda est invoquée et sa réponse est renvoyée.

Étape 4 : préparer votre application pour le déploiement

Au cours de cette étape, vous préparez votre application pour le déploiement en créant, si nécessaire, et en effectuant une validation de base à l'aide de la AWS CDK CLI `cdk synth` commande.

Si nécessaire, créez votre application :

TypeScript

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ npm run build
```

JavaScript

Il n'est pas nécessaire de construire.

Python

Il n'est pas nécessaire de construire.

Java

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ mvn package
```

C#

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ dotnet build src
```

Go

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ go build
```

Exécutez `cdk synth` pour synthétiser un AWS CloudFormation modèle à partir de votre code CDK. En utilisant des constructions L2, de nombreux détails de configuration requis AWS CloudFormation

pour faciliter l'interaction entre votre fonction Lambda et vous REST API sont fournis par le. AWS CDK

À partir de la racine de votre projet, exécutez ce qui suit :

```
$ cdk synth
```

> **Note**
>
> Si le message d'erreur suivant s'affiche, vérifiez que vous êtes bien dans le `cdk-hello-world` répertoire et réessayez :
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

En cas de succès, le AWS CloudFormation modèle AWS CDK CLI sera affiché au YAML format à l'invite de commande. Un modèle JSON formaté est également enregistré dans le `cdk.out` répertoire.

Voici un exemple de sortie du AWS CloudFormation modèle :

AWS CloudFormation modèle

```
Resources:
  HelloWorldFunctionServiceRoleunique-identifier:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
        Version: "2012-10-17"
      ManagedPolicyArns:
        - Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Metadata:
```

```
      aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/ServiceRole/Resource
  HelloWorldFunctionunique-identifier:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        S3Bucket:
          Fn::Sub: cdk-unique-identifier-assets-${AWS::AccountId}-${AWS::Region}
        S3Key: unique-identifier.zip
      Handler: hello.handler
      Role:
        Fn::GetAtt:
          - HelloWorldFunctionServiceRoleunique-identifier
          - Arn
      Runtime: nodejs20.x
    DependsOn:
      - HelloWorldFunctionServiceRoleunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/Resource
      aws:asset:path: asset.unique-identifier
      aws:asset:is-bundled: false
      aws:asset:property: Code
  HelloWorldApiunique-identifier:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: HelloWorldApi
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Resource
  HelloWorldApiDeploymentunique-identifier:
    Type: AWS::ApiGateway::Deployment
    Properties:
      Description: Automatically created by the RestApi construct
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    DependsOn:
      - HelloWorldApihelloGETunique-identifier
      - HelloWorldApihellounique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Deployment/Resource
  HelloWorldApiDeploymentStageprod012345ABC:
    Type: AWS::ApiGateway::Stage
    Properties:
      DeploymentId:
        Ref: HelloWorldApiDeploymentunique-identifier
      RestApiId:
```

```
        Ref: HelloWorldApiunique-identifier
      StageName: prod
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/DeploymentStage.prod/Resource
  HelloWorldApihellounique-identifier:
    Type: AWS::ApiGateway::Resource
    Properties:
      ParentId:
        Fn::GetAtt:
          - HelloWorldApiunique-identifier
          - RootResourceId
      PathPart: hello
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/Resource
  HelloWorldApihelloGETApiPermissionCdkHelloWorldStackHelloWorldApiunique-identifier:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunctionunique-identifier
          - Arn
      Principal: apigateway.amazonaws.com
      SourceArn:
        Fn::Join:
          - ""
          - - "arn:"
            - Ref: AWS::Partition
            - ":execute-api:"
            - Ref: AWS::Region
            - ":"
            - Ref: AWS::AccountId
            - ":"
            - Ref: HelloWorldApi9E278160
            - /
            - Ref: HelloWorldApiDeploymentStageprodunique-identifier
            - /GET/hello
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/
ApiPermission.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello
  HelloWorldApihelloGETApiPermissionTestCdkHelloWorldStackHelloWorldApiunique-
identifier:
```

```
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunctionunique-identifier
          - Arn
      Principal: apigateway.amazonaws.com
      SourceArn:
        Fn::Join:
          - ""
          - - "arn:"
            - Ref: AWS::Partition
            - ":execute-api:"
            - Ref: AWS::Region
            - ":"
            - Ref: AWS::AccountId
            - ":"
            - Ref: HelloWorldApiunique-identifier
            - /test-invoke-stage/GET/hello
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/
ApiPermission.Test.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello
  HelloWorldApihelloGETunique-identifier:
    Type: AWS::ApiGateway::Method
    Properties:
      AuthorizationType: NONE
      HttpMethod: GET
      Integration:
        IntegrationHttpMethod: POST
        Type: AWS_PROXY
        Uri:
          Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - ":apigateway:"
              - Ref: AWS::Region
              - :lambda:path/2015-03-31/functions/
              - Fn::GetAtt:
                  - HelloWorldFunctionunique-identifier
                  - Arn
              - /invocations
      ResourceId:
```

```
        Ref: HelloWorldApihellounique-identifier
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/Resource
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Analytics: v2:deflate64:unique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/CDKMetadata/Default
    Condition: CDKMetadataAvailable
Outputs:
  HelloWorldApiEndpointunique-identifier:
    Value:
      Fn::Join:
        - ""
        - - https://
          - Ref: HelloWorldApiunique-identifier
          - .execute-api.
          - Ref: AWS::Region
          - "."
          - Ref: AWS::URLSuffix
          - /
          - Ref: HelloWorldApiDeploymentStageprodunique-identifier
          - /
Conditions:
  CDKMetadataAvailable:
    Fn::Or:
      - Fn::Or:
          - Fn::Equals:
              - Ref: AWS::Region
              - af-south-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-east-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-northeast-1
          - Fn::Equals:
              - Ref: AWS::Region
              - ap-northeast-2
          - Fn::Equals:
              - Ref: AWS::Region
```

```
            - ap-south-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-southeast-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-southeast-2
        - Fn::Equals:
            - Ref: AWS::Region
            - ca-central-1
        - Fn::Equals:
            - Ref: AWS::Region
            - cn-north-1
        - Fn::Equals:
            - Ref: AWS::Region
            - cn-northwest-1
    - Fn::Or:
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-central-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-north-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-south-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-west-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-west-2
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-west-3
        - Fn::Equals:
            - Ref: AWS::Region
            - il-central-1
        - Fn::Equals:
            - Ref: AWS::Region
            - me-central-1
        - Fn::Equals:
            - Ref: AWS::Region
            - me-south-1
```

```
          - Fn::Equals:
              - Ref: AWS::Region
              - sa-east-1
      - Fn::Or:
          - Fn::Equals:
              - Ref: AWS::Region
              - us-east-1
          - Fn::Equals:
              - Ref: AWS::Region
              - us-east-2
          - Fn::Equals:
              - Ref: AWS::Region
              - us-west-1
          - Fn::Equals:
              - Ref: AWS::Region
              - us-west-2
Parameters:
  BootstrapVersion:
    Type: AWS::SSM::Parameter::Value<String>
    Default: /cdk-bootstrap/hnb659fds/version
    Description: Version of the CDK Bootstrap resources in this environment,
 automatically retrieved from SSM Parameter Store. [cdk:skip]
Rules:
  CheckBootstrapVersion:
    Assertions:
      - Assert:
          Fn::Not:
            - Fn::Contains:
                - - "1"
                  - "2"
                  - "3"
                  - "4"
                  - "5"
                - Ref: BootstrapVersion
        AssertDescription: CDK bootstrap stack version 6 required. Please run 'cdk
 bootstrap' with a recent version of the CDK CLI.
```

En utilisant des constructions L2, vous définissez quelques propriétés pour configurer vos ressources et utilisez des méthodes d'assistance pour les intégrer ensemble. Il AWS CDK configure la majorité de vos AWS CloudFormation ressources et propriétés nécessaires au provisionnement de votre application.

Étape 5 : déployer votre application

Au cours de cette étape, vous devez utiliser la AWS CDK CLI `cdk deploy` commande pour déployer votre application. Il AWS CDK travaille avec le AWS CloudFormation service pour fournir vos ressources.

> **Important**
>
> Vous devez effectuer un démarrage unique de votre AWS environnement avant le déploiement. Pour obtenir des instructions, consultez Bootstrap your environment.

À partir de la racine de votre projet, exécutez ce qui suit. Confirmez les modifications si vous y êtes invité :

```
$ cdk deploy

#  Synthesis time: 2.44s

...

Do you wish to deploy these changes (y/n)? y
```

Une fois le déploiement terminé, l'URL de votre point de terminaison AWS CDK CLI sera affichée. Copiez cette URL pour l'étape suivante. Voici un exemple :

```
...
#  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
...
```

Étape 6 : Interagissez avec votre application

Au cours de cette étape, vous lancez une requête GET à destination de votre point de terminaison d'API et recevez la réponse de votre fonction Lambda.

Localisez l'URL de votre point de terminaison à l'étape précédente et ajoutez le `/hello` chemin. Ensuite, à l'aide de votre navigateur ou de votre invite de commande, envoyez une requête GET à votre terminal. Voici un exemple :

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello
{"message":"Hello World!"}%
```

Félicitations, vous avez créé, déployé et interagi avec succès avec votre application à l'aide du AWS CDK !

Étape 7 : Supprimer votre application

Au cours de cette étape, vous devez AWS CDK CLI utiliser le pour supprimer votre application du AWS Cloud.

Pour supprimer votre application, exécutez`cdk destroy`. Lorsque vous y êtes invité, confirmez votre demande de suppression de l'application :

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
 #  CdkHelloWorldStack: destroyed
```

Résolution des problèmes

Erreur : {« message » : « Erreur interne du serveur »} %

Lorsque vous appelez la fonction Lambda déployée, vous recevez cette erreur. Cette erreur peut se produire pour plusieurs raisons.

Pour résoudre d'autres problèmes

Utilisez le AWS CLI pour appeler votre fonction Lambda.

1. Modifiez votre fichier de pile pour capturer la valeur de sortie du nom de votre fonction Lambda déployée. Voici un exemple :

```
...

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. Déployez à nouveau votre application. La valeur du nom de votre fonction Lambda déployée AWS CDK CLI sera affichée :

```
$ cdk deploy

#  Synthesis time: 0.29s
...
 #  CdkHelloWorldStack

#  Deployment time: 20.36s

Outputs:
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. Utilisez le AWS CLI pour appeler votre fonction Lambda dans le AWS Cloud et afficher la réponse dans un fichier texte :

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-
identifier output.txt
```

4. Vérifiez `output.txt` vos résultats.

Cause possible : la ressource API Gateway n'est pas définie correctement dans votre fichier de pile.

Si `output.txt` la réponse de la fonction Lambda est réussie, le problème vient peut-être de la façon dont vous avez défini votre API REST API Gateway. AWS CLI Invoque votre Lambda directement, et non via votre point de terminaison. Vérifiez votre code pour vous assurer qu'il correspond à ce didacticiel. Ensuite, déployez à nouveau.

Cause possible : la ressource Lambda n'est pas définie correctement dans votre fichier de pile.

S'il `output.txt` renvoie une erreur, le problème peut être lié à la façon dont vous avez défini votre fonction Lambda. Vérifiez votre code pour vous assurer qu'il correspond à ce didacticiel. Déployez ensuite à nouveau.

Créez une application avec plusieurs piles

Vous pouvez créer une AWS Cloud Development Kit (AWS CDK) application contenant plusieurs piles. Lorsque vous déployez l' AWS CDK application, chaque pile devient son propre AWS CloudFormation modèle. Vous pouvez également synthétiser et déployer chaque pile individuellement à l'aide de la AWS CDK CLI `cdk deploy` commande.

Ce didacticiel couvre les sujets suivants :

- Comment étendre la `Stack` classe pour accepter de nouvelles propriétés ou de nouveaux arguments.
- Comment utiliser les propriétés pour déterminer les ressources contenues dans la pile et leur configuration.
- Comment instancier plusieurs piles à partir de cette classe.

L'exemple présenté dans cette rubrique utilise une propriété booléenne nommée (`encryptBucket`Python :`encrypt_bucket`). Il indique si un compartiment Amazon S3 doit être chiffré. Si tel est le cas, la pile active le chiffrement à l'aide d'une clé gérée par AWS Key Management Service (AWS KMS). L'application crée deux instances de cette pile, l'une avec chiffrement et l'autre sans chiffrement.

Rubriques

- Avant de commencer

- Ajouter un paramètre facultatif
- Définissez la classe de pile
- Créez deux instances de pile
- Synthétiser et déployer la pile
- Nettoyage

Avant de commencer

Installez d'abord Node.js et les outils de ligne de AWS CDK commande, si ce n'est pas déjà fait. Consultez Commencer à utiliser le AWS CDK pour plus de détails.

Créez ensuite un AWS CDK projet en saisissant les commandes suivantes sur la ligne de commande.

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
```

Vous pouvez importer le projet Maven obtenu dans votre IDE Java.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

Vous pouvez ouvrir le fichier `src/Pipeline.sln` dans Visual Studio.

Ajouter un paramètre facultatif

L'`props`argument du `Stack` constructeur remplit l'interface. `StackProps` Dans cet exemple, nous voulons que la pile accepte une propriété supplémentaire nous indiquant s'il faut chiffrer le compartiment Amazon S3. Nous devons créer une interface ou une classe qui inclut la propriété. Cela permet au compilateur de s'assurer que la propriété possède une valeur booléenne et d'activer l'autocomplétion pour celle-ci dans votre IDE.

Ouvrez donc le fichier source indiqué dans votre IDE ou votre éditeur et ajoutez la nouvelle interface, classe ou argument. Le code devrait ressembler à ceci après les modifications. Les lignes que nous avons ajoutées apparaissent en gras.

TypeScript

Dossier : `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

Dossier : `lib/multistack-stack.js`

JavaScript n'a pas de fonctionnalité d'interface ; nous n'avons pas besoin d'ajouter de code.

```
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

Dossier : `multistack/multistack_stack.py`

Python ne possède pas de fonctionnalité d'interface, nous allons donc étendre notre pile pour accepter la nouvelle propriété en ajoutant un argument de type mot-clé.

```
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

Dossier : `src/main/java/com/myorg/MultistackStack.java`

C'est plus compliqué que ce que nous voulons vraiment aborder d'étendre un type d'accessoire en Java. Écrivez plutôt le constructeur de la pile pour accepter un paramètre booléen facultatif. Comme `props` il s'agit d'un argument facultatif, nous allons écrire un constructeur supplémentaire qui vous permettra de l'ignorer. Il sera défini par défaut sur`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#

Dossier : `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 nouvelle propriété est facultative. Si `encryptBucket` (Python :`encrypt_bucket`) n'est pas présent, sa valeur est `undefined` ou son équivalent local. Le bucket ne sera pas chiffré par défaut.

Définissez la classe de pile

Définissons maintenant notre classe de pile en utilisant notre nouvelle propriété. Faites en sorte que le code ressemble à ce qui suit. Le code que vous devez ajouter ou modifier apparaît en gras.

TypeScript

Dossier : `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

Dossier : `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

Dossier : `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

Dossier : `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#

Dossier : `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
                });
            }
        }
    }
}
```

Créez deux instances de pile

Nous allons maintenant ajouter le code pour instancier deux piles distinctes. Comme précédemment, les lignes de code indiquées en gras sont celles que vous devez ajouter. Supprimez la `MultistackStack` définition existante.

TypeScript

Dossier : `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

Dossier : `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

Dossier : `./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

Dossier : `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#

Fichier : 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();
        }
    }
}
```

Ce code utilise la nouvelle propriété `encryptBucket` (Python :`encrypt_bucket`) de la `MultistackStack` classe pour instancier ce qui suit :

- Une pile avec un compartiment Amazon S3 chiffré dans la `us-east-1` AWS région.
- Une pile contenant un compartiment Amazon S3 non chiffré dans la `us-west-1` AWS région.

Synthétiser et déployer la pile

Vous pouvez désormais déployer des stacks depuis l'application. Tout d'abord, synthétisez un AWS CloudFormation modèle pour `MyEastCdkStack` —stack in`us-east-1`. Il s'agit de la pile contenant le compartiment S3 chiffré.

```
$ cdk synth MyEastCdkStack
```

Pour déployer cette pile sur votre AWS compte, exécutez l'une des commandes suivantes. La première commande utilise votre AWS profil par défaut pour obtenir les informations d'identification nécessaires au déploiement de la pile. Le second utilise un profil que vous spécifiez. Remplacez *`PROFILE_NAME`* par le nom d'un AWS CLI profil contenant les informations d'identification appropriées pour le déploiement dans la `us-east-1` AWS région.

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

Nettoyage

Pour éviter de facturer les ressources que vous avez déployées, détruisez la pile à l'aide de la commande suivante.

```
cdk destroy MyEastCdkStack
```

L'opération de destruction échoue si quelque chose est stocké dans le compartiment de la pile. Cela ne devrait pas être le cas si vous avez uniquement suivi les instructions de cette rubrique. Mais si vous avez mis quelque chose dans le compartiment, vous devez supprimer le contenu du compartiment avant de détruire la pile. (Ne supprimez pas le compartiment lui-même.) Utilisez le AWS Management Console ou AWS CLI pour supprimer le contenu du compartiment.

Exemples

Cette rubrique contient les exemples suivants :

- Création d'une application Hello World sans serveurCrée une application sans serveur à l'aide de Lambda, d'API Gateway et d'Amazon S3.
- Création d'un service AWS Fargate à l'aide du AWS CDKCrée un service Amazon ECS Fargate à partir d'une image sur. DockerHub

Création d'un service AWS Fargate à l'aide du AWS CDK

Cet exemple explique comment créer un service AWS Fargate exécuté sur un cluster Amazon Elastic Container Service (Amazon ECS) dirigé par un Application Load Balancer connecté à Internet à partir d'une image sur Amazon ECR.

Amazon ECS est un service de gestion de conteneurs hautement évolutif et rapide, qui permet d'exécuter, d'arrêter et de gérer facilement des conteneurs Docker sur un cluster. Vous pouvez héberger votre cluster sur une infrastructure sans serveur gérée par Amazon ECS en lançant vos services ou tâches à l'aide du type de lancement Fargate. Pour plus de contrôle, vous pouvez héberger vos tâches sur un cluster d'instances Amazon Elastic Compute Cloud (Amazon EC2) que vous gérez à l'aide du type de lancement Amazon EC2.

Ce didacticiel explique comment lancer certains services à l'aide du type de lancement Fargate. Si vous avez utilisé le AWS Management Console pour créer un service Fargate, vous savez qu'il existe de nombreuses étapes à suivre pour accomplir cette tâche. AWS propose plusieurs didacticiels et rubriques de documentation qui vous guident dans la création d'un service Fargate, notamment :

- Comment déployer des conteneurs Docker - AWS
- Configuration avec Amazon ECS
- Commencer à utiliser Amazon ECS à l'aide de Fargate

Cet exemple crée un service Fargate similaire dans le code. AWS CDK

La structure Amazon ECS utilisée dans ce didacticiel vous aide à utiliser AWS les services en offrant les avantages suivants :

- Configure automatiquement un équilibreur de charge.

- Ouvre automatiquement un groupe de sécurité pour les équilibreurs de charge. Cela permet aux équilibreurs de charge de communiquer avec les instances sans que vous ne créiez explicitement de groupe de sécurité.
- Ordonne automatiquement la dépendance entre le service et l'équilibreur de charge attaché à un groupe cible, où il AWS CDK applique l'ordre correct de création de l'écouteur avant la création d'une instance.
- Configure automatiquement les données utilisateur pour dimensionner automatiquement les groupes. Cela crée la configuration correcte pour associer un cluster aux AMI.
- Valide les combinaisons de paramètres à un stade précoce. Cela permet de détecter les AWS CloudFormation problèmes plus tôt, ce qui vous permet de gagner du temps de déploiement. Par exemple, en fonction de la tâche, il est facile de mal configurer les paramètres de mémoire. Auparavant, vous ne rencontriez aucune erreur tant que vous n'aviez pas déployé votre application. Mais maintenant, ils AWS CDK peuvent détecter une mauvaise configuration et émettre une erreur lorsque vous synthétisez votre application.
- Ajoute automatiquement des autorisations pour Amazon Elastic Container Registry (Amazon ECR) si vous utilisez une image provenant d'Amazon ECR.
- Échelle automatiquement. Il AWS CDK fournit une méthode qui vous permet de dimensionner automatiquement les instances lorsque vous utilisez un cluster Amazon EC2. Cela se produit automatiquement lorsque vous utilisez une instance dans un cluster Fargate.

  En outre, cela AWS CDK empêche la suppression d'une instance lorsque le dimensionnement automatique tente de tuer une instance, alors qu'une tâche est en cours d'exécution ou est planifiée sur cette instance.

  Auparavant, vous deviez créer une fonction Lambda pour bénéficier de cette fonctionnalité.
- Assure le support des actifs, afin que vous puissiez déployer une source depuis votre machine vers Amazon ECS en une seule étape. Auparavant, pour utiliser une source d'application, vous deviez effectuer plusieurs étapes manuelles, telles que le téléchargement sur Amazon ECR et la création d'une image Docker.

Consultez ECS pour plus de détails.

> **Important**
>
> Les `ApplicationLoadBalancedFargateService` concepts que nous utiliserons incluent de nombreux AWS composants, dont certains ont des coûts non négligeables s'ils sont

laissés provisionnés dans votre AWS compte, même si vous ne les utilisez pas. Assurez-vous de nettoyer (cdk destroy) après avoir terminé cet exemple.

Création du répertoire et initialisation du AWS CDK

Commençons par créer un répertoire contenant le AWS CDK code, puis créer une AWS CDK application dans ce répertoire.

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
```

Vous pouvez maintenant importer le projet Maven dans votre IDE.

C#

```
mkdir MyEcsConstruct
```

```
cd MyEcsConstruct
cdk init --language csharp
```

Vous pouvez désormais l'ouvrir `src/MyEcsConstruct.sln` dans Visual Studio.

Lancez l'application et vérifiez qu'elle crée une pile vide.

```
cdk synth
```

Création d'un service Fargate

Il existe deux manières différentes d'exécuter vos tâches de conteneur avec Amazon ECS :

- Utilisez le type de `Fargate` lancement, dans lequel Amazon ECS gère pour vous les machines physiques sur lesquelles vos conteneurs s'exécutent.
- Utilisez le type de `EC2` lancement, dans lequel vous effectuez la gestion, par exemple en spécifiant le dimensionnement automatique.

Dans cet exemple, nous allons créer un service Fargate exécuté sur un cluster ECS dirigé par un Application Load Balancer connecté à Internet.

Ajoutez les importations du module AWS Construct Library suivantes au fichier indiqué.

TypeScript

Dossier : `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

Dossier : `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

Dossier : `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

Dossier : `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#

Dossier : `src/MyEcsConstruct/MyEcsConstructStack.cs`

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

Remplacez le commentaire à la fin du constructeur par le code suivant.

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
                }
            );
```

Enregistrez-le et assurez-vous qu'il fonctionne et crée une pile.

```
cdk synth
```

La pile est composée de centaines de lignes, nous ne la montrerons donc pas ici. La pile doit contenir une instance par défaut, un sous-réseau privé et un sous-réseau public pour les trois zones de disponibilité, ainsi qu'un groupe de sécurité.

Déployez la pile.

```
cdk deploy
```

AWS CloudFormation affiche des informations sur les dizaines d'étapes qu'il effectue lors du déploiement de votre application.

C'est à quel point il est facile de créer un service Amazon ECS alimenté par Fargate pour exécuter une image Docker.

Nettoyage

Pour éviter AWS des charges inattendues, détruisez votre AWS CDK pile une fois que vous aurez terminé cet exercice.

```
cdk destroy
```

AWS CDK exemples

Pour plus d'exemples de AWS CDK piles et d'applications dans votre langage de programmation compatible préféré, consultez le référentiel AWS CDK d'exemples sur GitHub.

AWS CDK outils

Cette section contient des informations sur les AWS CDK outils répertoriés ci-dessous.

Rubriques

AWS CDK Boîte à outils (**cdk**commande)

Le AWS CDK Toolkit, la commande CLI`cdk`, est le principal outil d'interaction avec votre AWS CDK application. Il exécute votre application, interroge le modèle d'application que vous avez défini, produit et déploie les AWS CloudFormation modèles générés par le. AWS CDK Il fournit également d'autres fonctionnalités utiles pour créer et travailler avec AWS CDK des projets. Cette rubrique contient des informations sur les cas d'utilisation courants du kit d'outils CDK.

Le AWS CDK Toolkit est installé avec le Node Package Manager. Dans la plupart des cas, nous recommandons de l'installer globalement.

```
npm install -g aws-cdk             # install latest version
npm install -g aws-cdk@X.YY.Z      # install specific version
```

> **Tip**
>
> Si vous travaillez régulièrement avec plusieurs versions du Toolkit AWS CDK, envisagez d'installer une version correspondante du AWS CDK Toolkit dans des projets CDK individuels. Pour ce faire, omettez `-g` de le faire dans la `npm install` commande. Utilisez-le ensuite `npx aws-cdk` pour l'invoquer. Cela exécute la version locale s'il en existe une, puis revient à une version globale dans le cas contraire.

commandes du kit d'outils

Toutes les commandes du CDK Toolkit commencent par`cdk`, qui est suivie d'une sous-commande (`list`,`synthesize`,`deploy`, etc.). Certaines sous-commandes ont une version plus courte

(`lssynth`,, etc.) équivalente. Les options et les arguments suivent la sous-commande dans n'importe quel ordre. Les commandes disponibles sont résumées ici.

Command	Fonction
`cdk list` (`ls`)	Répertorie les piles dans l'application
`cdk synthesize` (`synth`)	Synthèse et imprime le CloudFormation modèle pour une ou plusieurs piles spécifiées
`cdk bootstrap`	Déploie la pile intermédiaire du CDK Toolkit ; voir the section called “Action d'amorçage”
`cdk deploy`	Déploie une ou plusieurs piles spécifiées
`cdk destroy`	Détruit une ou plusieurs piles spécifiées
`cdk diff`	Compare la pile spécifiée et ses dépendances avec les piles déployées ou un modèle local CloudFormation
`cdk import`	Utilise les importations de CloudFormation ressources pour intégrer les ressources existantes dans une pile gérée par CDK
`cdk metadata`	Affiche les métadonnées relatives à la pile spécifiée
`cdk init`	Crée un nouveau projet CDK dans le répertoire actuel à partir d'un modèle spécifié
`cdk context`	Gère les valeurs de contexte mises en cache
`cdk docs` (`doc`)	Ouvre la référence d'API CDK dans votre navigateur
`cdk doctor`	Vérifie l'absence de problèmes potentiels dans votre projet CDK

Pour connaître les options disponibles pour chaque commande, consultezthe section called “Aide intégrée”.

Spécification des options et de leurs valeurs

Les options de ligne de commande commencent par deux tirets ()`--`. Certaines options fréquemment utilisées comportent des synonymes composés d'une seule lettre commençant par un seul trait d'union (par exemple, `--app` contient un synonyme`-a`). L'ordre des options dans une commande du AWS CDK Toolkit n'est pas important.

Toutes les options acceptent une valeur qui doit suivre le nom de l'option. La valeur peut être séparée du nom par un espace blanc ou par un signe égal=. Les deux options suivantes sont équivalentes.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

Certaines options sont des drapeaux (booléens). Vous pouvez spécifier `true` ou `false` comme valeur. Si vous ne fournissez aucune valeur, la valeur est considérée comme étant`true`. Vous pouvez également préfixer le nom de l'option par `no-` « implicitement `false` ».

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

Quelques options, à savoir `--context`--`parameters`,`--plugin`,`--tags`, et`--trust`, peuvent être spécifiées plusieurs fois pour spécifier plusieurs valeurs. Ils sont indiqués comme ayant été saisis dans l'aide du kit `[array]` d'aide du CDK Toolkit. Par exemple :

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

Aide intégrée

La AWS CDK boîte à outils contient une aide intégrée. Vous pouvez obtenir de l'aide générale concernant cet utilitaire ainsi qu'une liste des sous-commandes fournies en émettant :

```
cdk --help
```

Pour obtenir de l'aide concernant une sous-commande en particulier, par exemple`deploy`, spécifiez-la avant le `--help` drapeau.

```
cdk deploy --help
```

Problème `cdk version` d'affichage de la version du AWS CDK kit d'outils. Fournissez ces informations lorsque vous demandez de l'aide.

Rapport sur les versions

Pour mieux comprendre comment le AWS CDK est utilisé, les constructions utilisées par les AWS CDK applications sont collectées et signalées à l'aide d'une ressource identifiée comme`AWS::CDK::Metadata`. Cette ressource est ajoutée aux AWS CloudFormation modèles et peut être facilement consultée. Ces informations peuvent également être utilisées pour identifier les piles AWS à l'aide d'une structure présentant des problèmes de sécurité ou de fiabilité connus. Il peut également être utilisé pour contacter leurs utilisateurs avec des informations importantes.

> **Note**
>
> Avant la version 1.93.0, ils AWS CDK indiquaient les noms et les versions des modules chargés lors de la synthèse, au lieu des constructions utilisées dans la pile.

Par défaut, le AWS CDK signale l'utilisation de constructions dans les modules NPM suivants utilisés dans la pile :

- AWS CDK module de base
- AWS Construire des modules de bibliothèque
- AWS Module Solutions Constructs
- AWS Module du kit de déploiement de Render Farm

La `AWS::CDK::Metadata` ressource ressemble à ce qui suit.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
```

```
  Properties:
    Analytics:
 "v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/
XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz
+AqqLDt2Y3YMnLYWwAAAA="
```

La `Analytics` propriété est une liste gzippée, codée en base64 et codée par préfixe des constructions de la pile.

Pour désactiver le reporting des versions, utilisez l'une des méthodes suivantes :

- Utilisez la cdk commande avec l'--no-version-reportingargument pour désactiver une seule commande.

  ```
  cdk --no-version-reporting synth
  ```

  N'oubliez pas que le AWS CDK kit d'outils synthétise les nouveaux modèles avant le déploiement. Vous devez donc également `--no-version-reporting` ajouter des `cdk deploy` commandes.
- versionReportingRéglé sur false dans `./cdk.json` ou`~/.cdk.json`. Cela permet de se désinscrire, sauf si vous l'acceptez `--version-reporting` en spécifiant une commande individuelle.

  ```
  {
    "app": "...",
    "versionReporting": false
  }
  ```

Authentification avec AWS

Vous pouvez configurer l'accès programmatique aux AWS ressources de différentes manières, en fonction de l'environnement et de l' AWS accès dont vous disposez.

Pour choisir votre méthode d'authentification et la configurer pour le kit d'outils CDK, consultez la section Authentification et accès dans le guide de référence AWS des SDK et des outils.

L'approche recommandée pour les nouveaux utilisateurs qui se développent localement et qui ne disposent pas d'une méthode d'authentification de la part de leur employeur est la mise en place AWS IAM Identity Center. Cette méthode inclut l'installation AWS CLI pour faciliter la configuration et pour vous connecter régulièrement au portail AWS d'accès. Si vous choisissez cette méthode,

votre environnement doit contenir les éléments suivants une fois que vous avez terminé la procédure d'authentification IAM Identity Center décrite dans le Guide de référence AWS des SDK et des outils :

- Le AWS CLI, que vous utilisez pour démarrer une session de portail d' AWS accès avant d'exécuter votre application.
- AWSconfigFichier partagé comportant un `[default]` profil avec un ensemble de valeurs de configuration pouvant être référencées à partir du AWS CDK. Pour connaître l'emplacement de ce fichier, consultez Location of the shared files dans le manuel AWS SDKs and Tools Reference Guide.
- Le `config` fichier partagé définit le `region`paramètre. Cela définit l'utilisation par défaut Région AWS du kit CDK AWS CDK et du kit CDK pour les AWS demandes.
- Le kit d'outils CDK utilise la configuration du fournisseur de jetons SSO du profil pour obtenir des informations d'identification avant d'envoyer des demandes à. AWS La `sso_role_name` valeur, qui est un rôle IAM connecté à un ensemble d'autorisations IAM Identity Center, doit autoriser l'accès à l'utilisateur dans Services AWS votre application.

  Le `config` fichier d'exemple suivant montre un profil par défaut configuré avec la configuration du fournisseur de jetons SSO. Le `sso_session` paramètre du profil fait référence à la `sso-session`section nommée. La `sso-session` section contient les paramètres permettant de lancer une session sur le portail AWS d'accès.

```
[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
```

Démarrer une session sur le portail AWS d'accès

Avant d'y accéder Services AWS, vous avez besoin d'une session de portail AWS d'accès active pour que le kit CDK puisse utiliser l'authentification IAM Identity Center pour résoudre les informations d'identification. En fonction de la durée de session que vous avez configurée, votre accès finira par

expirer et le kit CDK rencontrera une erreur d'authentification. Exécutez la commande suivante dans le AWS CLI pour vous connecter au portail AWS d'accès.

```
aws sso login
```

Si la configuration de votre fournisseur de jetons SSO utilise un profil nommé au lieu du profil par défaut, la commande est`aws sso login --profile NAME`. Spécifiez également ce profil lors de l'émission de cdk commandes à l'aide de l'`--profile`option ou de la variable d'`AWS_PROFILE`environnement.

Pour vérifier si vous avez déjà une session active, exécutez la AWS CLI commande suivante.

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

La réponse à cette commande doit indiquer le compte IAM Identity Center et l'ensemble d'autorisations configurés dans le fichier partagé `config`.

> **Note**
>
> Si vous disposez déjà d'une session active sur le portail AWS d'accès et que vous l'exécutez`aws sso login`, il ne vous sera pas demandé de fournir des informations d'identification.
> Le processus de connexion peut vous demander d'autoriser l' AWS CLI accès à vos données. Étant donné que le AWS CLI est construit au-dessus du SDK pour Python, les messages d'autorisation peuvent contenir des variantes du `botocore` nom.

Spécification de la région et d'autres configurations

Le kit d'outils CDK doit connaître la AWS région dans laquelle vous effectuez le déploiement et comment vous authentifier. AWS Cela est nécessaire pour les opérations de déploiement et pour récupérer les valeurs de contexte lors de la synthèse. Ensemble, votre compte et votre région constituent l'environnement.

La région peut être spécifiée à l'aide de variables d'environnement ou dans des fichiers de configuration. Il s'agit des mêmes variables et fichiers utilisés par d'autres AWS outils tels que les AWS SDK AWS CLI et les différents kits de développement logiciel. Le kit d'outils CDK recherche ces informations dans l'ordre suivant.

- La variable d'`AWS_DEFAULT_REGION`environnement.
- Un profil nommé défini dans le AWS `config` fichier standard et spécifié à l'aide de l'`--profile`option sur `cdk` les commandes.
- `[default]`Section du AWS `config` fichier standard.

Outre la spécification de AWS l'authentification et d'une région dans la `[default]` section, vous pouvez également ajouter une ou plusieurs `[profile` *`NAME`*`]` sections, où *`NAME`* est le nom du profil. Pour plus d'informations sur les profils nommés, consultez la section Fichiers de configuration et d'identification partagés dans le Guide de référence AWS des SDK et des outils.

Le AWS `config` fichier standard se trouve à l'emplacement `~/.aws/config` (macOS/Linux) ou `%USERPROFILE%\.aws\config` (Windows). Pour plus de détails et d'autres emplacements, voir Emplacement des fichiers de configuration et d'identification partagés dans le Guide de référence AWS des SDK et des outils

L'environnement que vous spécifiez dans votre AWS CDK application à l'aide de la `env` propriété de la pile est utilisé lors de la synthèse. Il est utilisé pour générer un AWS CloudFormation modèle spécifique à l'environnement et, lors du déploiement, il remplace le compte ou la région spécifié par l'une des méthodes précédentes. Pour plus d’informations, consultez the section called “Environnements”.

> **Note**
>
> Il AWS CDK utilise des informations d'identification provenant des mêmes fichiers source que les autres AWS outils et SDK, y compris le AWS Command Line Interface. Cependant, ils AWS CDK peuvent se comporter quelque peu différemment de ces outils. Il utilise le AWS SDK for JavaScript dessous du capot. Pour plus de détails sur la configuration des informations d'identification pour le AWS SDK for JavaScript, consultez la section Configuration des informations d'identification.

Vous pouvez éventuellement utiliser l'option `--role-arn` (ou`-r`) pour spécifier l'ARN d'un rôle IAM à utiliser pour le déploiement. Ce rôle doit être assumé par le AWS compte utilisé.

Spécification de la commande de l'application

De nombreuses fonctionnalités du kit d'outils CDK nécessitent la synthèse d'un ou de plusieurs AWS CloudFormation modèles, ce qui nécessite à son tour l'exécution de votre application. Il AWS

CDK prend en charge les programmes écrits dans une variété de langues. Par conséquent, il utilise une option de configuration pour spécifier la commande exacte nécessaire pour exécuter votre application. Cette option peut être spécifiée de deux manières.

Tout d'abord, et le plus souvent, il peut être spécifié à l'aide de la `app` clé contenue dans le fichier`cdk.json`. Il se trouve dans le répertoire principal de votre AWS CDK projet. Le kit d'outils CDK fournit une commande appropriée lors de la création d'un nouveau projet avec`cdk init`. Voici l'`cdk.json`extrait d'un nouveau TypeScript projet, par exemple.

```
{
  "app": "npx ts-node bin/hello-cdk.ts"
}
```

Le kit d'outils CDK recherche `cdk.json` dans le répertoire de travail actuel lorsque vous essayez d'exécuter votre application. Pour cette raison, vous pouvez garder un shell ouvert dans le répertoire principal de votre projet pour émettre des commandes du CDK Toolkit.

Le CDK Toolkit recherche également la clé de l'application `~/.cdk.json` (c'est-à-dire dans votre répertoire personnel) s'il ne la trouve pas dedans`./cdk.json`. L'ajout de la commande d'application ici peut être utile si vous travaillez habituellement avec du code CDK dans la même langue.

Si vous vous trouvez dans un autre répertoire, ou si vous souhaitez exécuter votre application à l'aide d'une commande autre que celle qui se `cdk.json` trouve dans, utilisez l'option `--app` (ou`-a`) pour la spécifier.

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

Lors du déploiement, vous pouvez également spécifier un répertoire contenant des assemblages cloud synthétisés`cdk.out`, tel que la valeur de--app. Les piles spécifiées sont déployées à partir de ce répertoire ; l'application n'est pas synthétisée.

Spécification des piles

De nombreuses commandes du CDK Toolkit (par exemple,`cdk deploy`) fonctionnent sur des piles définies dans votre application. Si votre application ne contient qu'une seule pile, le CDK Toolkit suppose que vous voulez dire cette pile si vous ne spécifiez pas de pile explicitement.

Dans le cas contraire, vous devez spécifier la ou les piles avec lesquelles vous souhaitez travailler. Vous pouvez le faire en spécifiant les piles souhaitées par ID individuellement sur la ligne de

commande. Rappelez-vous que l'ID est la valeur spécifiée par le deuxième argument lorsque vous instanciez la pile.

```
cdk synth PipelineStack LambdaStack
```

Vous pouvez également utiliser des caractères génériques pour spécifier des identifiants correspondant à un modèle.

- ?correspond à n'importe quel caractère
- *correspond à un nombre quelconque de caractères (*seul correspond à toutes les piles)
- **correspond à tous les éléments d'une hiérarchie

Vous pouvez également utiliser l'--alloption pour spécifier toutes les piles.

Si votre application utilise des CDK Pipelines, le CDK Toolkit considère vos piles et vos étapes comme une hiérarchie. De plus, l'--alloption et le * joker correspondent uniquement aux piles de haut niveau. Pour faire correspondre toutes les piles, utilisez**. Également utilisé ** pour indiquer toutes les piles d'une hiérarchie particulière.

Lorsque vous utilisez des caractères génériques, placez le motif entre guillemets ou évitez les caractères génériques avec. \ Si ce n'est pas le cas, votre shell peut essayer d'étendre le modèle aux noms des fichiers du répertoire en cours. Au mieux, cela ne donnera pas les résultats escomptés ; au pire, vous pourriez déployer des piles dont vous n'aviez pas l'intention. Cela n'est pas strictement nécessaire sous Windows car `cmd.exe` cela n'étend pas les caractères génériques, mais c'est néanmoins une bonne pratique.

```
cdk synth "*Stack"    # PipelineStack, LambdaStack, etc.
cdk synth 'Stack?'    # StackA, StackB, Stack1, etc.
cdk synth \*          # All stacks in the app, or all top-level stacks in a CDK
 Pipelines app
cdk synth '**'        # All stacks in a CDK Pipelines app
cdk synth 'PipelineStack/Prod/**'   # All stacks in Prod stage in a CDK Pipelines app
```

> **Note**
>
> L'ordre dans lequel vous spécifiez les piles n'est pas nécessairement l'ordre dans lequel elles seront traitées. La AWS CDK boîte à outils tient compte des dépendances entre les piles lorsqu'il s'agit de décider de l'ordre dans lequel les traiter. Par exemple, supposons qu'une

pile utilise une valeur produite par une autre (telle que l'ARN d'une ressource définie dans la deuxième pile). Dans ce cas, la deuxième pile est synthétisée avant la première en raison de cette dépendance. Vous pouvez ajouter des dépendances entre les piles manuellement à l'aide de la `addDependency()`méthode de la pile.

Booster votre environnement AWS

Le déploiement de piles avec le CDK nécessite le provisionnement de AWS CDK ressources dédiées spéciales. La `cdk bootstrap` commande crée les ressources nécessaires pour vous. Vous n'avez besoin de démarrer que si vous déployez une pile qui nécessite ces ressources dédiées. Consultez the section called “Action d'amorçage” pour plus de détails.

```
cdk bootstrap
```

Si elle est émise sans argument, comme indiqué ici, la `cdk bootstrap` commande synthétise l'application actuelle et démarre les environnements dans lesquels ses piles seront déployées. Si l'application contient des piles indépendantes de l'environnement, qui ne spécifient pas explicitement d'environnement, le compte et la région par défaut sont démarrés, ou l'environnement est spécifié à l'aide de. `--profile`

En dehors d'une application, vous devez spécifier explicitement l'environnement à démarrer. Vous pouvez également le faire pour démarrer un environnement qui n'est pas spécifié dans votre application ou votre AWS profil local. Les informations d'identification doivent être configurées (par exemple dans`~/.aws/credentials`) pour le compte et la région spécifiés. Vous pouvez spécifier un profil contenant les informations d'identification requises.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.
cdk bootstrap 1111111111/us-east-1
cdk bootstrap --profile test 1111111111/us-east-1
```

Important

Chaque environnement (combinaison compte/région) dans lequel vous déployez une telle pile doit être amorcé séparément.

Vous pouvez encourir des AWS frais pour ce que les ressources AWS CDK stockées dans les ressources amorcées sont susceptibles de vous être facturées. En outre, si vous l'utilisez`--bootstrap-customer-key`, une clé AWS KMS sera créée, ce qui entraîne également des frais par environnement.

> **Note**
>
> Les versions antérieures du modèle bootstrap créaient une clé KMS par défaut. Pour éviter les frais, redémarrez le fichier en utilisant. `--no-bootstrap-customer-key`

> **Note**
>
> Le CDK Toolkit v2 ne prend pas en charge le modèle bootstrap d'origine, surnommé le modèle hérité, utilisé par défaut avec CDK v1.

> **Important**
>
> Le modèle bootstrap moderne accorde efficacement les autorisations implicites `--cloudformation-execution-policies` à n'importe quel AWS compte de la `--trust` liste. Par défaut, cela étend les autorisations de lecture et d'écriture à n'importe quelle ressource du compte bootstrap. Assurez-vous de configurer la pile d'amorçage avec des politiques et des comptes fiables avec lesquels vous êtes à l'aise.

Création d'une nouvelle application

Pour créer une nouvelle application, créez un répertoire pour celle-ci, puis, dans le répertoire, publiez`cdk init`.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

Les langues prises en charge (*LANGUE*) sont les suivantes :

Code	Langue
`typescript`	TypeScript
`javascript`	JavaScript
`python`	Python
`java`	Java
`csharp`	C#

`TEMPLATE` est un modèle facultatif. Si le modèle souhaité est app, le modèle par défaut, vous pouvez l'omettre. Les modèles disponibles sont les suivants :

Modèle	Description
`app`(par défaut)	Crée une AWS CDK application vide.
`sample-app`	Crée une AWS CDK application avec une pile contenant une file d'attente Amazon SQS et une rubrique Amazon SNS.

Les modèles utilisent le nom du dossier du projet pour générer des noms pour les fichiers et les classes de votre nouvelle application.

Piles d'annonces

Pour voir la liste des identifiants des piles de votre AWS CDK application, entrez l'une des commandes équivalentes suivantes :

```
cdk list
cdk ls
```

Si votre application contient des piles CDK Pipelines, le CDK Toolkit affiche les noms des piles sous forme de chemins en fonction de leur emplacement dans la hiérarchie des pipelines. (Par exemple`PipelineStack`,`PipelineStack/Prod`, et`PipelineStack/Prod/MyService`.)

Si votre application contient de nombreuses piles, vous pouvez spécifier des ID de pile complets ou partiels pour les piles à répertorier. Pour plus d'informations, consultez the section called "Spécification des piles".

Ajoutez le `--long` drapeau pour obtenir plus d'informations sur les piles, notamment les noms des piles et leur environnement (AWS compte et région).

Synthétiser des piles

La `cdk synthesize` commande (presque toujours abrégée`synth`) synthétise une pile définie dans votre application dans un modèle. CloudFormation

```
cdk synth         # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"     # all stacks in app
```

> **Note**
>
> Le kit d'outils CDK exécute réellement votre application et synthétise de nouveaux modèles avant la plupart des opérations (par exemple lors du déploiement ou de la comparaison de piles). Ces modèles sont stockés par défaut dans le `cdk.out` répertoire. La `cdk synth` commande imprime simplement les modèles générés pour une ou plusieurs piles spécifiées.

Consultez toutes `cdk synth --help` les options disponibles. Quelques-unes des options les plus fréquemment utilisées sont abordées dans la section suivante.

Spécification des valeurs de contexte

Utilisez l'`-c`option `--context` ou pour transmettre les valeurs du contexte d'exécution à votre application 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
```

Lors du déploiement de plusieurs piles, les valeurs de contexte spécifiées sont normalement transmises à chacune d'entre elles. Si vous le souhaitez, vous pouvez spécifier des valeurs différentes pour chaque pile en préfixant le nom de la pile par la valeur de contexte.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

Spécification du format d'affichage

Par défaut, le modèle synthétisé est affiché au format YAML. Ajoutez le `--json` drapeau pour l'afficher au format JSON à la place.

```
cdk synth --json MyStack
```

Spécification du répertoire de sortie

Ajoutez l'option `--output` (`-o`) pour écrire les modèles synthétisés dans un répertoire autre que`cdk.out`.

```
cdk synth --output=~/templates
```

Déploiement de piles

La `cdk deploy` sous-commande déploie une ou plusieurs piles spécifiées sur votre compte. AWS

```
cdk deploy        # if app contains only one stack
cdk deploy MyStack
cdk deploy Stack1 Stack2
cdk deploy "*"    # all stacks in app
```

> **Note**
>
> Le kit d'outils CDK exécute votre application et synthétise de nouveaux AWS CloudFormation modèles avant de déployer quoi que ce soit. Par conséquent, la plupart des options de ligne de commande que vous pouvez utiliser `cdk synth` (par exemple,`--context`) peuvent également être utilisées avec`cdk deploy`.

Consultez toutes `cdk deploy --help` les options disponibles. Quelques-unes des options les plus utiles sont abordées dans la section suivante.

Ignorer la synthèse

La cdk deploy commande synthétise normalement les piles de votre application avant le déploiement afin de s'assurer que le déploiement reflète la dernière version de votre application. Si vous savez que vous n'avez pas modifié votre code depuis la dernière foiscdk synth, vous pouvez supprimer l'étape de synthèse redondante lors du déploiement. Pour cela, spécifiez le `cdk.out` répertoire de votre projet dans l'--appoption.

```
cdk deploy --app cdk.out StackOne StackTwo
```

Désactiver le rollback

AWS CloudFormation a la capacité d'annuler les modifications afin que les déploiements soient atomiques. Cela signifie qu'ils réussissent ou échouent dans leur ensemble. Le AWS CDK hérite de cette fonctionnalité car il synthétise et déploie AWS CloudFormation des modèles.

Le rollback garantit que vos ressources sont dans un état constant à tout moment, ce qui est vital pour les chaînes de production. Cependant, pendant que vous êtes encore en train de développer votre infrastructure, certaines défaillances sont inévitables, et l'annulation de déploiements échoués peut vous ralentir.

Pour cette raison, le kit d'outils CDK vous permet de désactiver le rollback en ajoutant des éléments `--no-rollback` à votre `cdk deploy` commande. Avec cet indicateur, les déploiements ayant échoué ne sont pas annulés. Au lieu de cela, les ressources déployées avant la ressource défaillante restent en place, et le déploiement suivant commence par la ressource défaillante. Vous passerez beaucoup moins de temps à attendre les déploiements et beaucoup plus de temps à développer votre infrastructure.

Échange à chaud

Utilisez le `--hotswap` drapeau avec `cdk deploy` pour essayer de mettre à jour vos AWS ressources directement au lieu de générer un ensemble de AWS CloudFormation modifications et de le déployer. Le déploiement revient au AWS CloudFormation déploiement si le hot swap n'est pas possible.

Actuellement, le hot swapping prend en charge les fonctions Lambda, les machines d'état Step Functions et les images de conteneurs Amazon ECS. Le `--hotswap` drapeau désactive également le rollback (c'est-à-dire implique`--no-rollback`).

> **Important**
>
> L'échange à chaud n'est pas recommandé pour les déploiements de production.

Mode montre

Le mode veille (cdk deploy --watchou cdk watch abrégé) du kit CDK Toolkit surveille en permanence les fichiers source et les actifs de votre application CDK pour détecter les modifications. Il effectue immédiatement un déploiement des piles spécifiées lorsqu'une modification est détectée.

Par défaut, ces déploiements utilisent l'`--hotswap`indicateur, qui accélère le déploiement des modifications apportées aux fonctions Lambda. Cela revient également au déploiement AWS CloudFormation si vous avez modifié la configuration de l'infrastructure. Pour avoir `cdk watch` toujours effectué des AWS CloudFormation déploiements complets, ajoutez l'`--no-hotswap`indicateur à`cdk watch`.

Toutes les modifications apportées alors qu'un déploiement `cdk watch` est déjà en cours sont combinées en un seul déploiement, qui commence dès que le déploiement en cours est terminé.

Le mode Watch utilise la `"watch"` clé du projet `cdk.json` pour déterminer les fichiers à surveiller. Par défaut, ces fichiers sont les fichiers et les actifs de votre application, mais cela peut être modifié en modifiant les `"exclude"` entrées `"include"` et de la `"watch"` clé. Le `cdk.json` fichier suivant montre un exemple de ces entrées.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

`cdk watch`exécute la `"build"` commande from `cdk.json` pour créer votre application avant la synthèse. Si votre déploiement nécessite des commandes pour créer ou empaqueter votre code Lambda (ou tout autre élément ne figurant pas dans votre application CDK), ajoutez-le ici.

Les caractères génériques de style Git, à la fois * et**, peuvent être utilisés dans les "watch" touches et. "build" Chaque chemin est interprété par rapport au répertoire parent decdk.json. La valeur par défaut include est**/*, c'est-à-dire tous les fichiers et répertoires du répertoire racine du projet. excludeest facultatif.

> **Important**
>
> Le mode Watch n'est pas recommandé pour les déploiements de production.

Spécification AWS CloudFormation des paramètres

Le AWS CDK kit d'outils permet de spécifier AWS CloudFormation des paramètres lors du déploiement. Vous pouvez les fournir sur la ligne de commande après le --parameters drapeau.

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

Pour définir plusieurs paramètres, utilisez plusieurs --parameters indicateurs.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
 downloadBucketName=DownBucket
```

Si vous déployez plusieurs piles, vous pouvez spécifier une valeur différente pour chaque paramètre pour chaque pile. Pour ce faire, préfixez le nom du paramètre avec le nom de la pile et deux points. Sinon, la même valeur est transmise à toutes les piles.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --
parameters YourStack:uploadBucketName=UpBucket
```

Par défaut, le AWS CDK conserve les valeurs des paramètres des déploiements précédents et les utilise dans les déploiements ultérieurs si elles ne sont pas spécifiées explicitement. Utilisez l'--no-previous-parametersindicateur pour exiger que tous les paramètres soient spécifiés.

Spécification du fichier de sorties

Si votre pile déclare AWS CloudFormation des sorties, celles-ci sont normalement affichées à l'écran à la fin du déploiement. Pour les écrire dans un fichier au format JSON, utilisez le --outputs-file drapeau.

```
cdk deploy --outputs-file outputs.json MyStack
```

Changements liés à la sécurité

Pour vous protéger contre les modifications involontaires qui affectent votre posture de sécurité, le AWS CDK kit d'outils vous invite à approuver les modifications liées à la sécurité avant de les déployer. Vous pouvez spécifier le niveau de modification qui doit être approuvé :

```
cdk deploy --require-approval LEVEL
```

Le *NIVEAU* peut être l'un des suivants :

Durée	Signification
`never`	L'approbation n'est jamais requise
`any-change`	Nécessite une approbation pour tout IAM ou modification security-group-related
`broadening` (par défaut)	Nécessite une approbation lorsque des instructions IAM ou des règles de circulation sont ajoutées ; les suppressions ne nécessitent pas d'approbation

Le paramètre peut également être configuré dans le `cdk.json` fichier.

```
{
  "app": "...",
  "requireApproval": "never"
}
```

Comparaison des piles

La `cdk diff` commande compare la version actuelle d'une pile (et ses dépendances) définie dans votre application avec les versions déjà déployées ou avec un AWS CloudFormation modèle enregistré, et affiche une liste des modifications.

```
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
```

Pour comparer les stacks de votre application avec le déploiement existant :

```
cdk diff MyStack
```

Pour comparer les stacks de votre application avec un CloudFormation modèle enregistré :

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

Importation de ressources existantes dans une pile

Vous pouvez utiliser la `cdk import` commande pour placer les ressources sous la gestion CloudFormation d'une AWS CDK pile particulière. Cela est utile si vous migrez vers des piles AWS CDK, si vous déplacez des ressources entre des piles ou si vous modifiez leur identifiant logique. `cdk import` Utilise des importations de CloudFormation ressources. Consultez la liste des ressources pouvant être importées ici.

Pour importer une ressource existante dans une AWS CDK pile, procédez comme suit :

- Assurez-vous que la ressource n'est actuellement gérée par aucune autre CloudFormation pile. Si tel est le cas, définissez d'abord la politique de suppression sur la pile `RemovalPolicy.RETAIN` dans laquelle se trouve actuellement la ressource et effectuez un déploiement. Supprimez ensuite la ressource de la pile et effectuez un autre déploiement. Ce processus permet de s'assurer que la ressource n'est plus gérée CloudFormation mais ne la supprime pas.
- Exécutez un `cdk diff` pour vous assurer qu'aucune modification n'est en attente dans la AWS CDK pile dans laquelle vous souhaitez importer des ressources. Les seules modifications autorisées lors d'une opération « d'importation » sont l'ajout de nouvelles ressources que vous souhaitez importer.
- Ajoutez des structures pour les ressources que vous souhaitez importer dans votre pile. Par exemple, si vous souhaitez importer un compartiment Amazon S3, ajoutez quelque chose comme`new s3.Bucket(this, 'ImportedS3Bucket', {});`. N'apportez aucune modification à aucune autre ressource.

  Vous devez également vous assurer de modéliser exactement l'état actuel de la ressource dans la définition. Pour l'exemple du bucket, veillez à inclure AWS KMS les clés, les politiques de cycle de vie et tout autre élément pertinent concernant le bucket. Dans le cas contraire, les opérations de mise à jour suivantes risquent de ne pas donner les résultats escomptés.

  Vous pouvez choisir d'inclure ou non le nom du compartiment physique. Nous recommandons généralement de ne pas inclure de noms de AWS CDK ressources dans vos définitions de ressources afin de faciliter le déploiement de vos ressources à plusieurs reprises.
- Exécutez `cdk import` *`STACKNAME`*.
- Si les noms des ressources ne figurent pas dans votre modèle, la CLI vous invite à transmettre les noms réels des ressources que vous importez. Ensuite, l'importation commence.
- Lorsque `cdk import` les rapports indiquent un succès, la ressource est désormais gérée par AWS CDK et CloudFormation. Toute modification ultérieure que vous apporterez aux propriétés des ressources de votre AWS CDK application, la configuration de construction sera appliquée lors du prochain déploiement.
- Pour vérifier que la définition de la ressource dans votre AWS CDK application correspond à l'état actuel de la ressource, vous pouvez démarrer une opération de détection de CloudFormation dérive.

Cette fonctionnalité ne prend actuellement pas en charge l'importation de ressources dans des piles imbriquées.

Configuration (`cdk.json`)

Les valeurs par défaut de nombreux indicateurs de ligne de commande du CDK Toolkit peuvent être stockées dans `cdk.json` le fichier d'un projet ou dans le `.cdk.json` fichier de votre répertoire utilisateur. Vous trouverez ci-dessous une référence alphabétique aux paramètres de configuration pris en charge.

Clé	Remarques	Option du kit d'outils CDK
`app`	Commande qui exécute l'application CDK.	--app
`assetMetadata`	Si`false`, CDK n'ajoute pas de métadonnées aux ressources qui utilisent des actifs.	--no-asset-metadata
`bootstrapKmsKeyId`	Remplace l'ID de la AWS KMS clé utilisée pour chiffrer le compartiment de déploiement Amazon S3.	--bootstrap-kms-key-id
`build`	Commande qui compile ou crée l'application CDK avant la synthèse. Non autorisé à entrer`~/.cdk.json` .	--build
`browser`	Commande permettant de lancer un navigateur Web pour la `cdk docs` sous-comm ande.	--browser
`context`	veuillez consulter the section called “Contexte”. Les valeurs de contexte d'un fichier de configuration ne seront pas effacées parcdk context --	--context

Clé	Remarques	Option du kit d'outils CDK
	clear. (Le kit d'outils CDK place les valeurs de contexte mises en `cdk.context.json` cache.)	
`debug`	Si `true`, le CDK Toolkit émet des informations plus détaillées utiles pour le débogage.	--debug
`language`	Langage à utiliser pour initialiser les nouveaux projets.	--language
`lookups`	Dans `false` le cas contraire, aucune recherche de contexte n'est autorisée. La synthèse échouera si des recherches de contexte doivent être effectuées.	--no-lookups
`notices`	Si `false`, supprime l'affichage de messages concernant les failles de sécurité, les régressions et les versions non prises en charge.	--no-notices
`output`	Nom du répertoire dans lequel l'assemblage cloud synthétisé sera émis (par défaut `"cdk.out"`).	--output
`outputsFile`	Le fichier dans lequel les AWS CloudFormation sorties des piles déployées seront écrites (au format JSON).	--outputs-file

Clé	Remarques	Option du kit d'outils CDK
`pathMetadata`	Si`false`, les métadonnées du chemin du CDK ne sont pas ajoutées aux modèles synthétisés.	--no-path-metadata
`plugin`	Tableau JSON spécifiant les noms de packages ou les chemins locaux des packages qui étendent le CDK	--plugin
`profile`	Nom du AWS profil par défaut utilisé pour spécifier la région et les informations d'identif ication du compte.	--profile
`progress`	S'il est défini sur`"events"`, le kit d'outils CDK affiche tous les AWS CloudForm ation événements pendant le déploiement, plutôt qu'une barre de progression.	--progress
`requireApproval`	Niveau d'approbation par défaut pour les modifications de sécurité. Consultez the section called “Changements liés à la sécurité”.	--require-approval
`rollback`	Si `false` les déploiements ayant échoué ne sont pas annulés.	--no-rollback

Clé	Remarques	Option du kit d'outils CDK
`staging`	Si `false` les actifs ne sont pas copiés dans le répertoire de sortie (à utiliser pour le débogage local des fichiers source avec AWS SAM).	--no-staging
`tags`	Objet JSON contenant des balises (paires clé-valeur) pour la pile.	--tags
`toolkitBucketName`	Le nom du compartiment Amazon S3 utilisé pour déployer des actifs tels que les fonctions Lambda et les images de conteneur (voir. the section called “Booster votre environnement AWS”	--toolkit-bucket-name
`toolkitStackName`	Le nom de la pile bootstrap (voirthe section called “Booster votre environnement AWS”.	--toolkit-stack-name
`versionReporting`	Si`false`, désactive le rapport de version.	--no-version-reporting
`watch`	Objet JSON contenant `"include"` des `"exclude"` clés indiquant quels fichiers doivent (ou ne doivent pas) déclencher une reconstruction du projet en cas de modification. veuillez consulter the section called “Mode montre”.	--watch

cdk migrateréférence de commande

Référence pour la AWS Cloud Development Kit (AWS CDK) commande Command Line Interface (CLI)`cdk migrate`. Pour plus d'informations sur l'utilisation`cdk migrate`, consultezMigrez les ressources et les AWS CloudFormation modèles existants vers le AWS CDK.

La `cdk migrate` commande migre les AWS ressources déployées, les AWS CloudFormation piles et les AWS CloudFormation modèles locaux vers. AWS CDK

Rubriques

Utilisation

```
$ cdk migrate <options>
```

Options

Options requises

`--stack-name` *STRING*

Nom de la AWS CloudFormation pile qui sera créée dans l'application CDK après la migration.

Obligatoire : oui

Options conditionnelles

`--from-path` *PATH*

Le chemin d'accès au AWS CloudFormation modèle à migrer. Fournissez cette option pour spécifier un modèle local.

Obligatoire : selon les conditions. Obligatoire en cas de migration depuis un AWS CloudFormation modèle local.

`--from-scan` *STRING*

Lorsque vous migrez des ressources déployées depuis un AWS environnement, utilisez cette option pour spécifier si un nouveau scan doit être lancé ou s'il AWS CDK CLI faut utiliser le dernier scan réussi.

Obligatoire : selon les conditions. Nécessaire lors de la migration à partir de AWS ressources déployées.

Valeurs acceptées :`most-recent`, `new`

`--from-stack`

Fournissez cette option pour migrer depuis une AWS CloudFormation pile déployée. `--stack-name`À utiliser pour spécifier le nom de la AWS CloudFormation pile déployée.

Obligatoire : selon les conditions. Obligatoire en cas de migration depuis une AWS CloudFormation pile déployée.

Options facultatives

`--account` *STRING*

Le compte à partir duquel récupérer le modèle de AWS CloudFormation pile.

Obligatoire : non

Par défaut : AWS CDK CLI obtient les informations du compte à partir de sources par défaut.

`--compress`

Fournissez cette option pour compresser le projet CDK généré dans un ZIP fichier.

Obligatoire : non

`--filter` *ARRAY*

À utiliser lors de la migration des ressources déployées depuis un AWS compte et Région AWS. Cette option spécifie un filtre pour déterminer les ressources déployées à migrer.

Cette option accepte un tableau de paires clé-valeur, où la clé représente le type de filtre et la valeur représente la valeur à filtrer.

Les clés suivantes sont acceptées :

- `resource-identifier`— Identifiant de la ressource. La valeur peut être l'identifiant logique ou physique de la ressource. Par exemple, `resource-identifier="ClusterName"`.
- `resource-type-prefix`— Le préfixe du type de AWS CloudFormation ressource. Par exemple, spécifiez `resource-type-prefix="AWS::DynamoDB::"` de filtrer toutes les ressources Amazon DynamoDB.
- `tag-key`— La clé d'une balise de ressource. Par exemple, `tag-key="myTagKey"`.
- `tag-value`— La valeur d'une balise de ressource. Par exemple, `tag-value="myTagValue"`.

Fournissez plusieurs paires clé-valeur pour la logique `AND` conditionnelle. L'exemple suivant filtre toute ressource DynamoDB étiquetée `myTagKey` avec comme clé de balise :. `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

Fournissez l'`--filter`option plusieurs fois dans une seule commande pour la logique `OR` conditionnelle. L'exemple suivant filtre toute ressource qui est une ressource DynamoDB ou qui est étiquetée comme clé `myTagKey` de balise :. `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

Obligatoire : non

`--language` *`STRING`*

Langage de programmation à utiliser pour le projet CDK créé lors de la migration.

Obligatoire : non

Valeurs acceptées :`typescript`,`python`,`java`,`csharp`,`go`.

Par défaut : `typescript`

`--output-path` *`PATH`*

Le chemin de sortie du projet CDK migré.

Obligatoire : non

Par défaut : Par défaut, votre répertoire de travail actuel AWS CDK CLI sera utilisé.

`--region` *`STRING`*

Le Région AWS pour récupérer le modèle de AWS CloudFormation pile.

Obligatoire : non

Par défaut : AWS CDK CLI obtient des Région AWS informations à partir de sources par défaut.

AWS Boîte à outils pour Visual Studio Code

Le AWS Toolkit for Visual Studio Code est un plugin open source pour Visual Studio Code qui facilite la création, le débogage et le AWS déploiement d'applications. La boîte à outils fournit une expérience intégrée pour le développement AWS CDK d'applications. Il inclut la fonctionnalité AWS CDK Explorer pour répertorier vos AWS CDK projets et parcourir les différents composants de l'application CDK. Installez le AWS kit d'outils et apprenez-en plus sur l'utilisation de l' AWS CDK explorateur.

AWS SAM intégration

Le AWS CDK et le AWS Serverless Application Model (AWS SAM) peuvent fonctionner ensemble pour vous permettre de créer et de tester localement des applications sans serveur définies dans le CDK. Pour des informations complètes, consultez AWS Cloud Development Kit (AWS CDK)le guide du AWS SAM développeur. Pour installer la CLI SAM, reportez-vous à la section Installation de la AWS SAM CLI.

Constructions de test

Avec le AWS CDK, votre infrastructure peut être aussi testable que n'importe quel autre code que vous écrivez. L'approche standard pour tester AWS CDK les applications utilise le module AWS CDK d'assertions et des frameworks de test populaires tels que Jest pour JavaScript et/ou TypeScript Pytest pour Python.

Il existe deux catégories de tests que vous pouvez écrire pour les AWS CDK applications.

- Des assertions détaillées testent des aspects spécifiques du AWS CloudFormation modèle généré, tels que « cette ressource possède cette propriété avec cette valeur ». Ces tests permettent de détecter des régressions. Ils sont également utiles lorsque vous développez de nouvelles fonctionnalités à l'aide du développement piloté par les tests. (Vous pouvez d'abord écrire un test, puis le réussir en écrivant une implémentation correcte.) Les assertions précises sont les tests les plus fréquemment utilisés.
- Les tests instantanés testent le AWS CloudFormation modèle synthétisé par rapport à un modèle de référence précédemment stocké. Les tests instantanés vous permettent de refactoriser librement, car vous pouvez être sûr que le code refactorisé fonctionne exactement de la même manière que le code original. Si les modifications étaient intentionnelles, vous pouvez accepter une nouvelle référence pour les futurs tests. Cependant, les mises à niveau du CDK peuvent également entraîner la modification des modèles synthétisés. Vous ne pouvez donc pas vous fier uniquement aux instantanés pour vous assurer que votre implémentation est correcte.

> **Note**
>
> Les versions complètes TypeScript des applications Python et Java utilisées comme exemples dans cette rubrique sont disponibles sur GitHub.

Premiers pas

Pour illustrer comment écrire ces tests, nous allons créer une pile contenant une machine à AWS Step Functions états et une AWS Lambda fonction. La fonction Lambda est abonnée à une rubrique Amazon SNS et transmet simplement le message à la machine d'état.

Créez d'abord un projet d'application CDK vide à l'aide du kit d'outils CDK et en installant les bibliothèques dont nous aurons besoin. Les constructions que nous utiliserons se trouvent toutes

dans le package CDK principal, qui est une dépendance par défaut dans les projets créés avec le CDK Toolkit. Cependant, vous devez installer votre framework de test.

TypeScript

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

Créez un répertoire pour vos tests.

```
mkdir test
```

Modifiez les projets `package.json` pour indiquer à NPM comment exécuter Jest et pour indiquer à Jest quels types de fichiers collecter. Les modifications nécessaires sont les suivantes.

- Ajouter une nouvelle `test` clé à la `scripts` section
- Ajoutez Jest et ses types à la section `devDependencies`
- Ajouter une nouvelle clé `jest` de niveau supérieur avec une déclaration `moduleFileExtensions`

Ces modifications sont présentées dans le schéma suivant. Placez le nouveau texte à l'endroit indiqué dans`package.json`. Les espaces réservés «... » indiquent les parties existantes du fichier qui ne doivent pas être modifiées.

```
{
  ...
  "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
```

Créez un répertoire pour vos tests.

```
mkdir test
```

Modifiez les projets `package.json` pour indiquer à NPM comment exécuter Jest et pour indiquer à Jest quels types de fichiers collecter. Les modifications nécessaires sont les suivantes.

- Ajouter une nouvelle `test` clé à la `scripts` section
- Ajoutez Jest à la section `devDependencies`
- Ajouter une nouvelle clé `jest` de niveau supérieur avec une déclaration `moduleFileExtensions`

Ces modifications sont présentées dans le schéma suivant. Placez le nouveau texte à l'endroit indiqué dans`package.json`. Les espaces réservés «... » indiquent les parties existantes du fichier qui ne doivent pas être modifiées.

```
{
  ...
  "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
```

Ouvrez le projet dans l'IDE Java de votre choix. (Dans Eclipse, utilisez Fichier > Importer > Projets Maven existants.)

C#

```
mkdir state-machine && cd-state-machine
cdk init --language=csharp
```

Ouvrez `src\StateMachine.sln` dans Visual Studio.

Cliquez avec le bouton droit sur la solution dans l'Explorateur de solutions et choisissez Ajouter > Nouveau projet. Recherchez MSTest C# et ajoutez un projet de test MSTest pour C#. (Le nom par défaut `TestProject1` est correct.)

Cliquez avec le bouton droit de la souris `TestProject1` et choisissez Ajouter > Référence du `StateMachine` projet, puis ajoutez le projet comme référence.

La pile d'exemples

Voici la pile qui sera testée dans cette rubrique. Comme nous l'avons décrit précédemment, il contient une fonction Lambda et une machine d'état Step Functions, et accepte une ou plusieurs rubriques Amazon SNS. La fonction Lambda est abonnée aux rubriques Amazon SNS et les transmet à la machine à états.

Vous n'avez rien à faire de spécial pour rendre l'application testable. En fait, cette pile de CDK n'est pas différente des autres exemples de piles présentés dans ce guide.

TypeScript

Entrez le code suivant dans `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

Entrez le code suivant dans `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

Entrez le code suivant dans `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);
            }

        }
    }
}
```

Nous allons modifier le point d'entrée principal de l'application afin de ne pas réellement instancier notre stack. Nous ne voulons pas le déployer accidentellement. Nos tests créeront une application et une instance de la pile à des fins de test. Cette tactique est utile lorsqu'elle est associée au développement piloté par les tests : assurez-vous que la pile passe tous les tests avant d'activer le déploiement.

TypeScript

Dans `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

Dans `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

Dans `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 fonction Lambda

Notre exemple de pile inclut une fonction Lambda qui démarre notre machine à états. Nous devons fournir le code source de cette fonction afin que le CDK puisse la regrouper et la déployer dans le cadre de la création de la ressource de fonction Lambda.

- Créez le dossier `start-state-machine` dans le répertoire principal de l'application.
- Dans ce dossier, créez au moins un fichier. Par exemple, vous pouvez enregistrer le code suivant dans`start-state-machines/index.js`.

```
exports.handler = async function (event, context) {
 return 'hello world';
};
```

Cependant, n'importe quel fichier fonctionnera, car nous ne déploierons pas réellement la pile.

Exécution de tests

À titre de référence, voici les commandes que vous utilisez pour exécuter des tests dans votre AWS CDK application. Il s'agit des mêmes commandes que vous utiliseriez pour exécuter les tests dans n'importe quel projet utilisant le même framework de test. Pour les langages qui nécessitent une étape de compilation, incluez-la pour vous assurer que vos tests ont été compilés.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

Créez votre solution (F6) pour découvrir les tests, puis exécutez-les (Test > Exécuter tous les tests). Pour choisir les tests à exécuter, ouvrez l'Explorateur de tests (Test > Explorateur de tests).

Ou:

```
dotnet test src
```

Assertions fines

La première étape pour tester une pile avec des assertions détaillées consiste à synthétiser la pile, car nous écrivons des assertions par rapport au modèle généré. AWS CloudFormation

Nous `StateMachineStackStack` exigeons que nous lui transmettions le sujet Amazon SNS à transmettre à la machine d'état. Dans notre test, nous allons donc créer une pile séparée pour contenir le sujet.

Normalement, lorsque vous écrivez une application CDK, vous pouvez sous-classer `Stack` et instancier la rubrique Amazon SNS dans le constructeur de la pile. Dans notre test, nous instancions `Stack` directement, puis nous transmettons cette pile comme scope, en `Topic` l'attachant à la pile. C'est équivalent sur le plan fonctionnel et moins verbeux. Cela permet également de rendre les piles utilisées uniquement dans les tests « différentes » des piles que vous avez l'intention de déployer.

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
        }
```

```
    }
}
```

Nous pouvons maintenant affirmer que la fonction Lambda et l'abonnement Amazon SNS ont été créés.

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);
```

Notre test de fonction Lambda affirme que deux propriétés particulières de la ressource fonctionnelle ont des valeurs spécifiques. Par défaut, la `hasResourceProperties` méthode effectue une correspondance partielle sur les propriétés de la ressource telles qu'elles sont indiquées dans le CloudFormation modèle synthétisé. Ce test nécessite que les propriétés fournies existent et aient les valeurs spécifiées, mais la ressource peut également avoir d'autres propriétés, qui ne sont pas testées.

Notre assertion Amazon SNS affirme que le modèle synthétisé contient un abonnement, mais rien sur l'abonnement lui-même. Nous avons inclus cette assertion principalement pour illustrer comment affirmer le nombre de ressources. La `Template` classe propose des méthodes plus spécifiques pour écrire des assertions par rapport aux `Mapping` sections `ResourcesOutputs`, et du CloudFormation modèle.

Allumeurs

Le comportement de correspondance partielle par défaut de `hasResourceProperties` peut être modifié à l'aide des matchers de la `Match`classe.

Les matchers vont de indulgent (`Match.anyValue`) à strict (`Match.objectEquals`). Ils peuvent être imbriqués pour appliquer différentes méthodes de correspondance aux différentes parties des propriétés des ressources. En utilisant `Match.objectEquals` et `Match.anyValue` ensemble, par exemple, nous pouvons tester le rôle IAM de la machine à états de manière plus complète, sans exiger de valeurs spécifiques pour les propriétés susceptibles de changer.

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" }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }));
```

De nombreuses CloudFormation ressources incluent des objets JSON sérialisés représentés sous forme de chaînes. Le `Match.serializedJson()` matcher peut être utilisé pour faire correspondre les propriétés de ce JSON.

Par exemple, les machines à états Step Functions sont définies à l'aide d'une chaîne dans le langage Amazon States Language basé sur JSON. Nous allons nous `Match.serializedJson()` assurer que notre état initial est la seule étape. Encore une fois, nous utiliserons des matchers imbriqués pour appliquer différents types de correspondance aux différentes parties de l'objet.

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

Capture

Il est souvent utile de tester les propriétés pour s'assurer qu'elles suivent des formats spécifiques ou qu'elles ont la même valeur qu'une autre propriété, sans avoir besoin de connaître leurs valeurs exactes à l'avance. Le `assertions` module fournit cette fonctionnalité dans sa `Capture`classe.

En spécifiant une `Capture` instance à la place d'une valeur in`hasResourceProperties`, cette valeur est conservée dans l'`Capture`objet. La valeur capturée réelle peut être récupérée à l'aide `as` des méthodes de l'objet `asNumber()``asString()`, notamment`asObject`, et soumise à un test. À utiliser `Capture` avec un comparateur pour spécifier l'emplacement exact de la valeur à capturer dans les propriétés de la ressource, y compris les propriétés JSON sérialisées.

L'exemple suivant teste pour s'assurer que l'état de départ de notre machine à états porte un nom commençant par`Start`. Il vérifie également que cet état est présent dans la liste des états de la machine.

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()));
```

Tests instantanés

Lors des tests instantanés, vous comparez l'intégralité du CloudFormation modèle synthétisé à un modèle de référence précédemment stocké (souvent appelé « modèle principal »). Contrairement aux assertions détaillées, les tests instantanés ne sont pas utiles pour détecter les régressions. Cela est dû au fait que les tests instantanés s'appliquent à l'ensemble du modèle et que d'autres éléments que les modifications de code peuvent entraîner de légères (ou not-so-small) différences dans les résultats de synthèse. Ces modifications n'affecteront peut-être même pas votre déploiement, mais elles entraîneront tout de même l'échec d'un test de capture instantanée.

Par exemple, vous pouvez mettre à jour une structure de CDK pour intégrer une nouvelle bonne pratique, ce qui peut entraîner des modifications des ressources synthétisées ou de la façon dont elles sont organisées. Vous pouvez également mettre à jour le kit d'outils CDK vers une version qui indique des métadonnées supplémentaires. Les modifications apportées aux valeurs de contexte peuvent également affecter le modèle synthétisé.

Les tests instantanés peuvent toutefois être d'une grande aide pour le refactoring, à condition de maintenir constants tous les autres facteurs susceptibles d'affecter le modèle synthétisé. Vous saurez immédiatement si une modification que vous avez apportée a involontairement modifié le modèle. Si le changement est intentionnel, il suffit d'accepter le nouveau modèle comme base de référence.

Par exemple, si nous avons cette `DeadLetterQueue` construction :

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()
            });
        }
    }
}
```

Nous pouvons le tester comme ceci :

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());
        }
    }
}
```

Conseils pour les tests

N'oubliez pas que vos tests dureront aussi longtemps que le code qu'ils testent, et qu'ils seront lus et modifiés aussi souvent. Par conséquent, il vaut la peine de prendre un moment pour réfléchir à la meilleure façon de les écrire.

Ne copiez pas et ne collez pas de lignes de configuration ou d'assertions courantes. Refactorisez plutôt cette logique en accessoires ou en fonctions auxiliaires. Utilisez de bons noms qui reflètent ce que chaque test teste réellement.

N'essayez pas d'en faire trop en un seul test. De préférence, un test ne doit tester qu'un seul comportement. Si vous rompez accidentellement ce comportement, un seul test doit échouer, et le nom du test doit vous indiquer ce qui a échoué. Cependant, il s'agit plutôt d'un idéal à atteindre ; il arrive que vous écriviez inévitablement (ou par inadvertance) des tests qui testent plus d'un comportement. Pour les raisons que nous avons déjà décrites, les tests instantanés sont particulièrement sujets à ce problème. Utilisez-les donc avec parcimonie.

Sécurité pour AWS Cloud Development Kit (AWS CDK)

Chez Amazon Web Services (AWS), la sécurité dans le cloud est la priorité principale. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité. La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cela comme la sécurité du cloud et la sécurité dans le cloud.

Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute tous les services proposés dans le AWS cloud et de vous fournir des services que vous pouvez utiliser en toute sécurité. Notre responsabilité en matière de sécurité est notre priorité absolue AWS, et l'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de AWS conformité.

Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez et par d'autres facteurs, notamment la sensibilité de vos données, les exigences de votre organisation et les lois et réglementations applicables.

Il AWS CDK suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Rubriques

Gestion des identités et des accès pour AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser AWS les ressources. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez. AWS

Utilisateur du service : si vous avez l' Services AWS habitude de faire votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de nouvelles AWS fonctionnalités pour effectuer votre travail, vous aurez peut-être besoin d'autorisations supplémentaires. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur.

Administrateur du service — Si vous êtes responsable des AWS ressources de votre entreprise, vous avez probablement un accès complet aux AWS ressources. C'est à vous de déterminer à quelles ressources Services AWS les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM.

Administrateur IAM – Si vous êtes un administrateur IAM, vous souhaiterez peut-être en savoir plus sur la façon d'écrire des politiques pour gérer l'accès à Services AWS.

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié (connecté à AWS) en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez la section Comment vous connecter à votre compte Compte AWS dans le guide de Connexion à AWS l'utilisateur.

Pour y accéder AWS par programmation, AWS fournit des kits de AWS CDK développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer vos demandes vous-même, veuillez consulter la rubrique Processus de signature Signature Version 4 dans la Références générales AWS.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour en savoir plus, veuillez consulter Multi-factor authentication (Authentification multifactorielle) dans le Guide de l'utilisateur AWS IAM Identity Center et Utilisation de l'authentification multifactorielle (MFA) dans l'interface AWS dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur root, consultez Tâches nécessitant des informations d'identification d'utilisateur root dans le Guide de l'utilisateur IAM.

Identité fédérée

La meilleure pratique consiste à obliger les utilisateurs humains, y compris ceux qui ont besoin d'un accès administrateur, à utiliser la fédération avec un fournisseur d'identité pour accéder à l'aide Services AWS d'informations d'identification temporaires.

Une identité fédérée est un utilisateur de l'annuaire des utilisateurs de votre entreprise, d'un fournisseur d'identité Web AWS Directory Service, du répertoire Identity Center ou de tout utilisateur qui y accède à l'aide des informations d'identification fournies Services AWS par le biais d'une source d'identité. Lorsque des identités fédérées y accèdent Comptes AWS, elles assument des rôles, qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Vous pouvez créer des utilisateurs et des groupes dans IAM Identity Center, ou vous pouvez vous connecter et synchroniser avec un ensemble d'utilisateurs et de groupes dans votre propre source d'identité afin de les utiliser dans toutes vos applications Comptes AWS et applications. Pour obtenir des informations sur IAM Identity Center, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou une seule application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme tels que les clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé IAMAdmins et accorder à ce groupe les autorisations d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, consultez Quand créer un utilisateur IAM (au lieu d'un rôle) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité au sein de vous Compte AWS dotée d'autorisations spécifiques. S'il est comparable à un utilisateur IAM, il n'est toutefois pas associé à une personne déterminée. Vous pouvez assumer temporairement un rôle IAM dans le en AWS Management Console changeant de rôle. Vous pouvez assumer un rôle en appelant une opération d' AWS API AWS CLI ou en utilisant une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Utilisation de rôles IAM dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré – Pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, veuillez consulter la rubrique Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte – Vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez Différence entre les rôles IAM et les politiques basées sur les ressources dans le Guide de l'utilisateur IAM.
- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, une fonction de service ou un rôle lié au service.
  - Fonction du service – Il s'agit d'un rôle IAM attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction du service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.
  - Rôle lié à un service — Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés au service apparaissent dans votre Compte AWS fichier et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.
- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une instance EC2

et qui envoient des demandes d'API. AWS CLI AWS Cette solution est préférable au stockage des clés d'accès au sein de l'instance EC2. Pour attribuer un AWS rôle à une instance EC2 et le mettre à la disposition de toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2 dans le Guide de l'utilisateur IAM.

Pour savoir dans quel cas utiliser des rôles ou des utilisateurs IAM, consultez Quand créer un rôle IAM (au lieu d'un utilisateur) dans le Guide de l'utilisateur IAM.

Validation de conformité pour AWS Cloud Development Kit (AWS CDK)

Il AWS CDK suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

La sécurité et la conformité des AWS services sont évaluées par des auditeurs tiers dans le cadre de multiples programmes de AWS conformité. Il s'agit notamment du SOC, du PCI, du FedRAMP, de l'HIPAA et d'autres. AWS fournit une liste fréquemment mise à jour des AWS services relevant de programmes de conformité spécifiques sur la page AWS Services in Scope by Compliance Program.

Les rapports d'audit tiers peuvent être téléchargés à l'aide d' AWS Artifact. Pour plus d'informations, consultez la section Téléchargement de rapports dans AWS Artifact.

Pour plus d'informations sur les programmes de AWS conformité, consultez AWS la section Programmes de conformité.

Votre responsabilité en matière de conformité lorsque vous utilisez le AWS CDK pour accéder à un AWS service est déterminée par la sensibilité de vos données, les objectifs de conformité de votre organisation et les lois et réglementations applicables. Si votre utilisation d'un AWS service est soumise au respect de normes telles que HIPAA, PCI ou FedRAMP, fournit des ressources pour vous aider à : AWS

- Guides de démarrage rapide sur la sécurité et la conformité : guides de déploiement qui abordent les considérations architecturales et indiquent les étapes à suivre pour déployer des environnements de référence axés sur la sécurité et sur la conformité sur. AWS
- AWS Ressources relatives à la conformité : collection de classeurs et de guides susceptibles de s'appliquer à votre secteur d'activité et à votre région.
- AWS Config – Service qui permet d'évaluer comment les configurations de vos ressources se conforment aux pratiques internes, aux normes et aux directives industrielles.
- AWS Security Hub— Une vue complète de votre état de sécurité interne AWS qui vous permet de vérifier votre conformité aux normes et aux meilleures pratiques du secteur de la sécurité.

Résilience pour AWS Cloud Development Kit (AWS CDK)

L'infrastructure mondiale d'Amazon Web Services (AWS) est construite autour des AWS régions et des zones de disponibilité.

AWS Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant.

Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS mondiale.

Il AWS CDK suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Sécurité de l'infrastructure pour AWS Cloud Development Kit (AWS CDK)

Il AWS CDK suit le modèle de responsabilité partagée par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des

AWS services, consultez la AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité.

Résolution des AWS CDK problèmes courants

Cette rubrique explique comment résoudre les problèmes suivants liés au AWS CDK.

Après avoir mis à jour le AWS CDK, le AWS CDK Toolkit (CLI) signale une incompatibilité avec la bibliothèque AWS Construct

La version du AWS CDK Toolkit (qui fournit la `cdk` commande) doit être au moins égale à la version du module principal de AWS Construct Library,`aws-cdk-lib`. Le kit d'outils est destiné à être rétrocompatible. La dernière version 2.x de la boîte à outils peut être utilisée avec n'importe quelle version 1.x ou 2.x de la bibliothèque. C'est pourquoi nous vous recommandons d'installer ce composant globalement et de le maintenir à jour.

```
npm update -g aws-cdk
```

Si vous devez travailler avec plusieurs versions du AWS CDK kit d'outils, installez une version spécifique du kit localement dans le dossier de votre projet.

Si vous utilisez TypeScript ou JavaScript, le répertoire de votre projet contient déjà une copie locale versionnée du kit d'outils CDK.

Si vous utilisez une autre langue, utilisez-le `npm` pour installer le AWS CDK Toolkit, en omettant le `-g` drapeau et en spécifiant la version souhaitée. Par exemple :

```
npm install aws-cdk@2.0
```

Pour exécuter un AWS CDK kit d'outils installé localement, utilisez la commande `npx aws-cdk` au lieu de uniquement`cdk`. Par exemple :

```
npx aws-cdk deploy MyStack
```

`npx aws-cdk`exécute la version locale du AWS CDK Toolkit s'il en existe une. Il revient à la version globale lorsqu'un projet n'a pas d'installation locale. Vous trouverez peut-être pratique de configurer un alias de shell pour vous assurer qu'il `cdk` est toujours invoqué de cette façon.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

(retour à la liste)

Lors du déploiement de ma AWS CDK pile, je reçois un **`NoSuchBucket`** message d'erreur

Votre AWS environnement n'a pas été amorcé et ne dispose donc pas d'un compartiment Amazon S3 pour stocker les ressources pendant le déploiement. Vous pouvez créer le bucket intermédiaire et les autres ressources requises à l'aide de la commande suivante :

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

Pour éviter de générer AWS des frais inattendus, aucun environnement AWS CDK ne démarre automatiquement. Vous devez amorcer explicitement chaque environnement dans lequel vous allez effectuer le déploiement.

Par défaut, les ressources bootstrap sont créées dans la ou les régions utilisées par les piles de l'application actuelle AWS CDK . Ils sont également créés dans la région spécifiée dans votre AWS profil local (définie par`aws configure`), en utilisant le compte de ce profil. Vous pouvez spécifier

un compte et une région différents sur la ligne de commande comme suit. (Vous devez spécifier le compte et la région si vous ne vous trouvez pas dans le répertoire d'une application.)

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

Pour plus d'informations, consultez the section called "Action d'amorçage".

(retour à la liste)

Lors du déploiement de ma AWS CDK pile, je reçois un **`forbidden: null`** message

Vous déployez une pile qui nécessite des ressources d'amorçage, mais vous utilisez un rôle ou un compte IAM qui n'est pas autorisé à y écrire. (Le bucket intermédiaire est utilisé lors du déploiement de piles contenant des actifs ou synthétisant un AWS CloudFormation modèle supérieur à 50 000.) Utilisez un compte ou un rôle autorisé à effectuer l'action sur `s3:*` le bucket mentionné dans le message d'erreur.

(retour à la liste)

Lors de la synthèse d'une AWS CDK pile, je reçois le message **`--app is required either in command-line, in cdk.json or in ~/.cdk.json`**

Ce message signifie généralement que vous n'êtes pas dans le répertoire principal de votre AWS CDK projet lorsque vous le publiez`cdk synth`. Le fichier `cdk.json` de ce répertoire, créé par la `cdk init` commande, contient la ligne de commande nécessaire pour exécuter (et ainsi synthétiser) votre AWS CDK application. Pour une TypeScript application, par exemple, la valeur par défaut `cdk.json` ressemble à ceci :

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

Nous vous recommandons d'émettre des `cdk` commandes uniquement dans le répertoire principal de votre projet, afin que la AWS CDK boîte à outils puisse `cdk.json` y accéder et exécuter correctement votre application.

Si cela n'est pas pratique pour une raison ou une autre, le AWS CDK Toolkit recherche la ligne de commande de l'application à deux autres emplacements :

- `cdk.json`Dans votre répertoire personnel
- Sur la `cdk synth` commande elle-même en utilisant l'`-a`option

Par exemple, vous pouvez synthétiser une pile à partir d'une TypeScript application comme suit.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

(retour à la liste)

Lors de la synthèse d'une AWS CDK pile, je reçois une erreur car le AWS CloudFormation modèle contient trop de ressources

AWS CDK Génère et déploie des AWS CloudFormation modèles. AWS CloudFormation impose une limite stricte au nombre de ressources qu'une pile peut contenir. Avec le AWS CDK, vous pouvez atteindre cette limite plus rapidement que prévu.

> **Note**
>
> La limite de AWS CloudFormation ressources est de 500 au moment d'écrire ces lignes. Voir AWS CloudFormation les quotas pour la limite de ressources actuelle.

Les constructions de niveau supérieur basées sur l'intention de la bibliothèque AWS Construct fournissent automatiquement toutes les ressources auxiliaires nécessaires à la journalisation, à la gestion des clés, à l'autorisation et à d'autres fins. Par exemple, l'octroi d'un accès à une ressource à une autre génère tous les objets IAM nécessaires à la communication des services concernés.

D'après notre expérience, l'utilisation dans le monde réel de constructions basées sur l'intention génère 1 à 5 AWS CloudFormation ressources par construction, bien que cela puisse varier. Pour les applications sans serveur, 5 à 8 AWS ressources par point de terminaison d'API sont typiques.

Les modèles, qui représentent un niveau d'abstraction supérieur, vous permettent de définir encore plus de AWS ressources avec encore moins de code. Le AWS CDK code inthe section called “ECS”, par exemple, génère plus de 50 AWS CloudFormation ressources tout en ne définissant que trois constructions !

Le dépassement de la limite de AWS CloudFormation ressources constitue une erreur lors de AWS CloudFormation la synthèse. Il AWS CDK émet un avertissement si votre pile dépasse 80 % de la limite. Vous pouvez utiliser une limite différente en définissant la `maxResources` propriété sur votre pile, ou désactiver la validation en la définissant `maxResources` sur 0.

> **Tip**
>
> Vous pouvez obtenir un décompte exact des ressources présentes dans votre sortie synthétisée à l'aide du script utilitaire suivant. (Comme chaque AWS CDK développeur a besoin de Node.js, le script est écrit dedans 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");
```

Lorsque le nombre de ressources de votre pile approche de la limite, envisagez de réorganiser l'architecture pour réduire le nombre de ressources que contient votre pile : par exemple, en combinant certaines fonctions Lambda ou en divisant votre pile en plusieurs piles. Le CDK prend en charge les références entre les piles, ce qui vous permet de séparer les fonctionnalités de votre application en différentes piles de la manière qui vous convient le mieux.

> **Note**
>
> AWS CloudFormation les experts suggèrent souvent l'utilisation de piles imbriquées comme solution à la limite des ressources. Ils AWS CDK soutiennent cette approche par le biais de la `NestedStack`construction.

(retour à la liste)

J'ai spécifié trois zones de disponibilité (ou plus) pour mon groupe Auto Scaling ou mon VPC, mais celui-ci n'a été déployé que dans deux

Pour obtenir le nombre de zones de disponibilité que vous demandez, spécifiez le compte et la région dans les `env` propriétés de la pile. Si vous ne spécifiez pas les deux, par défaut AWS CDK, la pile est synthétisée comme indépendante de l'environnement. Vous pouvez ensuite déployer la pile dans une région spécifique à l'aide de AWS CloudFormation. Certaines régions n'ayant que deux zones de disponibilité, un modèle indépendant de l'environnement n'en utilise pas plus de deux.

> **Note**
>
> Dans le passé, les régions se lançaient parfois avec une seule zone de disponibilité. Les AWS CDK stacks indépendants de l'environnement ne peuvent pas être déployés dans de telles régions. Au moment d'écrire ces lignes, toutes les AWS régions disposent toutefois d'au moins deux AZ.

Vous pouvez modifier ce comportement en remplaçant la propriété `availabilityZones`(Python :`availability_zones`) de votre pile pour spécifier explicitement les zones que vous souhaitez utiliser.

Pour plus d'informations sur la spécification du compte et de la région d'une pile au moment de la synthèse, tout en conservant la flexibilité nécessaire au déploiement dans n'importe quelle région, consultezthe section called “Environnements”.

(retour à la liste)

Mon compartiment S3, ma table DynamoDB ou toute autre ressource n'est pas supprimé lorsque je publie **`cdk destroy`**

Par défaut, les ressources qui peuvent contenir des données utilisateur ont une propriété `removalPolicy` (Python :`removal_policy`) de`RETAIN`, et la ressource n'est pas supprimée lorsque la pile est détruite. Au lieu de cela, la ressource devient orpheline de la pile. Vous devez ensuite supprimer la ressource manuellement une fois la pile détruite. Tant que vous ne le faites pas, le redéploiement de la pile échoue. Cela est dû au fait que le nom de la nouvelle ressource créée lors du déploiement est en conflit avec le nom de la ressource orpheline.

Si vous définissez la politique de suppression d'une ressource sur`DESTROY`, cette ressource sera supprimée lorsque la pile sera détruite.

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 Impossible de supprimer un compartiment Amazon S3 non vide. Si vous définissez la politique de suppression d'un compartiment Amazon S3 de manière à `DESTROY` ce qu'il contienne des données, toute tentative de destruction de la pile échouera car le compartiment ne peut pas être supprimé. Vous pouvez demander à

AWS CDK supprimer les objets du compartiment avant de tenter de le détruire en réglant l'`autoDeleteObjects`accessoire du compartiment sur`true`.

(retour à la liste)

Clés OpenPGP pour et jsii AWS CDK

Cette rubrique contient les clés OpenPGP actuelles et historiques pour AWS CDK et jsii.

Clés actuelles

Ces clés doivent être utilisées pour valider les versions actuelles de AWS CDK et jsii.

AWS CDK Clé OpenPGP

Identifiant de la clé :	0x42B9CF2286CD987A
Type :	RSA
Taille :	4096/4096
Créé :	05.07-05
Expire :	2026-07-04
ID utilisateur :	Kit de développement AWS Cloud < aws-cdk@amazon.com >
Empreinte digitale clé :	69B5 2D5B A295 1D11 FA65 413B 42B9 CF22 86CD 987A

Sélectionnez l'icône « Copier » pour copier la clé OpenPGP suivante :

```
-----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-----
```

clé OpenPGP jsii

Identifiant de la clé :	0x056C4E15DAE3D8D9
Type :	RSA
Taille :	4096/4096
Créé :	05.07-05
Expire :	2026-07-04
ID utilisateur :	L'équipe AWS JSII < aws-jsii@amazon.com >
Empreinte digitale clé :	1E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

Sélectionnez l'icône « Copier » pour copier la clé OpenPGP suivante :

```
-----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-----
```

Clés historiques

Ces clés peuvent être utilisées pour valider les versions de AWS CDK et jsii avant le 5 juillet.

> **Important**
>
> Les nouvelles clés sont créées avant que les précédentes n'expirent. Par conséquent, à tout moment, plusieurs clés peuvent être valides. Les clés sont utilisées pour signer les artefacts dès leur création. Utilisez donc la clé émise la plus récemment lorsque la validité des clés se chevauche.

AWS CDK Clé OpenPGP (04-04-07)

> **Note**
>
> Cette clé n'a pas été utilisée pour signer AWS CDK des artefacts après le 5 juillet.

Identifiant de la clé :	0x015584281F44A3C3
Type :	RSA
Taille :	4096/4096
Créé :	07/04/2018
Expire :	06/04/2026/
ID utilisateur :	Kit de développement AWS Cloud < aws-cdk@amazon.com >
Empreinte digitale clé :	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

Sélectionnez l'icône « Copier » pour copier la clé OpenPGP suivante :

```
-----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-----
```

clé OpenPGP jsii (04-04-07)

> **Note**
>
> Cette clé n'a pas été utilisée pour signer les artefacts jsii après le 5 juillet.

Identifiant de la clé :	0x985F5BC974B79356
Type :	RSA
Taille :	4096/4096
Créé :	07/04/2018
Expire :	06/04/2026/
ID utilisateur :	L'équipe AWS JSII < aws-jsii@amazon.com >
Empreinte digitale clé :	35A7 1785 8FA6 282D C5AC CD95 985F 5BC9 74B7 9356

Sélectionnez l'icône « Copier » pour copier la clé OpenPGP suivante :

```
-----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 Clé OpenPGP (2018-06-19)

Identifiant de la clé :	0x0566A784E17F3870
Type :	RSA
Taille :	4096/4096
Créé :	19/06/2018
Expire :	18/06/2018
ID utilisateur :	L'équipe AWS CDK < aws-cdk@amazon.com >

Empreinte digitale clé :	E88B E3B6 F0B1 E350 9E36 4F96 0566 A784 E17F 3870

Sélectionnez l'icône « Copier » pour copier la clé OpenPGP suivante :

```
-----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-----
```

clé OpenPGP jsii (2018-08-06)

Identifiant de la clé :	0x1C7ACE4CB2A1B93A
Type :	RSA

Taille :	4096/4096
Créé :	2018-08-06
Expire :	05.08-05
ID utilisateur :	L'équipe AWS JSII < aws-jsii@amazon.com >
Empreinte digitale clé :	85EF 6522 4CE2 1E8C 72 DB 28EC 1C7A CE4C B2A1 B93A

Sélectionnez l'icône « Copier » pour copier la clé OpenPGP suivante :

```
-----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 Historique du guide du développeur

Voir Releases pour plus d'informations sur AWS CDK les versions. AWS CDK Il est mis à jour environ une fois par semaine. Des versions de maintenance peuvent être publiées entre les versions hebdomadaires pour résoudre des problèmes critiques. Chaque version inclut un AWS CDK kit d'outils (CLI CDK), une bibliothèque de AWS construction et une référence d'API correspondants. Les mises à jour de ce guide ne sont généralement pas synchronisées avec AWS CDK les versions.

> **Note**
>
> Le tableau ci-dessous présente les principales étapes de la documentation. Nous corrigeons les erreurs et améliorons le contenu de manière continue.

Modification	Description	Date
Ajouter de la documentation pour la fonctionnalité CDK Migrate	Utilisez la AWS CDK CLI `cdk migrate` commande pour migrer les AWS ressources déployées, les AWS CloudFormation piles déployées et les AWS CloudFormation modèles locaux vers AWS CDK. Pour plus d'informations, consultez la section Migrer vers AWS CDK.	2 février 2024
Mises à jour des bonnes pratiques IAM	Mise à jour du guide s’aligner sur les bonnes pratiques IAM. Pour plus d’informations, consultez Bonnes pratiques de sécurité dans IAM.	23 mars 2023

Document `cdk.json`	Ajoutez de la documentation sur les valeurs `cdk.json` de configuration.	20 avril 2022
Gestion des dépendances	Ajoutez une rubrique sur la gestion des dépendances avec le AWS CDK.	7 avril 2022
Supprimer les doubles accolades dans les exemples Java	Remplacez cet anti-modèle par Java 9 `Map.of` dans son intégralité.	9 mars 2022
AWS CDK version v2	La version 2 du guide du AWS CDK développeur est publiée. Historique du document pour CDK v1.	4 décembre 2021

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.